Querying Cyclic Databases in Natural Language

by

Gary W. Hall

B.A. Simon Fraser University, Burnaby

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

SIMON FRASER UNIVERSITY
September 1986

Approval

Name:	Gary W. Hall
Degree:	Master of Science
Title of Thesis:	Querying Cyclic Databases in Natural Language

Examining Committee:

Chairperson: Dr. Joseph Peters

Dr. Nick Cercone
Senior Supervisor

[illegible]
Dr. Wo-Shun Luk
Senior Supervisor

Dr. Arthur L. Liestman

[illegible]
Dr. James Delgrande
External Examiner

August 15th, 1986
Date Approved

Title of Thesis/Project/Extended Essay

QUERYING CYCLIC DATABASES IN

NATURAL LANGUAGE

Author: [illegible]

(signature)

GARY HALL

(name)

Sept 4, 1986

(date)

Abstract

Knowledge of the logical structure of a computer database is necessary for efficient access to information in the database. It is unlikely that naive users of these systems would possess such knowledge. Thus, providing access to database systems to relatively unsophisticated users requires that "logical data independence" be incorporated into database systems, that is, such systems must be able to interpret queries which do not completely specify the access path to the desired data. The system must solve the "Multiple Access Path Problem" (MAPP), choosing the correct access path from a number of possible candidate paths, in order to provide logical data independence. Several systems which use highly ambiguous formal query languages make unrealistically restrictive assumptions about the database in order to solve the MAPP.

Accessing database information through natural language queries is a promising approach to logical data independence. Research into natural language interface systems indicates that syntactic and semantic analysis of natural language queries often provides an incomplete specification of the access paths to the requested data. Sections of the path must be filled in by the database system. Current natural language interfaces to databases are able to solve this constrained MAPP only for some *acyclic* databases. Further customization of the system to the application is not a desirable solution to this problem because the amount of customization required for practical applications is too great.

A method is provided to solve the MAPP in the context of natural language interfaces to *cyclic* databases. This method avoids the restrictive assumptions of the formal systems without requiring an unreasonable customization effort. We show that the access paths which are solutions to the constrained MAPPs that occur with natural language interfaces have a characteristic semantic structure. A heuristic method is developed which makes reasonable guesses of correct access paths based on this characteristic structure. A representation of the semantic structure of the database is presented and an algorithm is provided to map an incomplete specification of an access path into this database representation and derive a complete representation of the path.

Acknowledgements

I would like to thank my wife, Margarita Berrios, for her love and inspiration which make all things possible for me. In addition, I would like to thank my two senior supervisors, Dr. Nick Cercone and Dr. WoShun Luk, for their generosity and patience, Dr. Arthur Liestman, who pointed out a number of problems in the work and helped find solutions to them, and Dr. James Delgrande, for his careful reading and many helpful suggestions. I am also grateful to Dr. Stanley Petrick of the Thomas J. Watson Research Laboratory, who took the time to personally answer a number of inquiries and supplied a great deal of material about the TQA project. I would like to acknowledge the financial support of the School of Computer Science and the Senate Committee on Scholarships, Awards and Bursaries of Simon Fraser University, and of the Natural Sciences and Engineering Research Council of Canada.

Table of Contents

List of Figures

Chapter 1
Introduction

1.1. Motivation

In recent years much work has been done designing and implementing natural language interfaces (NLIs) to databases (DBs). The motivation behind this work has been twofold. Database researchers have long tried to make databases accessible to users who are naive about computers in general and databases in particular. Creating database systems that understand natural language (NL) is one approach to increasing accessibility for unsophisticated users. Researchers into the problem of Natural Language Understanding (NLU) recognized that narrowing the domain in which natural language discourse takes place makes the problem much more tractable. Thus natural language researchers have studied and built natural language interfaces to databases which typically represent small, well-defined domains.

Recent research into databases focuses on the relational model (Codd, 1970), which is easier to use than the other major models of database design - the network and hierarchical models (Ullman, 1982a). One area of research into relational databases which is of particular relevance to the problem of creating natural language interfaces is the *query inference problem*. Simply stated, the query inference problem is the problem of translating an ambiguous query, expressed in some query language (QL) which permits such queries, into an unambiguous representation which can be used to manipulate the database (Wald and Sorenson, 1984). The query language which has received the most attention in this regard is one which allows queries to be formulated without explicit mention of every base relation and join[1] required to produce the correct response. This type of query language may be described as an *implicit tuple calculus* (ITC). With ITC queries there is usually a number of different possible ways to combine the attributes, relations and/or joins which are explicitly mentioned in the query. Choosing the correct one from among these alternative "paths" through the database is known as the Multiple Access Path Problem (MAPP). Two different approaches to the solution of this problem in the context of ITC systems are described in (Ullman, 1982a) and (Wald and Sorenson, 1984).

The implications of this work on the MAPP for natural language interface research seem obvious. Natural languages are notoriously ambiguous and methods to resolve ambiguity in database queries can relate to natural language queries to databases. A great deal of research has been done to resolve ambiguity of natural language statements via syntactic analysis and semantic interpretation. See (Barr and Feigenbaum, 1981) for an overview of computational approaches. Natural language queries to databases must undergo such analysis and interpretation before they can be expressed in a logical form (LF) containing the names of attributes, attribute values and relations similar to the form of ITC queries. The question naturally arises: how

[1]Base relations are the basic data structures of relational databases and join is an operation which combines these basic structures in order to relate the data in one base relation to the data in other base relations. See Appendix A for a presentation of the terms and basic theory of relational data bases.

much ambiguity is left to resolve after natural language queries have been translated to such a logical form? In particular, it is important to discover to what extent the MAPP is present when natural language logical forms are to be used to access a database. The richer semantics of natural language in comparison to ITC suggests that in general, given effective syntactic analysis and semantic interpretation, natural language logical forms will be less ambiguous than ITC queries. If this is indeed the case, the methods used to resolve the MAPP for ITC queries may not be appropriate for natural language queries.

1.2. The Task

We consider the extent to which the MAPP arises in the context of natural language interfaces and come to a decision on the best method for its solution. The problem of choosing from among alternate paths becomes significantly more difficult when the database in question is *cyclic*.

To assess the extent to which the MAPP itself arises in natural language queries it is necessary to analyze how deficiencies in database queries lead to the MAPP and assess to what extent these deficiencies will appear in natural language logical forms. Data from current natural language interface systems, for example, the Transformational Query Answering (TQA) system (Petrick, 1984), which have been in use in practical applications, will be helpful in making this assessment of the natural language logical form. Analyzing the nature of the interaction between a human user and a natural language interface system can help explain such data and lead to generalizations which should apply to other natural language interface systems. By comparing the human/machine interaction with human/human linguistic interaction, for which there exists a great deal of analysis from philosophy of language and linguistics, progress toward a suitable understanding may be made.

To decide on the best methods for resolving the MAPP in the context of natural language interface systems, current approaches to the problem in ITC systems are examined. Alternative approaches need to be designed and evaluated. A comparison between the various alternatives can thus be made.

Chapter 2

The Multiple Access Path Problem

2.1. Defining the Problem

2.1.1. Data Manipulation Languages

Each relational database management system (DBMS) provides a language in which users may formulate instructions to the system in order to manipulate the database. To some extent these *data manipulation languages* (DMLs) are unique to each system. The subset of a DML which is used to retrieve data is called a *query language*. There are three abstract query languages which are equivalent to each other in expressive power and which represent the minimum capability of any reasonable relational database query language (Codd, 1972). These query languages include: the *relational algebra*, the *tuple relational calculus*, and the *domain relational calculus*. In practice query languages are usually blends and/or extensions of one or more of these abstract languages.

A DML allows the user to retrieve, insert or modify data by creating expressions in terms of the conceptual database; thus DMLs provide physical data independence. However, to be correctly formulated, query expressions in each of the abstract languages must be *logically complete*, that is, they must provide a full, unambiguous account of a correct process for retrieving the data, expressed in terms of the conceptual (logical) database.

2.1.2. Data Retrieval

The basic strategy for data retrieval in a relational database is to associate, via a natural join, the relations which contain the attributes for which values are known and those which contain the attributes whose values are desired, thus making accessible desired unknown values which correspond to known values. If two relations, R and S, are not connected, (do not share an attribute), they can only be associated by a natural join which includes other relations such that there is a series of connections which form a *path* between them.

SL = {<u>STUD#</u>, SNAME}
CL = {<u>CLASS, STUD#</u>, GRADE}
IL = {<u>CLASS</u>, INAME}

Figure 2-1: The COURSEGRADE Database Scheme

For example, the database scheme presented in Figure 2-1 is composed of three relational schemes. Following standard practice, we underline the primary key in a relation. The relations SL and IL do not share any attributes. Although SL and IL could be joined on the attributes SNAME and INAME, (to find a student who was taught by an instructor of the same name, for example), they can only be associated by a <u>natural</u> join if the relation CL is included in the operation. Part of any logically complete specification of a process to retrieve data from a relational database must include the naming of those relations which are on the path that connects the relations containing the known and unknown values. In addition, the formulator of a query must specify which attributes form the connections between the relations which are to participate in the natural join. Let the precise specification of the relations and connections which are required to answer a query Q be called the *access path* for

Q. Suppose a user of the COURSEGRADE database wished to know the names of the students of Professor Jones. "Jones" is a value of the attribute INAME in the relation IL which constrains the values of the attribute SNAME of the relation SL which must be returned in the answer. A specification of the access path to associate SL and IL must include references to the relation CL, the connection between SL and CL (the attribute STUD#), and the connection between CL and IL (the attribute CLASS). Thus a user who wishes to retrieve data using the DML must have an intimate knowledge of the conceptual database (or that part of it which comprises his particular user view) as well as the syntax of the DML and the operations that the DML describes.

2.1.3. Logical Data Independence

A goal of current research into relational DBMS's is to create systems which obviate the need for users to know the logical structure of the conceptual database. Such systems would thus provide *logical data independence*. Two proposals which free users from knowing connections between relations are detailed in (Ullman, 1982a) and (Wald and Sorenson, 1984). Both contain correspondingly simplified query languages which, nevertheless, require the user to know attribute and/or relation names and which have a somewhat rigid and unnatural syntax. For example, the query in the section above regarding the names of the students of Professor Jones would be expressed in the query language of System/U (Ullman, 1982a) as follows.

Retrieve(SNAME) where FNAME = "Jones" (2.1)

The system must discover the access path to retrieve the information that the user desires; all the user has specified is the type of attribute values which are to make up the answer and the value of the attribute which constrains the answer.

Discovering the access path for ambiguous queries of this type is known as the *query inference problem* (Wald and Sorenson, 1984). The query inference problem is to translate an ambiguous representation of a query into a correct unambiguous representation.

2.1.4. The UNIVERSITY Database

A simple university database is now described. The data for the UNIVERSITY database can be defined informally as follows.

1. Each student is represented uniquely by a student number. Each student number uniquely determines the name, major, minor and sex of the student it represents.

2. Each faculty member is represented by a faculty number. Each faculty number uniquely determines the name and sex of the faculty member it represents.

3. Courses are represented by course names. Each course is offered by only one department. Courses are worth credit of a specific number of units.

4. Classes are represented by unique class identifier, for example, m100862s1 is the name for the first section of the Math 100 class offered in the second term of 1986. Each class has one instructor (faculty member) and is one of a possible number of offerings of a course during one term.

5. Students attend many classes; classes contain many students. For each student in each class there is a final grade.

6. Departments employ many faculty members; faculty members are appointed to one or more departments.

The formal definition of the relation schemes in the database is given in Figure 2-2.

STUDENT = {$\underline{\text{STUD\#}}$, SNAME, MAJOR, MINOR, SSEX}
FACULTY = {$\underline{\text{FAC\#}}$, FNAME, FSEX}
COURSE = {$\underline{\text{CNAME}}$, DEPT, UNITS}
CLASS = {$\underline{\text{CLID}}$, CNAME, TERM, FAC#}
GRADES = {$\underline{\text{CLID}}$, $\underline{\text{STUD\#}}$, GRADE}
APPOINT = {$\underline{\text{FAC\#}}$, $\underline{\text{DEPT}}$}

Figure 2-2: The UNIVERSITY DB Scheme

2.1.5. Natural Join Graphs

We introduce a graphical representation of the connections in a database, called a *natural join graph* (NJ-graph), which is useful for plotting access paths in the database. Let **R** = $\{R_1, R_2, \ldots, R_p\}$ be a database scheme over **U**. The *complete intersection graph* for **R** is the complete undirected graph on nodes $R_1, R_2, \ldots, R_p$ with edge labels chosen from the subsets of **U**. For an edge $e = (R_i, R_j)$, $1 \leqslant i < j \leqslant p$, the label of e, denoted $L(e)$, is $R_i \cap R_j$. A *NJ-graph* for **R** is the complete intersection graph for **R** with any edge e removed where $L(e) = \varnothing$.

An NJ-graph is a graph (V,E) where each vertex in V represents a unique relation in the corresponding database, **R**, and every relation in **R** is represented by exactly one vertex in **V**. Each vertex in **V** is labelled with the name of its corresponding relation. For each pair of relations in **R** which share one or more attributes there is an edge between the corresponding vertices in the NJ-graph. This edge is labelled with the combined names of its corresponding attributes. Thus every natural join permitted by the database structure is represented by a unique edge in the NJ-graph, including its adjoining vertices.

Figure 2-3 depicts the NJ-graph for the UNIVERSITY database scheme given in

Figure 2-2. The nodes in the graph represent the relations in the database. Note that the relation names have been shortened for convenience in labelling the nodes. The edges represent the shared attributes; for example, the STUDENT (ST) relation and the GRADES (GR) relation share the STUD# attribute and the ST and GR nodes are linked by an edge labelled "stud#".

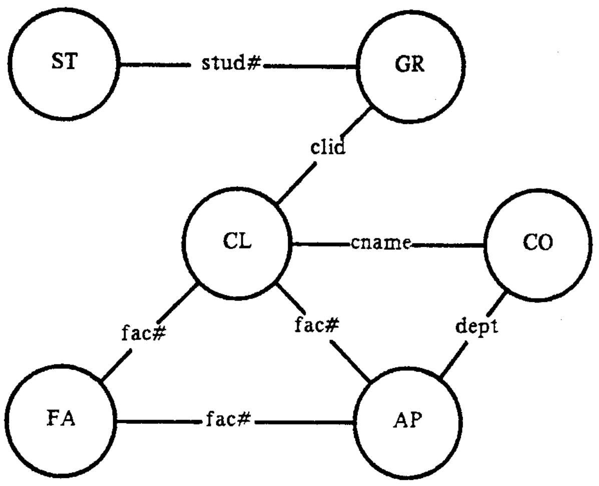


Figure 2-3: The UNIVERSITY DB NJ-graph

2.1.6. The Problem

It is easy to see that for every path in the NJ-graph there is a corresponding access path in the database, in which the relations corresponding to the nodes in the path are joined on the attributes corresponding to the edges in the path. We will simply use the term "path" if there is no need to distinguish between the type of path under discussion. Since an NJ-graph is an undirected graph and edges may be traversed more than once in a path through an undirected graph, in a NJ-graph there are an infinite number of paths between any two nodes. Thus there are an infinite number of access paths connecting any set of relations in the corresponding database. Therefore, given an ambiguous query which simply specifies, either directly or indirectly, a proper subset of the relations in the access path for that query, the number of candidate access paths is theoretically infinite. How to choose the access path that will produce the correct answer is the Multiple Access Path Problem (MAPP).

2.2. The Minimum Connection Strategy

The relational scheme for the SIMPLE/U database is given in Figure 2-4. The SIMPLE/U database is a slightly simplified UNIVERSITY database with a few attributes and one relation omitted.

STUDENT = {STUD#,SNAME}
FACULTY = {FAC#,FNAME}
CLASS = {CLID,FAC#}
GRADES = {CLID,STUD#,GRADE}
APPOINT = {FAC#,DEPT}

Figure 2-4: The SIMPLE/U DB Scheme

Figure 2-5 represents the NJ-graph for the SIMPLE/U database.

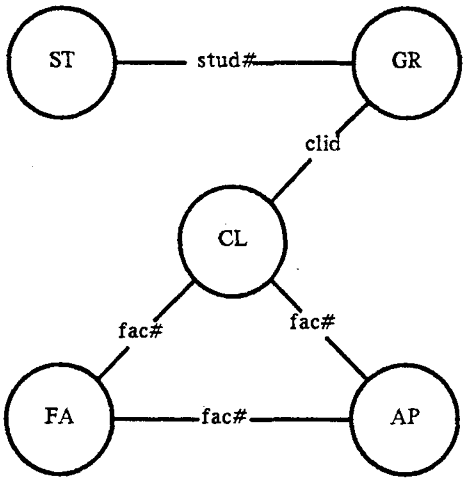


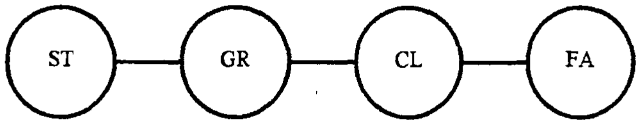
Figure 2-5: The SIMPLE/U DB NJ-graph

2.2.1. The Strategy

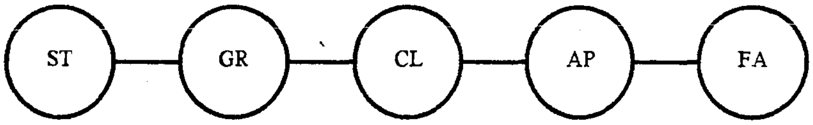
Recall Query (2.1), repeated below for convenience, about the names of Professor Jones' students, and suppose that (2.1) was to be applied to the SIMPLE/U database.

Retrieve(SNAME) where FNAME = "Jones"

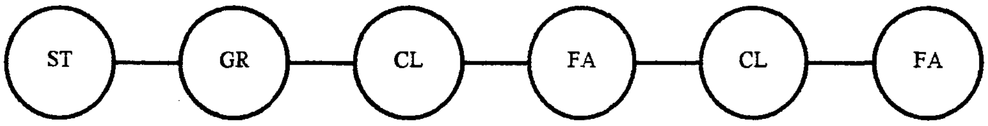
Figure 2-6 illustrates three of the infinite number of possible paths through the SIMPLE/U NJ-graph that include the STUDENT and FACULTY relations indicated by the attributes SNAME and FNAME contained in the query. A possible interpretation is included with each path.



Which students are taught by professors named Jones?
(a)



Who is taught by appointed professors named Jones?
(b)



Who is taught by professors who teach the classes Jones teaches?
(c)

Figure 2-6: Possible Paths for Query (2.1)

The MAPP for Query (2.1) is to decide which of these or the any number of other possible interpretations matches the question that the user had in mind when she formulated the query. There is no way of determining which interpretation is correct without asking the user directly. A reasonable guess, based on the structure of the database and the information available in the input query, is desired. One strategy for making such a reasonable guess is to choose the path with the minimum connection (Ullman, 1982a), that is, the path requiring as few or fewer joins as any candidate path. Let us call this strategy the Minimum Connection Strategy (MCS). The MCS is based on two assumptions:

1. Queries are more frequently simple, direct, and short rather than long, indirect, and circuitous.

2. Removing dangling tuples from the answer is normally not desirable.[2]

Let path P_1, connecting a set of relations R in a database, be such that there is no other path P_2 connecting the relations in R that has fewer joins than P_1. This path P_1 is called a *minimum path* for R. Path (a) in Figure 2-6 is the minimum path for the set {STUDENT,FACULTY} through the SIMPLE/U database and will therefore be the path that is chosen by the MCS in response to Query (2.1). Path (a) is obviously the least circuituous of the three paths and represents the least complex of the queries, where complexity is measured in terms of the grammatical form of the sentences expressing the queries.

[2]A *dangling tuple* is a tuple in a relation R which contains a value v for attribute A, while there exists another relation S which has A as one of its attributes but does not contain v as a value for A in any of its tuples. The second tuple in R2 below is dangling. Note that there is not a tuple in the join of R1 and R2 corresponding to the dangling tuple in R2.

R1

A	B
a1	b1

R2

B	C
b1	c1
b2	c2

R1$\bowtie$R2

A	B	C
a1	b1	c1

The following discussion explains the intuition behind the second assumption. Suppose the university allowed professors not employed by some department to teach - visiting professors for example. In this case, there could be dangling tuples in the FACULTY relation. Path (b) would eliminate these tuples from the answer. This is not normally desirable because the query indicates an interest in students taught by one or more professors named Jones, not whether the professor is a member of any department.

Thus, unless a relation is mentioned in a query or is on a minimum path which connects the relations which are mentioned, our assumption is that it does not belong in the query.

2.2.2. Acyclic vs Cyclic Databases

The MAPP is a more difficult problem to solve for a certain class of databases which are known as cyclic databases. After presenting a definition of acyclicity and cyclicity we will discuss how well the MCS performs on the MAPP for these two classes.

There are a number of different characterizations of acyclic database schemes due to various authors (Maier, 1983). We present the one that is most intuitive from the point of view of the NJ-graph.

Definition 1: Let $\mathbf{R} = \{R_1, R_2, ..., R_p\}$ be a database scheme over the set of attributes $\mathbf{U}$. Let G be a subgraph of the complete intersection graph for $\mathbf{R}$ and let $A \in \mathbf{U}$. A path $e_1, e_2, ..., e_k$ from node R_i to node R_j in G is an A-*path* if $A \in L(e_i)$ for all $1 \leqslant i \leqslant k$. It follows that A must be in every node R along the A-path. G is a *join graph* if, for every pair of nodes R_i, R_j in G, if $A \in (R_i \cap R_j)$ then there is an A-path from R_i to R_2. A *join tree* is a join graph that is a tree.

A database scheme **R** is *acyclic* iff it has a join tree. A database with an acyclic database scheme is an *acyclic database.* A *cyclic database scheme* has no join tree. A database with a cyclic database scheme is a *cyclic database.*

The SIMPLE/U database scheme is acyclic. Figure 2-7 represents the join tree for the SIMPLE/U database scheme.

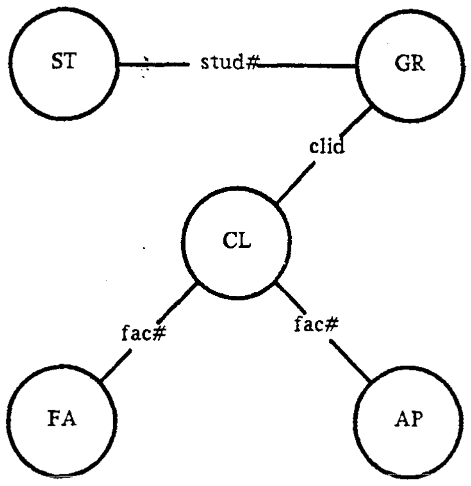


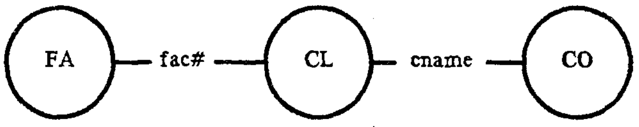
Figure 2-7: The SIMPLE/U DB Join Tree

A cursory examination of the NJ-graph in Figure 2-3 confirms that there is no join tree for the UNIVERSITY database scheme; therefore this scheme is cyclic.

The MCS is able to find an unique reasonable path connecting a set of attributes when the database scheme is acyclic but is not able to do so if the scheme is cyclic (Ullman, 1982a). Consider the following query to the cyclic UNIVERSITY database.

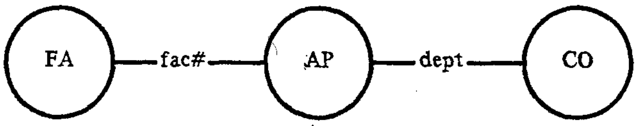
Retrieve (FNAME) where UNITS=3 (2.2)

There are two minimum paths connecting the attributes mentioned in this query, shown with their interpretations in Figure 2-8.



Who teaches 3 credit courses?

(a)



Who works in depts that offer 3 credit courses?

(b)

Figure 2-8: Minimum Paths for Query (2.2)

Although the latter English query is more complex grammatically than is the former, this is not reflected in the number of connections in their corresponding paths. Clearly the two paths may lead to different results. Thus the MCS alone fails to provide a unique reasonable guess of the correct access path for Query (2.2).

2.3. Previous Approaches to Solving the MAPP

In an early work on the MAPP problem, (Carlson & Kaplan, 1976), three approaches to its solution were provided; a) design the DB so that all access paths are identical, b) rename attributes until ambiguity is removed, and c) query the user. There are problems with each of the approaches. The first approach is that taken by those who make the universal relation assumption discussed below. The second approach, attribute renaming, can lead to a proliferation of attribute names which obviates logical data independence (Kent, 1981). The third approach must be part of any natural language interface system since dialogue with the user is required to verify that the correct interpretation of the query has been made, at least in those cases where there are closely competing candidate paths. However, the most likely candidates must be computed automatically if the user is not to be overburdened. Thus user dialogue cannot be a total solution.

A number of authors base solutions to the MAPP on one or more of the various universal relation (UR) assumptions (Osborn, 1979, Ullman, 1982a, Sagiv, 1981). The implementation of one of these solutions, known as System/U, has been described in (Ullman, 1982a). An overview of the universal relation assumptions is given in (Ullman, 1982b). All of these assumptions have been called into question (Atzeni and Parker, 1982). The weakest of the universal relation assumptions is called the Universal Relation Scheme Assumption (URSA). The URSA holds that sufficient renaming of attributes has occurred that a unique relationship exists among any set of attributes (Ullman, 1982b). All of the universal relation approaches cited above make the URSA.

We believe that even the URSA, is too restrictive. Consider the UNIVERSITY database scheme of Figure 2-2. Within this scheme there are two distinct relationships represented by {FAC#, DEPT}. In one, faculty are the appointees of departments; in the other, faculty teach in departments. While one of these attributes could be renamed to "correct" the situation, this does not seem reasonable in general, due to the proliferation of attribute names. Also, in the context of a natural language interface users do not know any database attribute names. A natural language question from such a user may be unclear as to the role of a particular attribute set. For example, the question

"Give me the names of the Math professors"

gives no indication of which of the relationships between faculty members and departments is involved. In cases such as this, all possibilities must be considered and attribute renaming would be to no avail.

Another approach to the solution of the MAPP is to use the Entity-Relationship (ER) model (Zhang, 1983, Wald and Sorenson, 1984). This approach avoids the proliferation of attribute names because the model represents different relationships between attributes by different paths in the database. The solution proposed by Wald and Sorenson has been implemented in a system called Verdi. It provides an efficient solution to the MAPP only for a subset of database schemes.

All of the previous approaches assume or use an interface that requires the user to input the names and values of the attributes as they appear in the database using a query language with a rigid syntax. In Chapter 3 we examine how the MAPP presents itself in the context of a natural language interface. In Chapter 4 we analyze the relationship between a certain class of natural language predicates and virtual relations in databases, with an eye towards constraining the MAPP in the context of natural language interface systems in order to make the MAPP more tractable. In Chapter 5 the results from the analysis in Chapter 4 are used to develop an approach to the MAPP in the context of natural language interfaces which avoids the restrictive assumptions of the universal relation approaches and is more broadly applicable than that of the ER approaches.

Chapter 3

Natural Language Interfaces and the MAPP

We begin with brief review of some issues relevant to the topic of the MAPP in natural language interfaces. We examine, in depth, the Transformational Question Answering (TQA) System (Petrick, 1984), a state-of-the-art example of a natural language interface to a relational database, to determine how the MAPP presents itself in such a system. We then discuss whether the TQA methods and results are generalizable to other natural language interfaces.

3.1. Introduction to Natural Language Interfaces

The systems mentioned in Chapter 2 are designed to provide users of databases a measure of logical data independence. Users of these systems are not required to know the connections between relations and the burden of computing access paths is placed onto the system. Users are typically required to know database attribute names and values as well as the syntax of the query language used to query the system. A natural language interface to a database would remove these remaining requirements. Users of a natural language interface could specify the information they desire in their own terms in their own language. For a discussion of the advantages of natural language interfaces see (Pylyshyn and Kittredge, 1985).

3.1.1. Basic Natural Language Interface Structure

Any database system that is able to respond satisfactorily to natural language queries must be able to understand those queries at some level. Basic to understanding of natural language sentences is an understanding of the words that make up the language. All natural language interface systems must make use of a lexicon containing some subset of the words in the natural language. Attached to each word in the lexicon is information used by the system to assess its meaning when encountered in the input. Some of the information attached to each word is grammatical. For example, "teach" may be identified as a transitive verb, which requires (or prefers) faculty members as subjects and classes as objects. Some of the information attached to each word is application-specific. For example, the entry for "teacher" in the lexicon of a natural language interface system which interfaces the UNIVERSITY database will include the information that it refers to the FAC# attribute. The lexical information is used to create a structure which represents the meaning of the input sentences to the system. This process involves syntactic analysis of the grammatical relationships among the words and semantic analysis of the meaning relationships among the words. The meaning structure is then converted into a query language expression. Figure 3-1, adapted from (Damerau, 1985), shows minimal structure for a natural language interface to a database. The subsystem that reads the lexical input looks up each word read in the lexicon and attaches to each input word the relevant information discovered in its lexical entry. Then the input which has been augmented by the lexical information is translated into the query language of the DBMS.

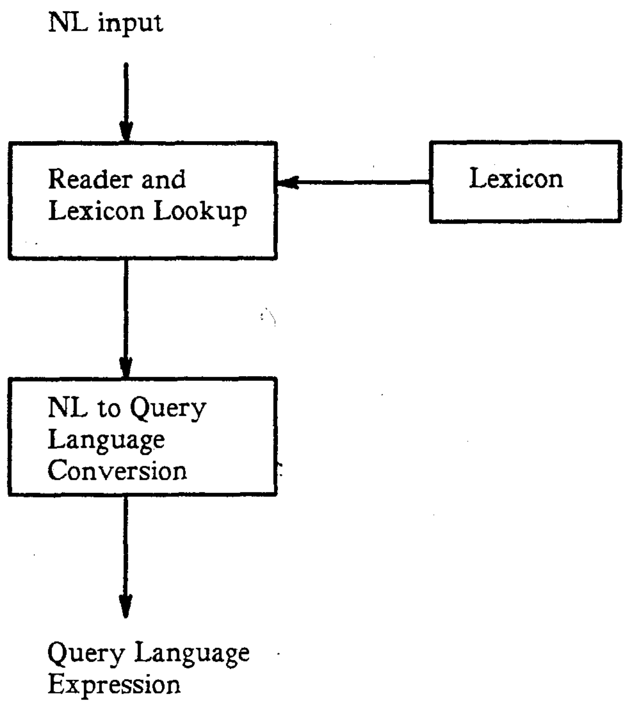

NL input
Reader and Lexicon Lookup
Lexicon
NL to Query Language Conversion
Query Language Expression

Figure 3-1: Minimal Structure of a Natural Language Interface

3.1.2. Customization and Portability

The process of building that part of the lexicon which is specific to the desired application and fine-tuning the translation subsystem to the specific requirements of that application is called *customization*. Customization is required by all natural language interface systems. Customization requirements can make a natural language interface system impractical or even impossible for many applications (Petrick, 1984). Thus it is important that these requirements be carefully considered when designing a natural language interface system. Recently much effort has been directed towards devising systems which are portable from one application to another (Hafner and Godden, 1985, Thompson and Thompson, 1985, Damerau, 1985, Martin and Appelt, 1985, Ballard and Lusth, 1985). Minimizing customization requirements makes systems more transportable. Since portable systems require less effort to customize they are less expensive to set up for any given application and therefore are more marketable.

Building the application specific lexicon is one of the major tasks in the customization process. Minimizing this aspect of customization means minimizing the amount and detail of the information that is attached to each entry. With less information available from the lexicon, a greater burden is placed on the translation subsystem. In Section 3.2 we discuss how reducing the requirements for customization of the lexicon resulted in the introduction of the MAPP to the TQA system.

3.1.3. Logical Form

Typically the interpretation of a natural language question proceeds through a series of transformations. At the final stage before formulation of the query in the database query language, all of the relevant information extractible from the natural language input is expressed in a structure (possibly) using database terms. This final structure, although often referred to as a *logical form*,[3] is not capable, in current working systems, of being used for making inferences. If some information about the necessary relations and connections required to extract the desired data is missing from this final form, the part of the system which is to formulate the database query is faced with the MAPP. In the next section we examine the problem as it appears in the TQA system in order to establish the nature of the MAPP in the context of natural language interfaces.

[3]This use of the term "logical form" is different from that in linguistics.

3.2. The TQA System

TQA uses its knowledge of English syntax and semantics to parse natural language input into composite substructures. These substructures represent base and virtual relations in the database. The system is powerful enough to deduce the connections between the substructures and, thereby, the connections needed to join their corresponding relations. The MAPP is thus reduced to those substructures corresponding to virtual relations. In this section we examine how TQA works; in the next section we discuss how TQA's success results from certain principles underlying natural language discourse which make TQA's methods and results generalizable. In Chapter 4 we examine the semantic relationship of the composite substructures of the natural language queries and their corresponding virtual relations in the database. This semantic connection leads to a useful solution to the MAPPs faced when translating substructures corresponding to virtual relations into access paths which yield those virtual relations (Chapter 5).

3.2.1. Description of the System

TQA is one of several state-of-the-art natural language interface systems designed to interface with a relational database. The DBMS to which TQA interfaces is the IBM product SQL/DS. TQA has been under development at the IBM Thomas J. Watson Research Center since the early 1970's, when it was known as REQUEST (Petrick, 1973).

TQA consists of four parts:

1. A Customization Program which can interact with a database expert and inspect the database (Damerau, 1985).

2. A Natural Language Processor which translates English questions to database queries (Johnson, 1984).

3. A Query Paraphrase Generator which validates system interpretations by feeding a English paraphrase of the translated query back to the user.

4. An Answer Processor which formats the response.

Since we are primarily concerned with the MAPP in query translation we consider most closely the natural language processor and those aspects of the customization program relevant to this problem. However, we consider validation of interpretations to be an important step in the generation of accurate responses to natural language questions.

Figure 3-2, from (Johnson, 1984), is a representation of the natural language processor. The proprocessor reads the input sentence and, using the lexicon, provides a list of lists of lexical entries, denoted here as "initial strings". The initial strings are simplified by the transformational parser, which amalgamates certain local lexical entries. For example, the phrase "June 10, 1986" is transformed into a noun. The modified strings are parsed by the context-free parser to produce surface structures. The surface structure shown in Figure 3-3 represents the string

"What suppliers are located in Paris?"

Feature information from the lexicon is then attached to the surface structure. Figure 3-4 shows the structure of Figure 3-3 after the addition of feature information. This structure is then parsed by the transformational parser to produce zero or more canonical forms. The parser has rejected the surface structure as ill-formed when zero canonical forms produced. More than one canonical form indicates some sort of ambiguity. The canonical form for the structure illustrated in Figure 3-3 is depicted

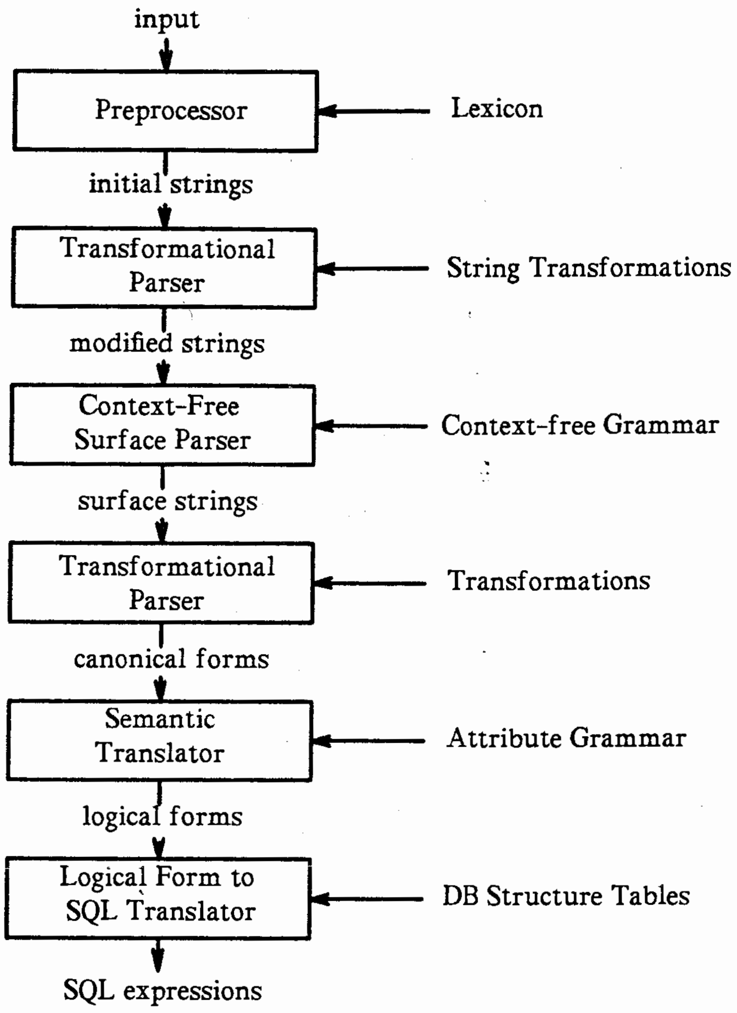


Figure 3-2: Organization of the TQA Natural Language Processor

in Figure 3-5. The canonical forms then undergo semantic translation producing logical forms which are then converted into expressions in SQL, the SQL/DS query language.

Figure 3-6 illustrates the schema of the PARTS database for which TQA produced the examples considered. The feature information attached to the abstract verb in Figure 3-4 indicates that 'locate' takes either a supplier number subject and a locative argument city from the ZS table, or it takes a part number subject and a locative

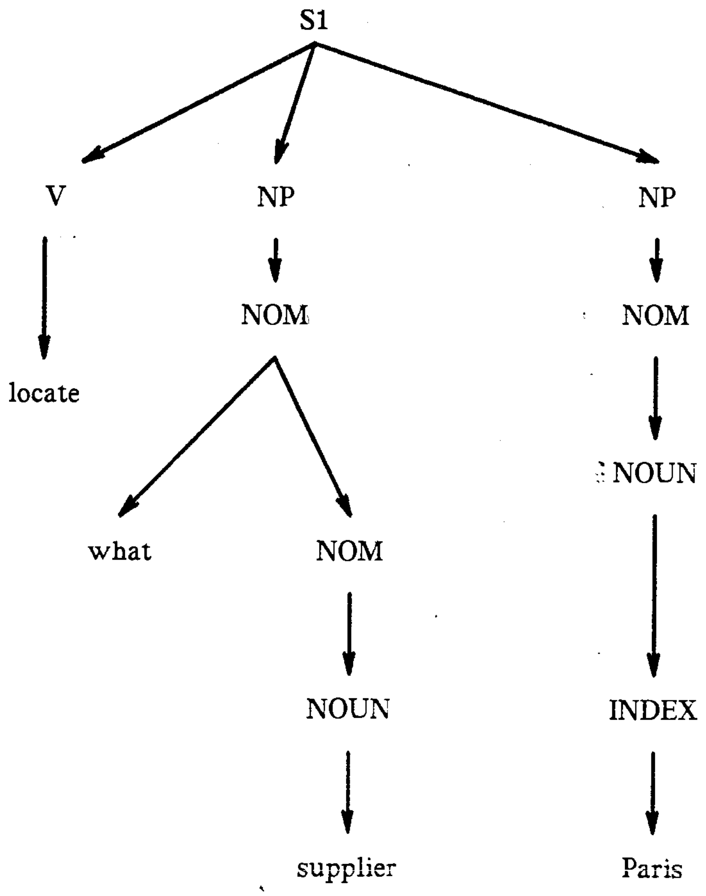


Figure 3-3: Query Surface Structure (from (Johnson, 1984))

argument city from the ZP table. The COLN feature attached to the NOUN in the first NP substructure indicates that "supplier" corresponds to the column name SNO of the ZS table. The +LOC feature of the second NP identifies it as a locative and the feature information attached to the corresponding NOUN node indicates that "paris" is a column value, either from the CITY column of the ZS table or the CITY column of the ZP table. The subsequent parse of this structure uses the COLN information attached to "supplier" to select the ZS alternative for "locate". This in turn restricts the choice for "paris" to the ZS alternative, producing the canonical form of Figure 3-5.

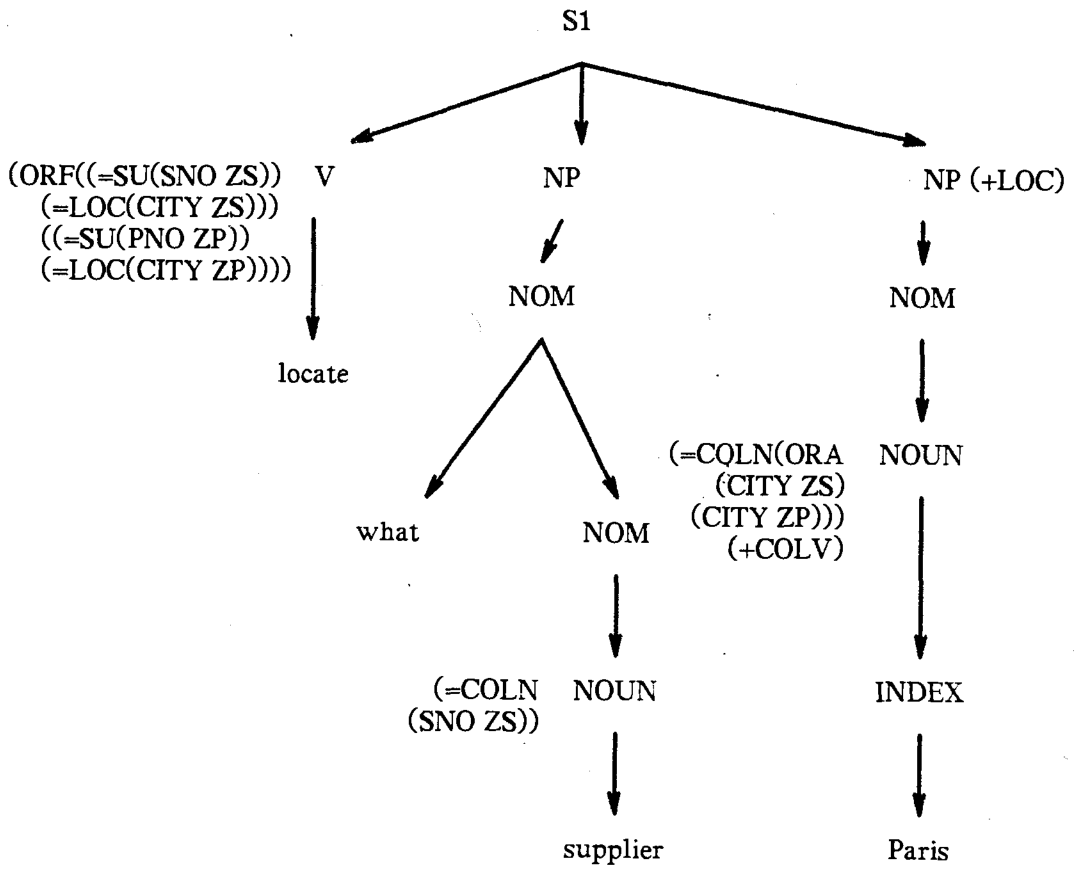

S1
(ORF((=SU(SNO ZS)) (=LOC(CITY ZS))) ((=SU(PNO ZP)) (=LOC(CITY ZP))))
V
locate
NP
NOM
what
NOM
(=COLN (SNO ZS))
NOUN
supplier
NP (+LOC)
NOM
(=COLN(ORA (CITY ZS) (CITY ZP))) (+COLV)
NOUN
INDEX
Paris

Figure 3-4: Structure with Feature Information Added

3.2.2. Structure Verbs and Database Relations

In general, a structure verb such as "locate" from the structure shown in Figure 3-3 corresponds to a vertical or base relation in the database. The feature information attached to a structure verb consists of *selectional restrictions* which associate grammatical roles governed by the verb such as subject, object, locative, and temporal, with relation/attribute names from the database. The elicitation of these selectional restrictions comprises a significant part of the customization process (Johnson, 1984). In fact, this aspect of the customization process is prohibitively difficult for a linguistically naive database administrator (DBA) (Petrick, 1984). Therefore the

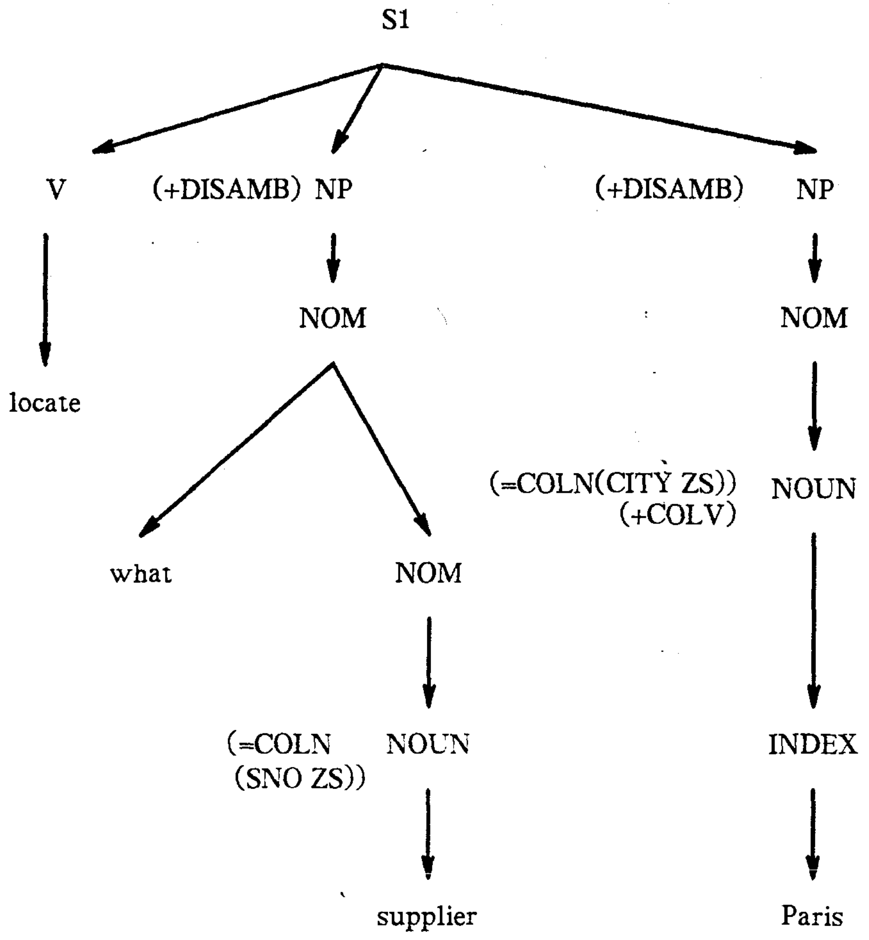


Figure 3-5: Canonical Form (from (Johnson, 1984))

ZS = {SNO, SNAME, STATUS, CITY}
ZP = {PNO, PNAME, COLOR, WEIGHT, CITY}
ZSP = {SNO, PNO, QTY}

Figure 3-6: Example Database Schema

creators of TQA changed the system to eliminate the need for some of the selectional restrictions by making greater use of the COLN features which associate noun phrases with their corresponding relation/attribute pairs. By eliminating some selectional restrictions the MAPP was introduced into the TQA system.

Figure 3-7 is a representation of the query structure for the sentence

"What is the zone of the vacant parcels in subplanning area 410?"

Some detail which is irrelevant to our present discussion has been left out of the representation.

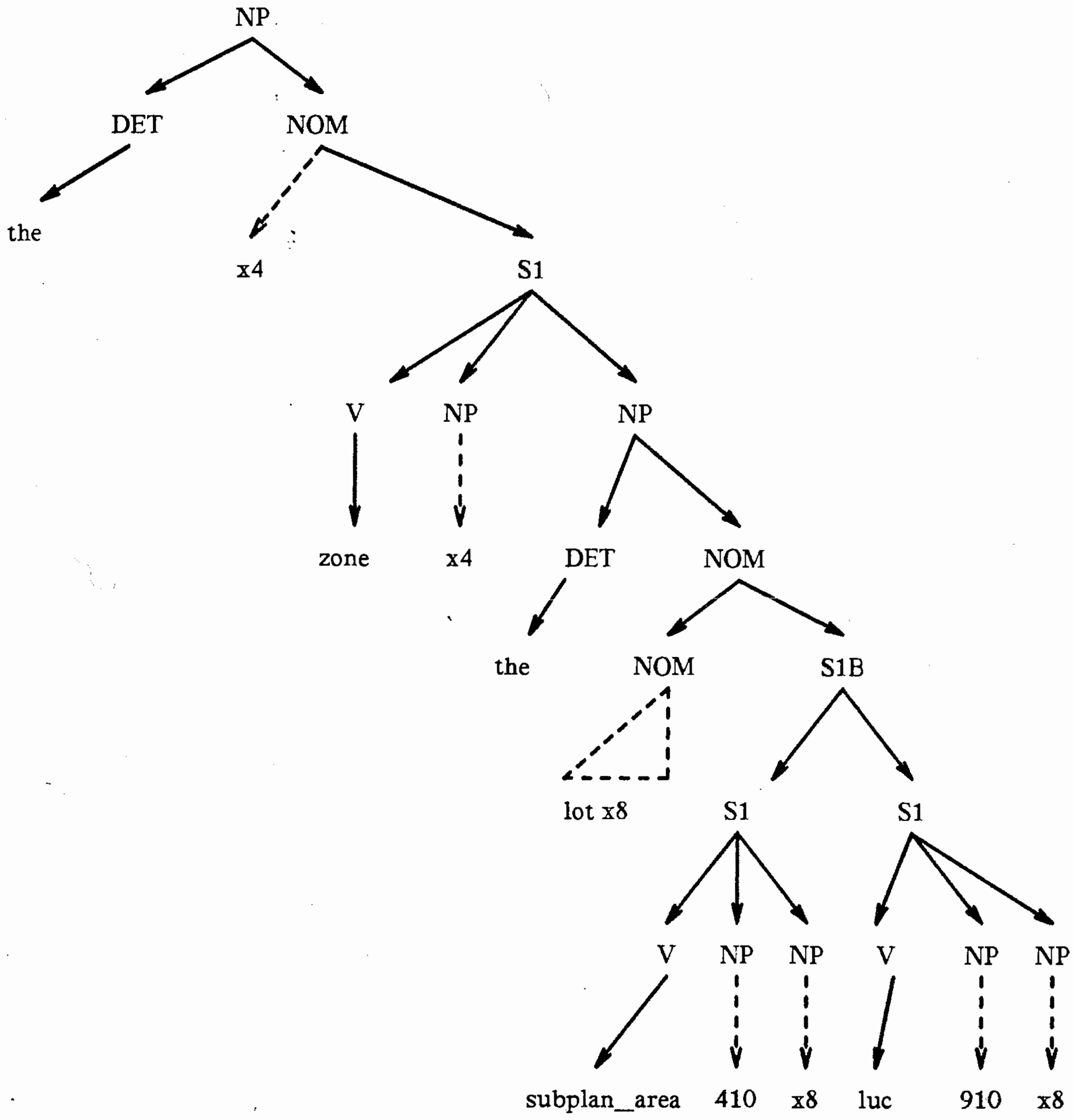


Figure 3-7: Example Query Structure (from (Petrick, 1984))

Figure 3-8 shows the detail for the last S1 substructure in the query structure. Without the selectional restrictions attached to the (abstract) verb "luc" the system

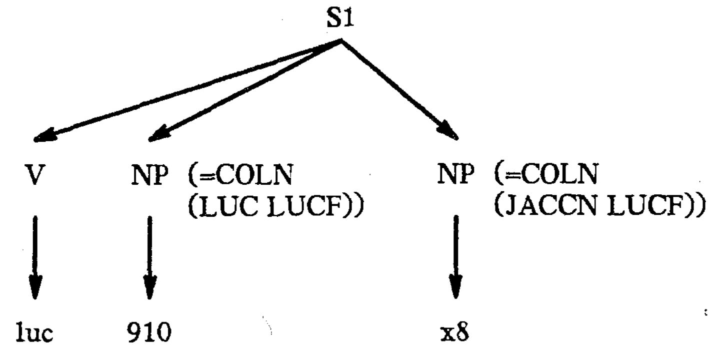


Figure 3-8: Canonical Form corresponding to a Base Relation

must determine which base or virtual relation is indicated. TQA uses the fact that the COLN feature for "910" indicates the LUCF relation and that one of the choices in the COLN feature for "x8" also indicates the LUCF relation to decide that this relation is the one in question. Note that this is an application of the Minimal Connection Strategy (MCS) discussed in Chapter 2. The logical form produced from the query structure depicted in Figure 3-7 is reproduced in Figure 3-9. Each S1 substructure has been translated into a RELATION clause in the logical form. Each RELATION clause represents one base or virtual relation in the database. The attribute list in each RELATION clause lists the attributes from the corresponding relations which are involved in the query. Also part of the RELATION clause is a list of attribute values and a list of the relational operators which relate the values to the appropriate attributes. An attribute value may be a constant or a variable. For example, the final clause in Figure 3-9 represents $\pi_{\{\text{LUC,JACCN}\}}\sigma_{\text{LUC}=910,\text{JACCN}=\text{X8}}\text{LUCF}$ [4]. Shared attribute values indicate how the relations are to be joined to produce the

[4]See Appendix A for an introduction to this notation for representing relational operations. The expression here means "the relation consisting of the LUC-values and JACCN-values projected from the relation consisting of the tuples of the relation LUCF whose LUC-values are "910" and whose JACCN-values are equal to the value of X8."

```
(setx 'X4
  '(setx 'X8
     '(and
        (RELATION 'ZONEF
          '(ZONE JACCN)
          '(X4 X8)
          '(= =) )
        (RELATION 'GEOBASE*
          '(SUBPLA JACCN)
          '('410  X8)
          '(= =) )
        (RELATION 'LUCF)
          '(LUC JACCN)
          '('910 X8)
          '(= =))))))
```

Figure 3-9: Logical Form from Fig 3-7

access path for the entire query. In this case the JACCN-value "X8" is shared among all the relations, indicating that the three relations are to be joined on this attribute. Since the system is able to deduce how to join the relations represented by the S1 substructures of the query structure, if every S1 substructure corresponds to a base relation the MCS is sufficient to solve the MAPP for that particular query. However, when one or more of the corresponding relations is a virtual relation composed of more than one base relation, further heuristics are required. In the logical form shown in Figure 3-9 GEOBASE* is a virtual relation. The SUBPLA attribute is not contained in any of the three relations which contain the JACCN attribute. TQA identifies the relations which make up the virtual relation by some process (which was not made clear in the literature). That is, for each given attribute the system decides which of the relations containing that attribute will be included in the join to create the the virtual relation. Then TQA consults a table compiled during customization from information supplied by the database administrator to retrieve the path required to join these relations. Since TQA presently requires that there be only one joinpath between two relations in the database (Petrick, 1986), compiling this table is a trivial matter. With this restriction in force, it is easy to see that the MCS will provide a unique minimum path between any two attributes. This is most likely the method used to identify the base relations which make up virtual relations.

The restriction to databases with only one access path between two relations results in simple MAPPs solvable by the MCS. However, this restriction is unrealistic (Petrick, 1986). With cyclic databases, the MCS will not be sufficient to resolve the MAPP. Some further strategy is required and we propose one such strategy in Chapter 5.

3.2.3. Structure Verbs and Simple Natural Language Predicates

TQA query structures consist of substructures composed of sentence (S1) nodes which dominate an abstract verb (V) followed by a sequence of noun phrases (NPs). These simple structures are joined together to form a complete sentence structure. Abstract query structure verbs correspond to virtual and base relations in the database and their NP arguments correspond to names and values of the attributes of their associated relations (Petrick, 1984). On the other hand, abstract query structure verbs correspond to natural language predicates (Johnson, 1984). The TQA parser maps natural language predicates into abstract verbs and then maps the verbs into database relations.

The natural language predicates corresponding to these verbs are simple expressions.

They may be natural language verbs. The verb in the query, "Which suppliers are located in Paris?", mapped to the abstract verb "locate". They may be simple nouns. The noun "location" from "Which suppliers have their location in Paris?" would also map to the abstract verb "locate". Sometimes the natural language predicates corresponding to abstract verbs are simple phrases. The phrase "subplanning area" from the land use query on page 29 is an example. Sometimes the predicate is understood. In the phrase "vacant parcel" of the same query, "vacant" corresponds to a value of the LUC (land use code) attribute, while "parcel" corresponds to the attribute name JACCN which refers to lots. The predicate is implicit in the phrase. Complex natural language predicates which are composed of more than one simple natural language predicate are parsed by TQA into the respective components. In Chapter 4 we argue that, since abstract query structure verbs correspond to relationships among entities which can be named by these simple natural language predicates, the database relations which they map to are constrained in ways which help solve the inherent MAPPs.

3.2.4. Performance of TQA

TQA was tested over a period of two years at the City Hall of a New York State municipality. It was used very successfully by municipal employees to access a land use database. Queries from two time periods were collected by the system builders and TQA's performance on them was analyzed. By the end of the test run, TQA was correctly answering 80% of the queries submitted by experienced users, that is, users who had previously submitted more than 25 queries to the system. For detailed results, see (Damerau, 1981). No training was provided to the users.

Subsequent to the test, improvements were made to TQA. In 1983 the collected queries were rerun and this time TQA correctly answered 91.5% of the queries submitted by experienced users. Many of the remaining 8.5% of the queries either were unintelligible to humans or referenced concepts and/or data outside the scope of the database (Petrick, 1984). The database which was used in the test was an acyclic database. We show, in Chapter 5, how the capability of the TQA system can be extended to cyclic databases.

It may be unwise to assume that TQA can achieve the same success that it had with the land-use database with all applications, based on this one test run. The test users may have been especially articulate and accomodating. Perhaps the application itself promoted queries which are easy to interpret. In the remainder of this chapter we argue that the successful performance of TQA during this one field test was not an accident due to the nature of the application or the nature of the users. Thus, given a method to interpret queries to cyclic databases, TQA or an equally powerful system will be able to provide logical data independence for users in practical situations.

3.3. Characterizing Human/Machine Discourse

We have shown that TQA parses natural language sentences into their basic substructures. The MAPP occurs at the level of these substructures. These basic substructures, because they cannot be further broken down, correspond to simple natural language predications. We argue in Chapter 4 that these simple predications refer to definite relationships between entities in the world, which are represented in the database by certain types of relationships between attributes. However, simple natural language predications can be indefinite and ambiguous. For example, does "the classes associated with Prof Jones" refer to the classes taught by Prof Jones, or to the classes offered by the department in which she works, or to the classes taken by the students she supervises? In this section, we argue that certain rules governing human speech acts will also govern human discourse with natural language interfaces. Speech acts have been the subject of extensive research by linguists, philosophers of language and others. We apply the results of Searle (Searle, 1969), one of the most prominent speech act theorists, to explain why accurate evaluation of database queries can be expected. The rules governing the speech acts of reference and predication require that these acts be unambiguous to the hearer. We therefore conclude that the simple predications in natural language queries will be unambiguous and refer to definite relationships between entities in the world. We argue further that these same rules that govern human reference and predication have conditioned people to adjust to the frames of reference of their discourse partners and that they adjust in the same way when their partners are computer systems. Therefore the predications appearing in natural language queries tend to be ones which the systems can interpret.

3.3.1. Propositional Acts and the Principle of Identification

Among the different types of speech acts identified by Searle are propositional acts. The propositional acts are acts of referring and predicating. These acts are committed by speakers in the performance of various illocutionary acts. Illocutionary acts are another type of speech act identified by Searle. Making a request and asking a question are two illocutionary acts. The propositional acts committed during these two illocutionary acts are the specification of a future action of the hearer and/or the identification of the information desired by the speaker. Thus propositional acts are made when querying a database in a natural language.

According to Searle, human speech is a rule-governed activity. The commission of propositional acts is governed by rules, as is the commission of all speech acts. The principle behind the rules governing propositional acts is the *principle of identification.* This principle holds that successful reference must contain enough information so that the hearer can uniquely identify the referent in the context of the utterance, that is, the reference must contain an *identifying description.* Since speakers are obligated by the rules governing reference to include a description of the object referred to that will allow the hearer to identify that object in the context of the discourse, it follows that speaker will adjust to the frame of reference of the hearer. See (Perrault and Cohen, 1981) for a discussion of the mutuality of accurate reference.

A speech act context provides information that, together with the utterance, makes the propositional content of that act unambiguous. In the following example dialogue there are five (potential) actors: John has a sister and a classmate both of whom are named Mary; Sue knows John's family but not his class; Fred is in John's class but does not know his family; Joe is also in John's class but does know John's family; Ann knows John but does not know anyone else in his class or his family.

X: Hi, John!
John: Hi, X!
X: What did you do last night?
John: I went to the movies.
X: Who did you go with?
John: Mary. (1)

Depending on whether X is Sue, Fred, Joe, or Ann, John's reference to his companion of the previous evening is an identifying or non-identifying description of that person. Sue recognizes John's sister, Fred recognizes John's classmate, Joe is confused between the sister and the classmate, and Ann has no idea to whom John is referring. Furthermore, in the same assertion (1), there is a successful reference to a particular time (the previous night), another successful reference to an individual (the speaker John), and a predication of the objects referred to (go_to_the_movies), all of which are provided by the discourse that preceded the assertion. If John is speaking with Sue or Fred, all of the references in assertion (1) are successful. Therefore, if the hearer is one of those two people, the assertion is successful. However, if the hearer was either Joe or Ann, John broke the rule governing reference and the assertion does not succeed.

3.3.2. Context of Human/Machine Discourse

The context of a speech act is partly defined by the common knowledge of the speaker and the hearer. The knowledge of the rules governing speech acts is an important part of this common knowledge. Knowledge may be common for different reasons. Among these reasons are the fact that both speaker and hearer are human, both belong to the same university, or both have participated in the same dialogue. Much of the discourse between people who are unfamiliar with each other is directed at establishing the boundaries of common knowledge.

A; "What did you think of the Leaf's game last night?"
B: "I don't follow baseball."

As a person 'gets to know' his partner better, as they establish and extend the boundaries of their common knowledge, they can communicate more deeply and broadly because they are able to successfully co-refer to more entities.

When a person discovers that her discourse partner does not share knowledge that she previously assumed he did share, she will adjust to this new frame of reference. In fact, the rules governing the propositional speech acts require that she do. A reference to an object must identify that object such that the hearer can recognize it. If the above dialogue about John's activities had taken place with Fred, it may have continued in the following manner:

> Fred: "I didn't know you two were going together!"
> John: "I meant my <u>sister</u> Mary!"

Realizing his mistake and remembering that Fred does not know his family, if John wants to refer to his own brother later in the conversation he will not use his brother's proper name alone but will include enough information to identify him clearly. People expect to adjust to others who have different frames of reference and accept their obligation to do so. This occurs naturally when adults talk with children and when people from different cultures converse.

When users are interacting with a machine in a natural language they will bring to that interaction the rules they have learned which govern speech in that language. This an assumption which underlies all work in Computational Linguistics. The motivation is to free the users from having to learn new rules. The rules governing propositional speech acts require that speakers clearly identify their referents. This means that speakers must adjust their speech to suit the hearers. Therefore, a reasonable expectation is that users will endeavour to clearly identify referents to the computer systems with which they are interacting and will adjust their queries in order to do so.

References in natural language queries to databases will, therefore, tend to be unambiguous to the system. This tendency will increase as the user identifies the areas of common knowledge and adjusts to the system.

Chapter 4
The Semantics of Attribute Relationships

A database is a model of some part of the world. The tuples in the base and virtual relations of the database represent facts in the modelled part of the world (Biller, 1979). Facts can be expressed by natural language expressions. Therefore a correspondence exists between database tuples and natural language expressions. We investigate this correspondence in order to develop a systematic way to translate certain natural language expressions into database language expressions which specify tuples.

If a database is to be an accurate model of the real world it must be constrained in such a way that tuples which do not represent facts are not derivable via legitimate operations on the database, including insertion and update. A database designer examines the part of the world he wishes to model and selects those facts which constrain the database to be an accurate model. These constraints are expressed as data dependencies. Functional dependencies (FDs) and multivalued dependencies (MVDs)[5] are the most important of the data dependencies relevant to the problem of designing a database which is an accurate model (Ullman, 1982a). Once the dependencies have been determined, the database is designed in such a manner that the dependencies are incorporated into the database structure.

[5]See Appendix A for a discussion of dependencies

One of the major purposes of database systems is to avoid redundancy of data. Normalization is a design process by which redundancy in a database can be reduced while maintaining the accuracy of the database. There are a number of levels to which relations in a database may be normalized. Each level yields relations with certain characteristics. To begin with we will assume our database is normalized into fourth normal form.[6] That is, each relation scheme R in the database contains only functional dependencies of the form $K \rightarrow X$, where K is a candidate key for R and $X \in R$. No other functional dependencies or multivalued dependencies exist in R. We will also assume, for simplicity, that each R has only one candidate key - its primary key.

Our purpose is to provide a mapping between expressions in natural language, which represent facts in the world, and tuples in the database, which also represent facts in the world. In Chapter 3 we saw how complex natural language expressions could be broken down into their composite structures and the words and phrases that make up those substructures finally transformed to abstract verbs and noun phrases. The abstract verbs are translations of simple natural language predicates. In general, the abstract verbs correspond to database relation names and the noun phrases correspond to attribute names and values. We saw that the translation of the abstract verbs into database base and virtual relation names presented a problem when the database in question was cyclic. We attempt to find a method to make a reasonable guess of the correct relation names corresponding to the abstract verbs in the query structure.

[6] See Appendix A for a discussion of normal forms

We proceed by examining the semantics of the relationships between attributes in the database and use what we learn to assist us in defining the mapping we seek.

4.1. Basic Relational Database Semantics

In this section we discuss the semantic connection between natural language expressions and simple database expressions. We will use the UNIVERSITY database introduced in Chapter 2 to illustrate our discussion. The database scheme is repeated below for convenience.

STUDENT = {STUD#, SNAME, MAJOR, MINOR, SSEX}
FACULTY = {FAC#, FNAME, FSEX}
COURSE = {CNAME, DEPT, UNITS}
CLASS = {CLID, CNAME, TERM, FAC#}
GRADES = {CLID, STUD#, GRADE}
APPOINT = {FAC#, DEPT}

Entities exist in the world. Entities may be substantial beings, such as faculty members. Entities may be insubstantial beings, such as departments. Entities may be relationships. For example, an appointment is a relationship between a faculty member and a department. Some entities are highly complex. Classes are complex entities composed of students and faculty members which stand in certain relationships to courses and departments. Entities have properties, or, from another perspective, entities have relationships with other entities which are properties. There is a property of being female; some faculty members have this property. Some courses have the property of being worth three credits.

Particular entities often have names in natural languages. One faculty member is named Ann Jones. One course is named Math 100. One property of Ann Jones is named "female". One property of Math 100 is called "three credits". Particular entities are referred to in databases by attribute values. Ann Jones is referred to with the FAC#-value "85110-1000". The faculty number is used rather than the name because it can be guaranteed to be unique to Ms. Jones, whereas "ann jones" cannot. The CNAME-value "math 100" refers to the course Math 100. The UNITS-value "3" refers to the property, 'three credits'. Natural language names for particular entities map to attribute values in the database.

Referencing the corresponding entity is one of the *semantic roles* played by an attribute value. Other semantic roles of attribute values are discussed below.

In the world, relationships exist between entities. Relationships are represented in relational databases by the structuring of attributes into relation schemes. A relation scheme represents a macro view of an entity type which includes certain of the significant relationships in which the entity type participates. For example, FACULTY = {$\underline{\text{FAC\#}}$,FNAME,FSEX} is a macro view of the 'faculty member' entity type, representing the entity type and some of its properties. CLASS = {$\underline{\text{CLID}}$,CNAME,TERM,FAC#} represents the 'class' entity type and certain relationships it has with the 'term', 'course' and 'faculty member' entity types. The tuples of relations represent instances of the macro views represented by the scheme of the relation to which they belong. When discussing the relationships between the entities represented in the database we will refer to instances of macro entities repesented by tuples as *tuple entities* and entities represented by attribute values as *attribute entities*. A tuple *t* in a *class* relation represents a particular tuple entity, a class. The FAC#-value in *t* represents a particular attribute entity, a faculty member. Thus tuples summarize entities. The key value of the tuple names the entity. The functional dependencies between the key value and the non-key attribute values represent relationships between the tuple entity and certain attribute entities.

The relationships between entities in the world which correspond to functional dependencies in the database are most likely to be referred to by simple natural language expressions. An attribute B is functionally dependent on an attribute A if and only if each A-value has associated with it exactly one B-value. Presuming this functional dependence represents an actual state of affairs, the corresponding relationship in the world can be characterized as direct, definite, and thus likely to be prominent in the awareness of the group of people involved with that state of affairs. Such relationships are therefore likely to be referred to simply by the group, especially if the relationships are significant to them. The relationships identified by a database designer as functional dependencies to be built into the database are almost certainly significant to the community of users of the database. Therefore, this community will usually have simple natural language expressions of the kind we saw in Chapter 3 to refer to these relationships.

However, such relationships are not named explicitly in relational databases. Rather, they exist implicitly in the structure and are specified by expressions which name the operations required to extract the implicit relationships. For example, consider the "majoring in" relationship which exists between students and departments. In general, this relationship is specified in the database by $\pi_{\{\text{STUD\#},\text{MAJOR}\}}(\text{STUDENT})$. For a particular world, the students of which are repesented by a particular ***student*** relation, all such relationships are specified by $\pi_{\{\mathrm{STUD\#,MAJOR}\}}(\mathit{student})$. A particular relationship within that world, for example, the one for the student represented by "86300-3394", is specified by $\pi_{\{\mathrm{STUD\#,MAJOR}\}}(\sigma_{\mathrm{STUD\#}=86300\text{-}3394}(\mathit{student}))$. Such expressions specify virtual relations implicit in the database. Simple natural language predicates, together with their arguments, map to the virtual relations in the database.

The mapping from natural language expressions to virtual relations in the database is problematic because it involves specifying the operations required to derive the virtual relations. The difficulty is more pronounced when multiple relations having multiple access paths between them are involved in the virtual relation. In order to devise a strategy to make reasonable guesses of the correct choice of access path, we examine the semantics of the relationships between attributes. A virtual relation is a relation between attributes which represents a relationship between entities in the world. The semantics of such relations between attributes constrain the choice of access paths.

4.2. Attributes within a Base Relation

Let us examine the relationship between the attributes belonging to a single relation scheme. Let an attribute A, which alone forms a key for relation scheme R, be called the *key attribute* for R. Let all other attributes in R be called the *non-key attributes* for R. For those relations in which the key is composed of more than one attribute, let each of the attributes that make up the key be called *co-key attributes*. A co-key attribute is, by definition, also a non-key attribute. For any key K of a relation scheme R and a tuple t of r, let the K-value of t be called the *key value*.

Consider the CLASS relation scheme,

CLASS = {<u>CLID</u>, CNAME, TERM, FAC#}.

Each tuple in a relation on the CLASS scheme represent a class in the university. Since CLID is the key in the scheme, each class represented by a tuple is uniquely identified and represented by a CLID value, that is, by a class identifier.

There are two semantic roles played by attribute values within a tuple, with respect to the tuple to which they belong. These are a *predicative role* and a *classificatory role.* These roles are separate from the role of referencing their respective attribute entities. The interplay of the attributes' semantic roles defines the semantics of the relationships between attributes. We discussed the roles attribute values play referencing attribute entities in Section 4.1. We now examine the predicative and classificatory roles.

4.2.1. Predicative Role

An attribute predicates a property of the entity type represented by the relation scheme to which the attribute belongs. For example, the attribute FAC# in the relation scheme CLASS predicates a property of classes, namely the property "instructor_of". In the relation scheme APPOINT, FAC# predicates a property of appointments, that is, the property "appointee_of". Note that we are using the term 'property' in a general sense when we say that an appointee is a property of an appointment, or a professor is a property of a class. In cases such as these, the predicative role of the attribute involved is to relate the tuple entity to the attribute entity rather than to relate the tuple entity to one of its properties. Consider a given appointment x which involves the appointment to the Math department of Ann Jones, whose faculty number is 85110-1000. Appointment x will be represented in the database by the following *appoint* tuple.

<85110-1000, math>.

The predication represented by the occurrence of the FAC#-value '85110-1000' in the above tuple may be represented in logical notation as

appointee-of(x, 85110-1000).

The relationships which exist between a tuple entity and its properties are the kinds of relationships for which natural language has simple expressions of the type in which we are interested. We have for the FAC# attribute of the CLASS relation scheme:

"the teacher of the class",
"the instructor of the class",
"the faculty member teaching the class",
"the professor taking the class",
"the lecturer in the class,

and so on. In addition, the relationship may be expressed in English in a passive voice. For example,

"the classes taught by the faculty member", and
"the classes given by the professor".

Since a key uniquely identifies a tuple, a key also uniquely identifies the entity which the tuple represents. Therefore the type of relationship which holds between a non-key attribute and the tuple to which it belongs, holds between that attribute and the key of that tuple. Thus the predicative relationship described above that holds between FAC# and the tuples of CLASS, holds between FAC# and CLID, the key for CLASS. Suppose the *class* relation from a given instance of the UNIVERSITY database contains the following tuple,

<m100/861/s1, math100, 86-1, 85110-1000>,

where 'm100/861/s1' is the class identifier of Section 1 of the MATH 100 offering in the first term of 1986, and 85110-1000 is the faculty number for Ann Jones. For simplicity, in the following discussion we refer to this instructor by her name instead of her number. The predications contained in this tuple could be expressed in English as follows,

"Professor Jones teaches m100/861/s1.",
"The instructor of m100/861/s1 is Jones.",
"m100/861/s1 is given by Jones.",

and so on. Similarily, for the remaining attributes of the example tuple, we have,

"m100/861/s1 is an offering of MATH 100.",
"m100/861/s1 is a MATH 100 class.", and
"m100/861/s1 is being offered in 86-1."

Note that this predicative relationship that holds between a tuple of a base relation and its attributes is inherent in the relationship that a database designer identifies as a functional dependency and builds into the database structure. The tuple entity is denoted by the key for the tuple. The non-key attribute values of the tuple are all functionally dependent on the key value. Implicit in the functional dependency relationship between key and non-key attributes of the same base relation is the predicative role described in this section. We can illustrate the functional dependency between the attributes of $R = \{\underline{A}, B\}$ with the dependency diagram in Figure 4-1. Given this functional dependency we would expect there to be in the world a

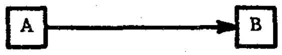


Figure 4-1: Dependency Diagram for $R = \{\underline{A}, B\}$

definite, direct relationship between the types of entities represented by A and B. Furthermore, for the reasons given in Section 4.1, it is likely that there are simple natural language expressions which describe this relationship.

4.2.2. Classificatory Role

There is a kind of semantic equivalence that exists between sets of values of an attribute of a given relation and sets of key values of that relation. Consider value a of the non-key attribute A in relation r on relation scheme R. Value a specifies a certain set of tuples **S** from r. For example, the value "math 100" in the CNAME attribute of the CLASS relation specifies the set of tuples that represent the offerings of the Math 100 course. The set **K** of key values for **S** also specifies **S**. Thus a given value a from dom(A) is *co-referent* with a unique set of values **K** in dom(K). That is, they both uniquely denote a certain set of tuples of the relation to which they both belong. No other value from dom(a) nor any key value in r which is not a member of **K** denotes any of the tuples denoted by a and the set **K**. It is this equivalence relation between non-key attribute values and sets of key values in relations that yields the semantics for the relationships between non-key attributes from the same relation and between attributes from different relations.

This semantic equivalence between the key and non-key attributes from the same relation arises from the functional dependence that exists between them. $K \rightarrow A$ implies a 1:n mapping between dom(A) and dom(K). Furthermore, the functional dependence implies that no two distinct values, a_1 and a_2, from dom(A) map to the same value k from dom(K). Thus every A-value occuring in any relation r specifies a unique set of keys which is disjoint from the set of keys specified by every other A-value occuring in r. Therefore any distinct set of A-values in r must specify a unique set of keys for r.

Natural language also supports this classification of entities by their properties. One may refer to a certain subset of the students in a university as being the "females", or to another as being the "Math majors".

Consider again the relationship illustrated in the dependency diagram of Figure 4-1. We expect to find simple natural language expressions denoting this relationship and we also expect to find natural language names corresponding to B-values used to denote sets of entities corresponding to sets of A-values.

4.2.3. The Co-dependency Relationship

The kind of relationship that can be referred to with a simple natural language expression often exists between the entities represented by non-key attributes in a relation. For example, associated with the CLASS relation scheme, we have

"the teachers of the courses",
"the courses taught by the teachers",
"the terms a course is offered",
"the courses offered this term",
"the faculty teaching this term",
"the terms a professor was lecturing",

and so on. The above expressions correspond to virtual relations formed by projecting the *class* relation onto pairs of attributes. For example, the first two expressions correspond to $\pi_{\{\text{CNAME,FAC\#}\}}(class)$.

It is the ability of attribute values to classify, or 'stand for', sets of tuples that establishes the connection between non-key attributes in a relation which is expressible in a simple natural language expression. In a given instance of a *class* relation, a particular FAC# value f stands for a set of tuples **S**, which can be represented by the set of keys **U**. The set **S** contains the set of CNAME values **T**, which are the properties 'offering_of' which are being predicated of **S**. We therefore have

offering_of(**S**,**T**).

Substituting f for **S** we have

offering_of(f,**T**).

This predicative relationship can be expressed in English as

"Professor (f)'s courses",
"the courses (**T**) offered by Professor (f)",
"the courses (**T**) taught by Professor (f)",

and so on.

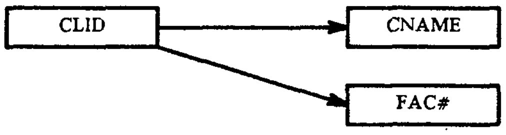


Figure 4-2: Co-dependency of CNAME and FAC#

Figure 4-2 illustrates these relationships. CNAME and FAC# are *co-determined* by CLID. The functional dependency CLID → CNAME permits a CNAME-value to classify a set of CLID-values of which are predicated a set of FAC#-values. The functional dependency CLID → FAC# permits the FAC-values to be predicated of the CLID-values. This *co-dependency* relationship between CNAME and FAC# reflects a relationship between courses and faculty members in the world. This relationship between courses and faculty members is characterized by the type of relationship faculty members have with classes and the type of relationship courses have with classes. We expect natural language names for courses to denote classes and natural language names for faculty members to denote properties of those classes. Similarly, names of faculty members can denote classes and names of courses can denote properties of those classes. We find, therefore, such simple natural language expressions as we have seen above describing the relationship between courses and faculty members. In general, we expect to find simple natural language expressions corresponding to co-dependency relationships between non-key attributes within a base relation.

4.3. Attributes from Different Base Relations

The relationship between attributes from different relations is also sometimes expressible in terms of simple natural language expressions. Consider the two relation schemes CLASS and GRADES from the UNIVERSITY database.

CLASS = {$\underline{\text{CLID}}$, CNAME, TERM, FAC#}
GRADES = {$\underline{\text{CLID}}$, $\underline{\text{STUD\#}}$, GRADE}

The tuples of the relations on GRADES represent an aspect of the experience of one student taking one class in the university. Let us refer to this 'entity' as an *undertaking*.

Some of the relationships which exist between attributes from the GRADES and CLASS schemes can be expressed in English as follows:

"the grade received for the course",
"the grades received for the term",
"the grades given by the professor",
"the student taking the course",
"the students registered for the term", and
"the students taught by the faculty member".

These expressions correspond to virtual relations formed by projecting pairs of attributes on the join of *class* and *grades* relations.

Whether or not such virtual relations have corresponding simple NL expressions is largely dependent on the semantics of the connection between the base relations which are joined to form the virtual relation. The semantic connection created between the two base relations when the join takes place on a set of attributes which contains a key for one of the relations often yields virtual relations which have corresponding simple natural language expressions. When the shared attribute set does not contain a key from one of the relations, the relationship represented is much weaker and it is much less likely that there will be corresponding simple natural language expressions.

4.3.1. Joining on a Key

To investigate the semantic connection between attributes from different relations where the connection between the relations contains a key for one of the relations, let us consider the virtual relation $\pi_{\{\mathrm{GRADE,CNAME}\}}(grades \bowtie class)$,[7] using for examples the relations depicted in Figure 4-3. Figure 4-3 contains a *class* relation which represents three classes, two of which are offerings of Math 100 and one offering of Math 250.

[7]"$\bowtie$" is the join operator symbol. *grades*$\bowtie$*class* refers to the relation created when the *grades* relation is joined to the *class* relation on their common attributes.

CLASS

m100/861/s1	math 100	86-1	85110-1000
m100/861/s2	math 100	86-1	86310-0002
m250/861/s1	math 250	86-1	81110-0218

GRADES

m100/861/s1	86100-1032	A
m100/861/s1	86100-1922	C
m100/861/s2	86100-1168	B
m100/861/s2	86100-0273	B
m250/861/s1	85200-0172	B
m250/861/s1	84300-1528	C

GRADES$\bowtie$CLASS

m100/861/s1	86100-1032	A	math 100	86-1	85110-1000
m100/861/s1	86100-1922	C	math 100	86-1	85110-1000
m100/861/s2	86100-1168	B	math 100	86-1	86310-0002
m100/861/s2	86100-0273	B	math 100	86-1	86310-0002
m250/861/s1	85200-0172	B	math 250	86-1	81110-0218
m250/861/s1	84300-1528	C	math 250	86-1	81110-0218

Figure 4-3: Semantic Connections Between Relations

The figure also contains a *grades* relation which represents the two undertakings of each of the three classes. The third relation is *grades*$\bowtie$*class*.

When two relations, R_1 and R_2, are joined on a set of attributes which contains the key K_1 for R_1, the key K_2 for $R_1 \bowtie R_2$ is the same as the key for R_2. Assuming that K_1 represents the same entity type in both R_1 and R_2, all the non-key attributes from R_1 are transitively functionally dependent on K. The entity type represented by $R_1 \bowtie R_2$ is the entity type that is represented by R_2. The join augments R_2 by including the additional relationships between the tuple entity and the entities represented by the attributes of R_1. Thus $R_1 \bowtie R_2$ represents a wider macro view of the entity type represented by R_2. The attribute values of R_2 perform the same roles in $R_1 \bowtie R_2$ as they did in R_2. That is, they refer to the same attribute entities and to the same relationships those entities have with the same tuple entity. The attribute values of R_1 which are included in $R_1 \bowtie R_2$ have expanded roles in the virtual relation. They refer to the same attribute entities. However, they now represent the functional dependency relationship between those attribute entities and the tuple entities of $R_1 \bowtie R_2$. Let us refer to a join of two relations R_1 and R_2, which takes place on a set of attributes that contains the key K_2 for R_2 as the *key join* of R_1 to R_2, written $R_1 \bowtie_k R_2$. All other joins are called *non-key joins*.

The relation *class*$\bowtie$*grades* of Figure 4-3 contains tuples which are an expanded view of the undertakings represented by the *grades* relation. This view incorporates details about the classes which make up the undertakings. {CLID,STUD#} is the key in both GRADES and GRADES$\bowtie$CLASS. The entities represented by the virtual relation, i.e. undertakings, are instances of an entity type which has been selected by the designer of the database as worthy of representation in a base relation. As we have discussed earlier, these entities and the relationships they have with the entities represented by the attributes of the base relation are likely to be referred to with simple natural language expressions. For example, tuple entities of an *appoint* relation represent "appointments". We speak of the "appointment of Prof. Jones" and of an "appointment to the Math Department". We shall see in the following subsection that the entities represented by virtual relations created by a join of two base relations on a set of attributes that do not contain a key for one of the base relations are much less likely to be referred to with a simple natural language predicate.

In the *class* relation of Figure 4-3, the CNAME-value "math 100" denotes the Math 100 course and specifies as well the two Math 100 classes represented by the first two tuples of the relation. In *class*$\bowtie$*grades*, "math 100" specifies the undertakings represented by the first four tuples of the virtual relation as well as the one course and the two classes. Inasmuch as as the functional dependence of CNAME on {CLID,STUD#} is transitive, this specification is also transitive. CNAME-values specify CLID-values which specify {CLID,STUD#}-values which denote undertakings. The attribute on which the transitivity depends is CLID, the attribute on which the two attributes are joined. The semantic roles CLID-values play in *class⋈grades* are relatively straightforward and uncomplicated. They denote classes and specify undertakings. These are the same roles they play in the *grades* relation. We shall see that when two relations are joined by a non-key join, the semantic roles played by the values of the attributes on which the join takes place are more complex.

In the relation *class⋈grades* "math 100" denotes a specific set of undertakings. These undertakings have the grades denoted by the GRADE-values in their respective tuples as properties. The relationship between the grades and the undertakings is the kind of predicative relationship which is usually named by a simple natural language expression. Thus we expect to find such expressions linking natural language names for these entities.

The attributes CNAME and GRADE belong to different relation schemes. However, as shown in Figure 4-4, they are co-dependents. When the transitive functional dependency is 'collapsed'[8] the dependency diagram in Figure 4-4(a) yields the diagram in Figure 4-4(b), which has the same structure as Figure 4-2.

[8]To 'collapse' a chain of transitive functional dependencies is to remove the intermediate attributes in the chain and thus make the functional dependence between the first and last attribute explicit.

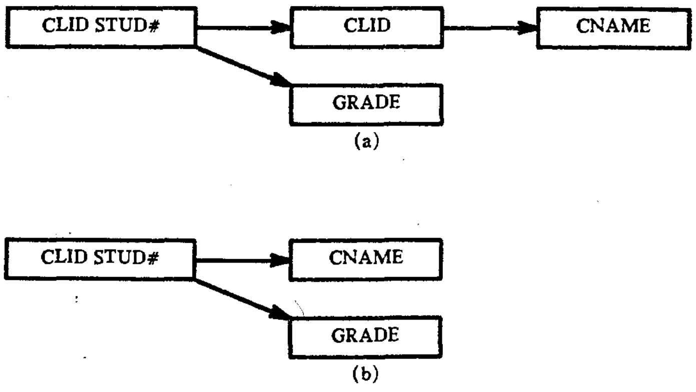


Figure 4-4: Co-dependency of CNAME and GRADE

Attributes need not belong to the same or adjacent relations to be co-dependents. We saw earlier that CNAME from CLASS and GRADE from GRADES were co-dependent. As Figure 4-5 illustrates, DEPT is also co-dependent with GRADE.

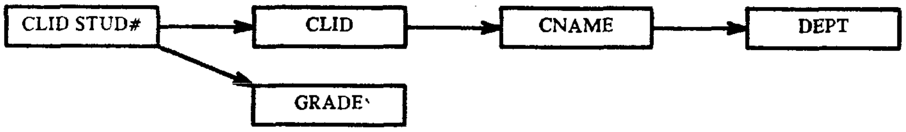


Figure 4-5: Co-dependency of DEPT and GRADE

Therefore the virtual relation resulting from

$$\pi_{\{\text{DEPT},\text{GRADE}\}}(grades \bowtie class \bowtie course)$$

is a candidate for having a corresponding simple natural language expression. We say, for example, in English

"the grades for the department".

A co-dependency may contain more than two co-dependents and still have a corresponding simple natural language expression. Consider the dependency diagram in Figure 4-6. The simple natural language predicate, "the names of the female Math

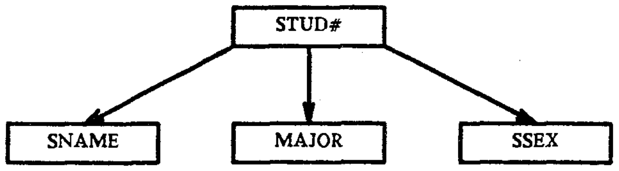


Figure 4-6: Co-dependency with three Dependents

majors", expresses an instance of the relationship depicted in Figure 4-6. In this case a SSEX-value and a MAJOR-value simultaneously classify a set of STUD#-values of which are predicated a set of SNAME-values.

Complex co-dependencies can correspond to simple natural language predicates. Consider the dependency structure in Figure 4-7.

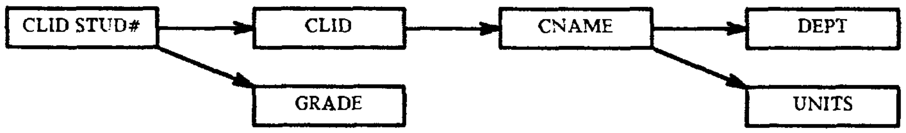


Figure 4-7: Complex Co-dependency

This structure is an extension of the dependency structure depicted in Figure 4-5; the node for UNITS has been added. The co-dependency depicted in Figure 4-5 resulted when a department classified a set of classes which classified a set of undertakings of which were predicated a set of grades. In the current example, the undertakings of which the grades are predicated, are simultaneously transitively classified by a department **and** a credit value. This co-dependency can be expressed by a simple predicate, for example, "the grades for the three-credit Math courses".

Not all co-dependency relationships have corresponding simple natural language expressions. UNITS and FAC# are co-dependent as shown in Figure 4-8. Nevertheless

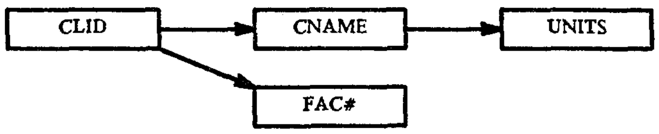


Figure 4-8: Co-dependency of UNITS and FAC#

there does not seem to be a way in English to refer to UNITS-values and to FAC#-values together without explicitly mentioning the courses which link them. That is, we say

"the teachers of three-credit courses"

rather than

"three-credit teachers"

or something similar. The semantic relationship between these two attributes is undistinguishable from that between DEPT and FAC# from the same relations, yet

"Who's teaching for the Math department?"

is meaningful. Presumably the relationship between UNITS and FAC# is not significant enough to warrant the simple expression. Both UNITS-values and DEPT-values denote specific sets of classes but we are more apt to classify classes by department than we are by credit.

4.3.2. Joining on Non-key Attributes

Consider the COURSE and APPOINT relation schemes and the virtual relations which may be formed by projecting pairs of attributes from their join where the members of the pair are from different relations.

$\pi_{\{\text{CNAME,FAC\#}\}}(course \bowtie appoint)$ corresponds to

"the courses offered by the departments employing the faculty members", or
"the faculty in the department offering the courses".

$\pi_{\{\text{UNITS,FAC\#}\}}(course \bowtie appoint)$ corresponds to

"the x-credit courses offered by the departments employing the faculty members", or
"the faculty in the department offering the x-credit courses",
where x is a variable for UNITS-values.

$\pi_{\{\text{CNAME,DEPT}\}}(course \bowtie appoint)$ corresponds to

"the courses offered by the departments which employ faculty members", or
"the faculty-employing departments offering the courses".

None of these or any of the other possible virtual relations can be expressed with a simple natural language expression.

We expect the same result for all such pairs of relations. Consider two relations with the following schemes, $R_1 = \{\underline{K}_1,A,B\}$ and $R_2 = \{\underline{K}_2,A,C\}$. Let $R_1 \bowtie R_2 = \{\underline{K}_1\underline{K}_2,A,B,C\}$ be called R_3. Let the attribute entity denoted by a given A-value a_i be ae_i. Let the non-empty set of R_1 tuple entities denoted by a_i be **S** and the non-empty set of R_2 tuple entities denoted by a_i be **R**.

The key for $R_1 \bowtie R_2$ is K_1K_2, which represents a relationship of some type between the entities represented by tuples of R_1 and the entities represented by tuples of R_2. Note that such a relationship is very possibly arbitrary; that is, its implicit incorporation into the structure of the database was not necessarily deliberate and thus is not necessarily a relationship of significance to the users of the database.

The attribute value a_i in R_1 denotes ae_i and a certain relationship r_1 that entity has with the members of the set **S** of R_1 tuple entities. In R_2, a_i again denotes ae_i; it also denotes a relationship r_2 that ae_i has with the members of the set **R** of R_2 tuple entities. In one tuple $t = <k_1, k_2, a, b, c>$ of R_3, a denotes the the entity ae which has relationship r_1 with the entity denoted by k_1 **and** the relationship r_2 with the entity denoted by k_2. The tuple entity t represents is the relationship of the entities denoted by k_1 and k_2. The relationship represented by t is the one that results when k_1 has relationship r_1 with the same a with which k_2 has relationship r_2.

Let us consider an example using the COURSE and APPOINT relation schemes:

COURSE = {CNAME, DEPT, UNITS},
APPOINT = {FAC#, DEPT}.

Suppose our database contained the *course* and *appoint* relations shown in Figure 4-9.

COURSE

math 100	math	3
cmpt 101	cmpt	4

APPOINT

85110-1000	math
86310-0210	math
82210-0014	cmpt

COURSE⋈APPOINT

85110-1000	math	math 100	3
86310-0210	math	math 100	3
82210-0014	cmpt	cmpt 101	4

Figure 4-9: Relations Joined on Non-key Attribute

In *course* "math" denotes the Math department and indicates that this department is in the 'offers' relationship to the course Math 100. In *appoint* "math" denotes the Math department and indicates that it is in the 'employer' relationship with the appointments denoted by "{85110-1000, math}" and "{86310-0121, math}". In *appoint⋈course* the key is {CNAME,FAC#,DEPT}. Tuples in this relation represent a relationship between appointments and courses. The relationship represented is that the appointments are in the same departments that offer the courses. The DEPT-value "math" in *course⋈appoint* denotes the Math department and indicates that this department both offers Math 100 and is the employer involved in the appointments denoted by "{85110-1000, math}" and "{86310-0121, math}".

4.3.3. The Co-incidental Relationship

Figure 4-10 illustrates the dependency relationships of K_1, K_2, and A from the relations $R_1 = \{\underline{K_1},A,B\}$ and $R_2 = \{\underline{K_2},A,C\}$.

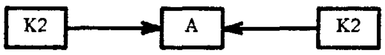


Figure 4-10: The Co-incidental Relationship

K_1 and K_2 both functionally determine A and there is no functional dependency or co-dependency between K_1 and K_2. Let the relationship between K_1, K_2 and A, where K_1 and K_2 both functionally determine A, be called *co-incidental* and K_1 and K_2 be called *co-determiners* on A. The dependencies between K_1 and A and K_2 and A may be transitive. The co-incidental relationship is neither direct nor definite. It is indefinite because there is no definite single K_1-value with which a given K_2-value has the co-incidental relationship, and the reverse is equally true. The co-incidental relationship is clearly indirect since it is based on the mediation of the attribute A. Unlike the role played by the intermediate attribute in a transitive functional dependency, the role of the A attribute in the co-incidental relationship is not to pass along the dependencies to which it is subject to its own dependents. Thus the relationship between co-determiners is much more indirect than that of attributes which are transitively functionally dependent. Since co-incidental relationships are indefinite, much less direct and less likely to be significant than the functional dependency and co-incidental relationships, it is less likely that they will be named in natural language. Tuples in virtual relations such as $R_1 \bowtie R_2$ represent instances of such co-incidental relationships. The relationships that attribute entities in such relations have with the tuple entities are even less likely to be named given that there is probably no simple natural language expression referring to the tuple entities themselves.

Note that if we were to join R_1 and R_2 together with a third relation $R_3 = \{\underline{A}, D\}$ the result would still contain a co-incidental relationship, even though neither $R_1 \bowtie R_2$ nor $R_3 \bowtie R_2$ contain such a relationship. Figure 4-11 illustrates the dependency diagram for $R_1 \bowtie R_2 \bowtie R_3$

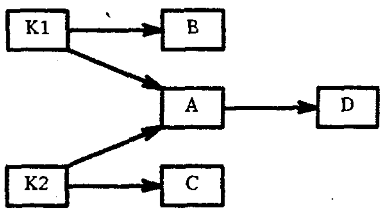


Figure 4-11: Dependency Diagram for $R_1 \bowtie R_2 \bowtie R_3$

This virtual relation contains the same co-incidental relationship between K_1 and K_2 that $R_1 \bowtie R_2$ contains.

Chapter 5
Choosing the Correct Access Path

We develop a strategy to identify the access path in the database which yields the virtual relation corresponding to a given S1 substructure in the query structure of a natural language query. The abstract verb dominating the substructure corresponds on the one hand to the access path which links the attributes corresponding to the NPs of the substructure, and on the other hand to a simple predicate in the natural language query. It is unlikely that the simple natural language predicate will correspond to an access path which links attributes in a co-incidental relationship. Thus, when considering candidate relations we will prefer those which contain only functional dependencies and/or co-dependency relationships to any relation containing a co-incidental relationship. In this chapter we develop a method to implement this strategy.

5.1. The Dependency and Key Join Graphs Defined

Suppose **R** = $\{R_1,R_2,\ldots,R_p\}$ is a database scheme over **U** = $\{A_1,A_2,\ldots,A_n\}$. Let **K** = $\{K_i \mid K_i$ is the key for R_i, $1 \leqslant i \leqslant p\}$. Let **C** = $\{K_j \mid K_j \notin \mathbf{U}$, $1 \leqslant j \leqslant p\}$, the set of all composite keys in **R**. The database *dependency graph* (DG) is a directed graph on nodes **N** = **U**$\cup$**C**. Some nodes (the set **U**) correspond to single attributes while the rest of the nodes (the set **C**) correspond to keys which contain multiple attributes. Note that, since some single attributes may be keys, **K**$\cap$**U** may be non-empty. There is an edge in the dependency graph from node $K_j \in \mathbf{K}$ to $A_i \in \mathbf{U}$ iff $A_i \in R_j$. Each node $K_i \in \mathbf{K}$ is also labelled R_i. Note that if R_i has a composite key K_i there will be a node K_i and edges from K_i to all nodes $A_1, A_2, \ldots, A_q \in K_i$.

Let us call the nodes corresponding to keys, *key nodes*. Since there is a one-to-one correspondence between keys and relations, key nodes are also referred to as *relation nodes*. We generally refer to a key node as a relation node if we are discussing its correspondence to a base relation in the database. Let us call the nodes corresponding to attributes, *attribute nodes*. As noted above, some key nodes may also be attribute nodes.

The *key join graph* for **R** is the subgraph of the dependency graph induced by the set K. An edge in the key join graph between K_i and K_j represents a potential key join between relations R_i and R_j. This join takes place on the key K_j for R_j.

Note the correspondence between the key join graph and the natural join graph. Both have a one-to-one correspondence between nodes and the relations of the DB. The natural join graph has an undirected edge between two nodes if and only if the relations corresponding to the nodes share some attribute, permitting a natural join of the two relations. The key join graph has a directed edge between two nodes if and only if the relation corresponding to the node at the head of the edge contains the key of the relation corresponding to the node at the tail of the edge, permitting a key join of the two relations. We abandon the natural join graph in favour of the key join graph as a representation because we wish to restrict our joins to be key joins in order to guarantee that the virtual relations we create contain only functional dependencies and co-dependencies. We demonstrate later in this chapter the usefulness of the key join graph for selecting access paths in the database which correspond to simple natural language predicates.

Figure 5-1 shows the dependency graph and the key join graphs for the UNIVERSITY database. The dashed line delimits the key join graph.

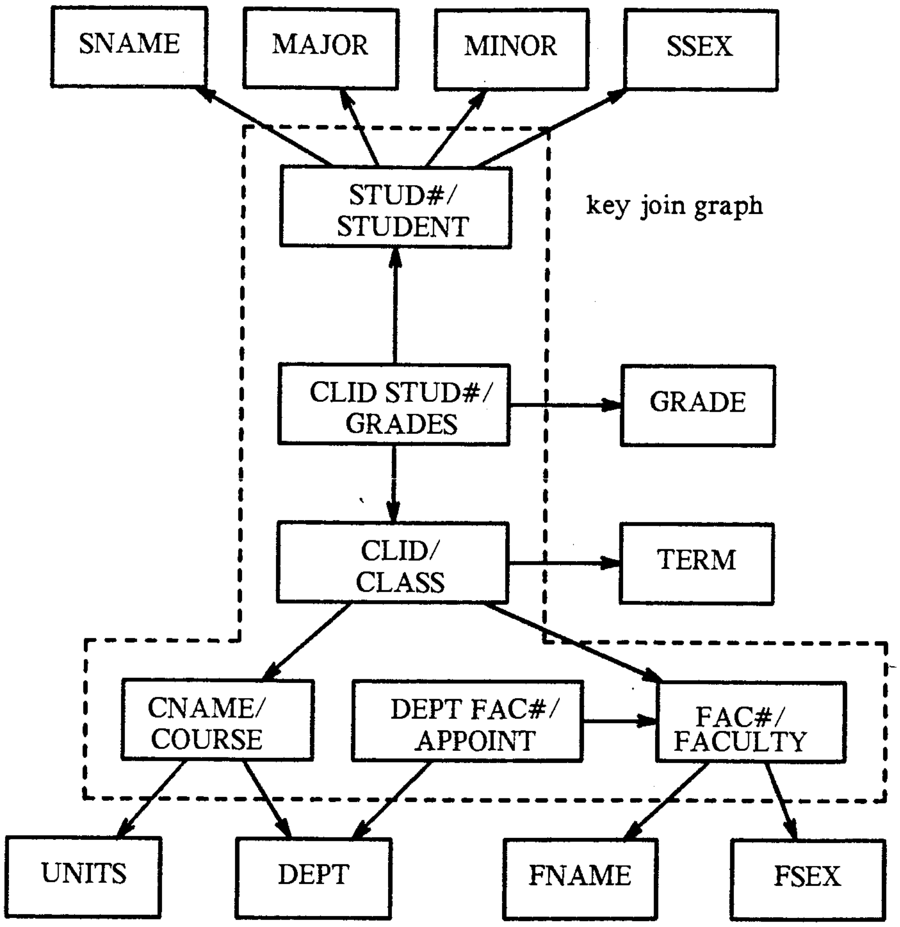


Figure 5-1: The UNIVERSITY Dependency and Key Join Graphs

Note that the dependency graph contains one node corresponding to each of the attributes in the database. In addition, there are nodes corresponding to the two composite keys CLID STUD# and DEPT FAC#. Each key node is also labelled with the name of the relation to which it belongs. There are edges from each of the nodes corresponding to keys in the database to each of the attributes in the corresponding relations.

5.2. Assumptions

We now make explicit our assumptions.

5.2.1. The Fourth Normal Form/Single Candidate Key Assumption

In Chapter 4 we assumed that all relations in the database are in fourth normal form (4NF) and that for each relation the primary key is the only candidate key. This assumption limits the dependency structure of each relation and, thereby, of the whole database. If we restrict the database in this manner the only dependencies that exist in a database relation R are those of the form $K \rightarrow A$ where K is the key for R and $A \in R$.

5.2.2. The Unique Key Name Assumption

In Chapter 4 we stated that key joins result in relations in which all non-key attributes in the resulting relation are functionally dependent on the key of that relation, assuming that the key on which the join takes place refers to the same entity in both the original relations. In this subsection and the next we make explicit the assumption that the key refers to the same entity in both relations.

We assume that if an attribute has the same name as an attribute which is part of a key for a relation then both attributes refer to the same entity. For example, although the keys for the FACULTY and STUDENT relations are both numbers, they cannot both be named NUMBER. Also, if we wished to keep a record of the number of students in each class we could not name the corresponding attribute in the CLASS relation STUD#. Call this the *Unique Key Name Assumption* (UKNA). We must make this assumption if we are going to infer that joining two relations on a key for one yields only functional dependencies and co-dependency relationships between the attributes in the resulting relation. The UKNA allows variables such as FSEX and SSEX in the UNIVERSITY database to both be named SEX, and similarily, FNAME and SNAME to both be called NAME, since none of these attributes are keys in their respective relations.

Note that this is a much weaker assumption than is the Universal Relation Scheme Assumption (URSA) made by the universal relation (UR) approaches mentioned in Chapter 2. Recall that the URSA required that there be only one relationship between any set of attributes. For example, if we made the URSA when designing the UNIVERSITY database, we could not represent both the "teaching for" and "appointee of" relationships between departments and faculty members using only the DEPT and FAC# attributes. One of these attributes would have to be renamed in one of the relations in which it occurs. Not only would this renaming be counter-intuitive but also it would place an additional burden on the natural language parser/interpreter. For example, if one of the occurrences of DEPT was renamed, given the name for a department, the system would be required to decide which of the attributes was being referenced. Even though the department in both cases is the same entity the URSA requires that there be two names. The only restriction required by the UKNA is that attributes which are part of keys not have their names duplicated by other attributes which do not refer to the same entity.

5.2.3. The Surrogate Key Assumption

Let us call an entity which is represented by a composite key a *composite entity*. We assume that there are no composite keys in the database which are functionally dependent. That is, we assume that if a composite entity is functionally dependent on some other entity which is represented in the database then the database administrator will create a surrogate key to represent that composite entity. Call this the *Surrogate Key Assumption (SKA)*.

There are two ways in which a composite entity may be represented other than as a key in a base relation. A composite entity may be represented by a set of attributes which is contained entirely within each of one or more base relations, or the set of attributes may be spread over a number of base relations. Let us consider first the case when the entity is represented entirely within base relations. For example, consider the relation scheme

CLASS = $\{\underline{\text{CLID}},\text{CNAME},\text{TERM},\text{FAC\#}\}$.

Suppose that there is only one section of a course offered during a term. Since a course and a term uniquely identify a class, the following relation scheme represents classes as well as CLASS,

CLASS2 = $\{\underline{\text{CNAME}},\underline{\text{TERM}},\text{FAC\#}\}$.

In CLASS2, {CNAME,TERM} is a composite key representing the composite entity type "class". However, since we desire to refer to classes in other relations, for example,

GRADES = $\{\underline{\text{CLID}},\underline{\text{STUD\#}},\text{GRADE}\}$,

we assume that the database administrator creates the surrogate key CLID and uses it to represent classes.

Adopting the SKA allows sets of attributes in the database to represent more than one type of relationship. Suppose the database administrator wished to keep a record of the majors and the faculty advisors of each student. She may decide to create the following relation scheme,

STUDENT = {<u>STUD#</u>,DEPT,FAC#}.

If the database contained

APPOINT = {<u>DEPT</u>,<u>FAC#</u>,SALARY}

it would appear that a join of these two relations could take place on the key for APPOINT, making explicit the implicit functional dependency between STUD# and SALARY. This would be erroneous because DEPT FAC# represents different entities in the two different relations. We adopt the SKA to avoid this type of error.

When the composite entity is represented in a virtual relation composed of more than one base relation, adding a surrogate becomes slightly more complex. Consider the following database scheme **R**,

R1 = {<u>K1</u>, K2, K3},
R2 = {<u>K2</u>, A},
R3 = {<u>K3</u>, B},
R4 = {<u>A</u>, <u>B</u>}.

The dependency graph for **R** is given in Figure 5-2. We are considering the case where the relationship between the entities represented by A and B is the same in both subgraphs (a) and (b) in Figure 5-2. In subgraph (a) the relationship which exists between a given A-value and B-value pair $<a,b>$ is the co-dependency of a and b on the corresponding K1-value k. Since {A,B} is the key for R4, each pair $<a,b>$ which occurs in R4 is unique. Therefore, given a composite key value $\{a_1\ b_1\}$ from

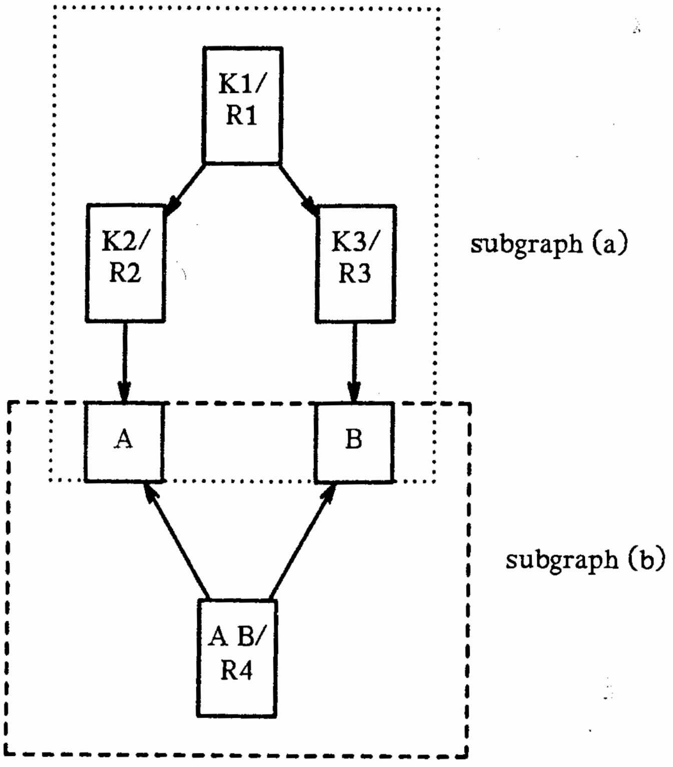


Figure 5-2: Dependency Graph Before Surrogate Added

R4, if the entity represented in R4 by $\{a_1\ b_1\}$ is represented by $\{a_1\ b_1\}$ in R1⋈R2⋈R3, then there is a unique K1-value on which the constituents of this composite key value are co-dependent. Therefore, when {A,B} is replaced by surrogate, S, the database scheme **R** becomes the modified scheme **M** composed of the following relation schemes,

R1 = {<u>K1</u>, K2, K3},
R2 = {<u>K2</u>, A},
R3 = {<u>K3</u>, B},
R4 = {<u>S</u>, K1}.

Figure 5-3 represents the dependency graph for **M**. To illustrate how a composite entity that is represented by a composite key may be also represented by the same attributes in a virtual relation consider the following example. In the UNIVERSITY

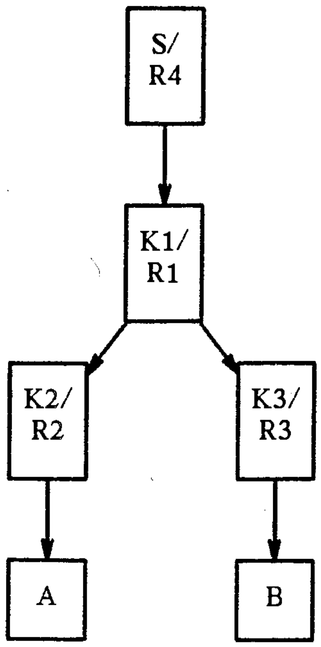


Figure 5-3: Dependency Graph After Surrogate Added

database there is the following co-dependency in which FAC# and DEPT are co-dependent on CLID,

FAC# ← CLID → COURSE → DEPT.

This co-dependency represents the "teaches a class for" relationship between departments and faculty members. Suppose the database administrator created a relation,

SPECASSIGN = {<u>DEPT</u>, <u>FAC#</u>},

to represent a special subset of the instances of this relationship, say, for visiting professors under a special project which restricts the teaching assignment to one class. Since the key for the relation is {DEPT,FAC#} there could only be one such relationship for each pair of faculty members and departments. For each occurrence of this special relationship, there is one and only one class. Therefore {DEPT,FAC#} → CLID when the {DEPT,FAC#}-value is a key in R4. Figure 5-4 show the relevant subset of the dependency graph after the surrogate ASSIGNID is substituted for the composite key.

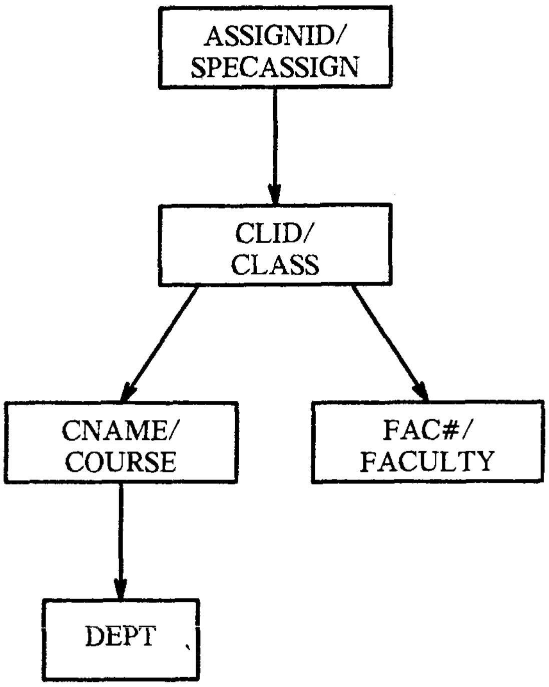


Figure 5-4: Dependency Graph After ASSIGNID Added

We believe the SKA to be a realistic assumption. A composite key represents an attribute entity. If that entity participates in a relationship it is more intuitive to refer to the entity by its own name rather than the names of its components. That is, it is more intuitive to refer to classes by "classes" rather than "course offerings during terms". If a composite entity type *ae* is not referred to outside the base relation R_1 which represents it, it is redundant to add the surrogate. To do so increases the storage requirements of the database by

$$n = (\text{size of surrogate}) \times (\# \text{ of tuples in } R_1).$$

However if *ae* is referred to wholly within base relations outside R_1 the size of the database is likely decreased if the surrogate is added. The size of the decrease will be

$$N((\text{size of composite}) - (\text{size of surrogate})) - n,$$

where N is the number of tuples in the database outside of R_1 containing the surrogate key. N is likely to be greater than the number of tuples in R_1 because each entity of type *ae* will be referred only once within R_1 and will usually be referred to more than once, on average, in the rest of the database. For example, there is only one tuple in the relation *class* for each class represented in the database. However, in the relation *grades* there are many tuples per class, one for each undertaking of each class. Since the size of the composite key is likely to be larger than the size of the surrogate key and N is likely to be greater than the number of tuples in R_1, some storage savings will be realized. There may not be a saving in storage if the composite entity is represented in a virtual relation composed of more than one base relation. The change in storage requirements in this case is

$$-((\text{size of composite}) - (\text{size of surrogate}) - (\text{size of } K_1)) \times (\#\text{ of tuples in } R_1)$$

However, representing a composite entity by a set of attributes shared among a number of base relations is a rare occurrence. Therefore, following the SKA in these cases does not result in unreasonable demands on storage. Since adding the surrogate is more "natural" and usually less costly when composite entities are referred to in more than one relation, we feel justified in making the SKA.

5.3. The Minimum Felicitous Path Strategy

Let us call functional dependencies and co-dependencies *felicitous relationships*. Let us call a virtual relation which contains only felicitous relationships a *felicitous virtual relation*, and an access path which yields a felicitous virtual relation a *felicitous access path*. Let us call access paths and virtual relations which are not felicitous, *infelicitous*. Any infelicitous access path creates a relation which contains at least one co-incidental relationship. If an access path does not contain a co-incidental relationship then it is felicitous because, given our assumptions, there is no relationship possible between attributes connected by an access path in the database other than the functional dependency, co-dependency or co-incidental relationship.

In Chapter 2 we introduced the Minimum Connection Strategy (MCS) to choose between candidate access paths. The MCS is to choose an access path which requires no more joins than any other candidate path. In Chapter 4 we demonstrated that felicitous access paths, (functional dependencies and co-dependencies), are preferable to infelicitous paths, (co-incidental relationships). We combine our preference of felicitous access paths with the MCS to create the *Minimum Felicitous Path* (MFP) strategy. The Minimum Felicitous Path strategy chooses a path which creates only functional dependencies or co-dependencies and which has no more joins than any other candidate path. In this section we demonstrate how to use the key join graph to find the minimum felicitous access path corresponding to a simple natural language predicate.

The state-of-the-art natural language interface system TQA translates a natural language query into a logical form which contains clauses representing the relations which are to be joined to answer the query. These clauses specify the attributes which are to be projected from each relation. The attributes specified include the attributes whose values are desired, the attributes whose values are known, and the attributes on which the joins of the relations will take place. When a relation clause specifies a virtual relation which is composed of attributes from more than one base relation the access path between those attributes must be calculated by the system. The MFP strategy is a suitable strategy for calculating the path. We show that the path may be calculated by calculating the corresponding tree in the key join graph.

Let the attributes in the attribute list of the virtual relation clause whose access path must be calculated be called the *target attributes*. We seek an access path which links the target attributes with a minimum number of key joins. Before presenting the method for calculating a minimum felicitous access path that links the target attributes, we show that a minimum tree in the key join graph that contains nodes representing relations containing a set of attributes A corresponds to a minimum felicitous access path in the database linking the attributes in A.

Since nodes in the key join graph correspond to relations in the database and edges in the key join graph correspond to key joins in the database between the relations represented by the nodes adjacent to the edges, it is obvious that a felicitous access path in the database linking a set of attributes A is represented by a path in the key join graph linking nodes representing relations that contain attributes A. We now show that a tree in the key join graph with n edges corresponds to a felicitous access path in the database with n joins. We leave out references to attributes with the understanding that "access path" refers to "access path linking attributes A", and "path/tree in the key join graph" refers to "path/tree linking nodes representing relations that contain attributes A". Since every edge in the key join graph represents a join in the database, every path of n edges in the graph represents an access path of n joins in the database. However, even though the joins represented by edges in the key join graph are key joins, not every path in the key join graph represents a felicitous access path. Consider the final example from Chapter 4 involving the following small database,

R_1 = $\{\underline{K_1},A,B\}$,
R_2 = $\{\underline{K_2},A,C\}$, and
R_3 = $\{\underline{A},D\}$.

The dependency graph for this database is shown in Figure 5-5.

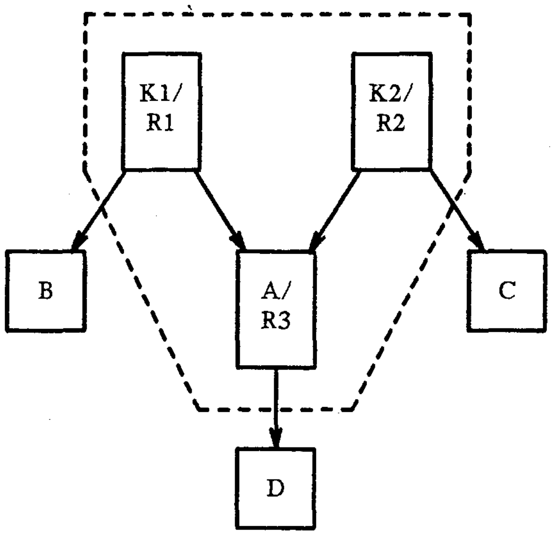


Figure 5-5: Co-incidental Relationship in a Key Join Graph

There is co-incidental relationship between the attributes K_1 and K_2. This relationship is contained in the virtual relation created by the joining of the three relations. If two joins represented by two co-incident key join edges are involved in the creation of the same virtual relation, the second join takes place on a non-key attribute of the virtual relation created by the first join and therefore results in a co-incidental relationship. Such relationships can be avoided by avoiding multiple joins on the same key. Multiple joins on the same key are represented by multiple edges co-incident on the same node in the key join graph. No tree has any co-incident edges, therefore an access path corresponding to a tree contains no co-incidental relationships and is felicitous. Similarly, a felicitous access path contains no co-incidental relationships. Therefore, a felicitous access path involves only key joins where no two joins take place on the key of the same relation. Therefore, a felicitous access path is represented by a tree in the key join graph. Since every join in the database corresponds to an edge in the key join graph, a felicitous access path involving n joins is represented by a tree of n edges in the key join graph.

A minimum tree in the key join graph corresponds to a minimum felicitous access path in the database. Assume this is not the case. Then a minimum tree in the key join graph which contained n edges would correspond to a felicitous access path in the database containing n joins and the minimum felicitous access path would contain m joins, where $m < n$. The minimum felicitous access path would therefore correspond to a tree in the key join graph containing m edges. Therefore the minimum tree would not be minimum. Therefore the assumption is wrong and the minimum tree in the key join graph corresponds to a minimum felicitous access path in the database.

Finding a minimum tree containing nodes representing relations containing a set of target attributes is a variation of the Minimum Directed Steiner Tree (MDST) problem in graphs. Given a connected digraph $G = (V,E)$, and a subset of the vertices $X \subseteq V$, the MDST in graphs is to find a rooted tree $T = (W,F)$ in G with arcs directed out from the root and $|F|$ minimized such that $X \subseteq W \subseteq V$ and $F \subseteq E$. The corresponding decision problem was proved NP-complete in (Karp, 1972).

Because the number of target attributes in a virtual relation clause is limited, the problem of finding the minimum tree in the key join graph is simpler than the general MDST problem. The number of target attributes in a relation clause does not exceed five because the number of NPs in the corresponding S1 substructure does not exceed five (Petrick, 1973). In fact, in most cases, there are only two or three NPs in a S1 substructure. There is an efficient linear time algorithm for solving the MDST problem in graphs when $|X| = k$, for any small, fixed k (Liestman & Richards, 1986). First, we create a *target node* for each target attribute and add the node to the key join graph. Then an edge is added to connect each target node to every relation node corresponding to a relation that contains the target attribute. The minimum Steiner tree connecting the target nodes is found using the Liestman/Richards algorithm. Let this tree be called the *preliminary tree*. The target nodes and their adjacent edges are removed from the preliminary tree. The remaining tree, the *join tree*, is a minimum tree in the key join graph corresponding to a minimum access path in the database connecting the target attributes.

The preliminary tree is more useful than the join tree for specifying the complete access path for the virtual relation specified in a virtual relation clause. The target attributes consist of those attributes which belong to the virtual relation and are referred to in the natural language query. These attributes are either known attributes, desired attributes, or attributes on which the virtual relation is joined to other relations in the query access path. Thus they must appear in a complete specification of the operations needed to create the virtual relation. Therefore we work with the preliminary tree which contains both the target nodes corresponding to the target attributes and the relation nodes corresponding to the relations which are to be joined. An edge connecting a target node to a relation node in the preliminary tree represents the projection of the corresponding relation on the corresponding target attribute. An edge connecting two relation nodes in the preliminary tree represents the join of the corresponding relations on the attribute which is the key of the relation represented by the node at the tail of the edge. Thus this edge represents the projection of both corresponding relations on this attribute. If one of the target attributes is the same as one of the attributes on which a join takes place then that attribute will be represented twice in the preliminary tree, once in a target node and once in a key/relation node. We eliminate the redundant nodes by traversing the tree and checking to see if any interior (key) nodes of the tree have the same name as a target node, and removing any target nodes where this is the case. The duplicate key node must be interior because if a join is required to include an attribute, the required attribute must be different from the attribute on which the join takes place and the node corresponding to the required attribute will be the child of the duplicate key node. The *final tree* which results when the redundant target nodes are removed is a subtree of the dependency graph, consisting of the join tree which is a subtree of the key join graph and some number of non-key attribute nodes connected to relation nodes in the join tree representing relations which contain the corresponding non-key attributes. Since adding and deleting target nodes to the join tree does not repesent the inclusion or elimination of joins in the minimum felicitous access path represented by the join tree, the path represented by the final tree is a minimum felicitous access path which links the target attributes.

We now give an example of how a virtual relation clause is transformed into a final tree corresponding to a minimum felicitous access path for the virtual relation specified by the clause. Figure 5-6 contains a virtual relation clause corresponding to the simple English predication "the Math 100 students' names"

```
(STUDENTS*
   (STUD# CNAME NAME)
   (x4  math100  x8)
   ( =   =   = ) )
```

Figure 5-6: Virtual Relation Clause Example

Figure 5-7 shows the key join graph of the UNIVERSITY database with the target nodes corresponding to the target attributes from the attribute list of the virtual relation clause depicted in Figure 5-6. Figure 5-8 contains the two possible preliminary trees that could be generated from the graph in Figure 5-7. Figure 5-9 depicts the final tree that results when the redundant STUD# target node is pruned from either of the candidate preliminary trees.

We now give an example of how the access path corresponding to a final tree is calculated and show how the MFP is an improvement over the MCS. In Chapter 2 we saw that the MCS alone was unable to choose a unique access path for Query (2.2). We repeat Query (2.2), for convenience.

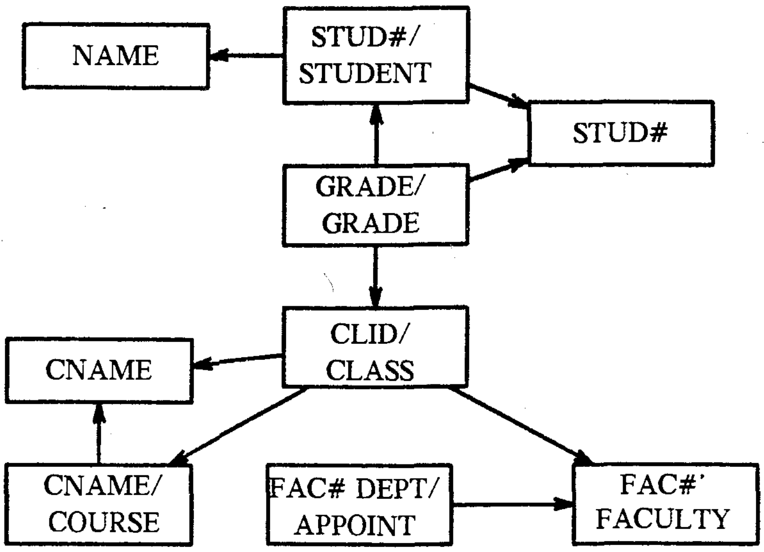


Figure 5-7: Key Join Graph with Target Nodes Added

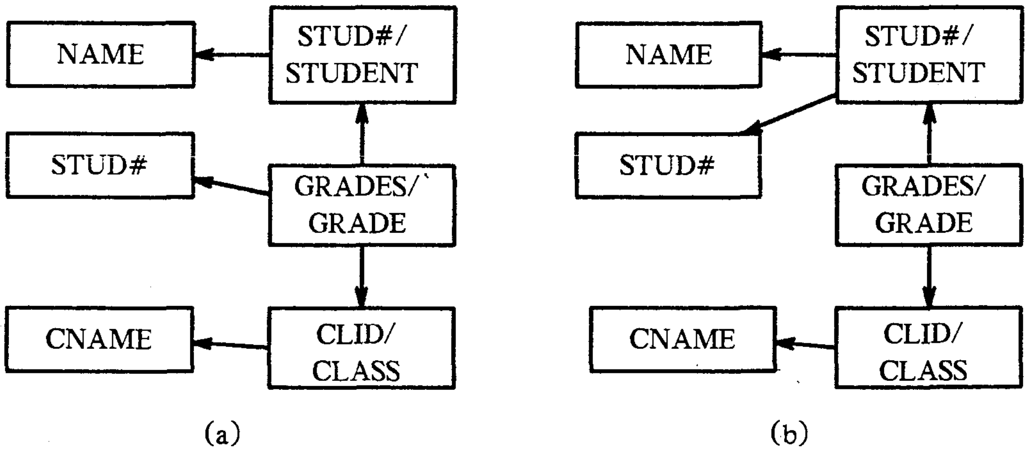


Figure 5-8: Candidate Preliminary Trees

Retrieve (FNAME) where UNITS=3 (5.1)

Figure 5-10 duplicates Figure 2-8 and shows the alternative minimum join paths which exist in the database for Query (5.1). Figure 5-11 shows a logical form relation clause containing the same information as Query (5.1). The relation is arbitrarily assigned the name "VIRTUAL1". It contains the attributes UNITS and FNAME. The UNITS-value of each tuple in the relation is "3". The set of

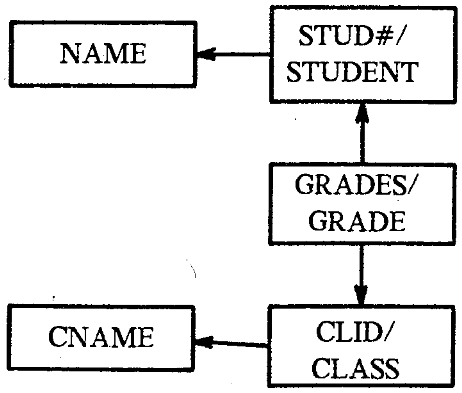


Figure 5-9: Final Tree

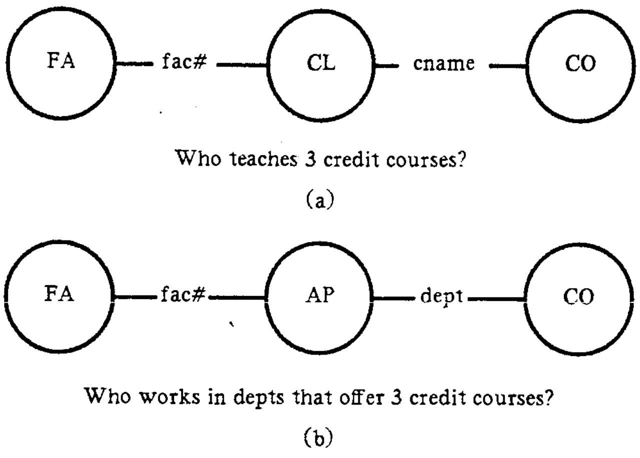

FA
fac#
CL
cname
CO
Who teaches 3 credit courses?
(a)
FA
fac#
AP
dept
CO
Who works in depts that offer 3 credit courses?
(b)

Figure 5-10: Minimum Paths for Query ((5.1))

FNAME-values in the virtual relation is unknown, and, after the relation has been derived, will be assigned to "x4".

```
(RELATION VIRTUAL1
  (UNITS FNAME)
  (3 x4)
  (= =) )
```

Figure 5-11: Logical Form equivalent to Query ((5.1))

Using identical information the MFP is able to find a unique access path where the MCS is unable to do so.

As can be easily verified by examining Figure 5-1, the final tree that would be produced for the current example is the tree shown in Figure 5-12.

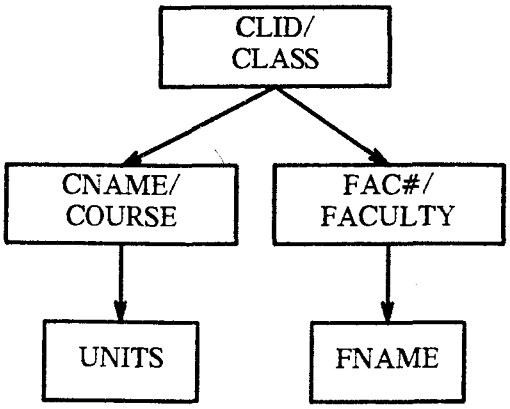


Figure 5-12: Tree for Logical Form in Figure 5-11

Once the tree has been computed, it is necessary to translate it into logical form so that it can be substituted into the query logical form for the virtual relation clause corresponding to the tree. TRANSLATE_TREE, (shown below), is a pseudocode algorithm which translates final trees into a list of logical form relation clauses. The variables used in the program include:

- TREE - the final tree
- CURRENT_NODE - node currently being visited in traversal of the tree
- MASTERLIST - list of entries each containing an attribute name, the value associated with that attribute, and the relational operator (=, <, >, ...) linking the name to the value. Values may be variables or constants. For example, the masterlist corresponding to the clause in Figure 5-11 is

  ((UNITS = 3) (FNAME = x4)).

- CURRENT_RELATION_CLAUSE - If the current node is a relation node, the program builds a corresponding relation clause; the current relation clause is the clause currently being built.

- CURRENT_ATTRIBUTE - After the attribute list for the current relation clause has been built, the attributes in the list are checked against the masterlist to see if their values (and relational operators) are known; the current attribute is the attribute from the list which is currently being checked.

- CURRENT_VALUE_LIST - the list of attribute values under construction for the current relation clause.

- CURRENT_OPERATOR_LIST - the list of relational operators under construction for the current relation clause.

- CLAUSELIST - the list of base relation clauses corresponding to the tree being translated. This list will be substituted into the query logical form for the virtual relation clause corresponding to the input tree.

```
TRANSLATE_TREE:
CREATE_MASTERLIST;
traverse TREE, if CURRENT_NODE is a key node do,
      CREATE_RELATION_CLAUSE,
      ADD_ATTRIBUTE_LIST,
      ADD_VALUE_LIST,
      ADD_OPERATOR_LIST,
      UPDATE_MASTER_LIST;
      ADD_RELATION_CLAUSE;

CREATE_MASTERLIST:
initialize MASTERLIST with information from virtual relation
clause;

CREATE_RELATION_CLAUSE:
create CURRENT_RELATION_CLAUSE with the relation name from
CURRENT_NODE;

ADD_ATTRIBUTE_LIST:
create a list of every attribute name in nodes adjacent to
CURRENT_NODE and make this list the attribute list of
CURRENT_RELATION_CLAUSE;

ADD_VALUE_LIST:
search MASTERLIST for attributes in CURRENT_RELATION_CLAUSE,
if CURRENT_ATTRIBUTE is in MASTERLIST then
   add value from MASTERLIST entry to CURRENT_VALUE_LIST
   else create unique variable and add it to CURRENT_VALUE_LIST;
make CURRENT_VALUE_LIST value list of CURRENT_RELATION_CLAUSE;

ADD_OPERATOR_LIST:
search MASTERLIST for attributes in CURRENT_RELATION_LIST,
if CURRENT_ATTRIBUTE is in MASTERLIST then
   add operator from MASTERLIST entry to CURRENT_OPERATOR_LIST
   else add "=" to CURRENT_OPERATOR_LIST;
make CURRENT_OPERATOR_LIST operator list of
CURRENT_RELATION_CLAUSE;
```

```
UPDATE_MASTERLIST:
for each attribute in CURRENT_RELATION_CLAUSE,
     if it does not have an entry in the MASTERLIST then
         create an entry and add it to MASTERLIST;

ADD_RELATION_CLAUSE:
add CURRENT_RELATION_CLAUSE to CLAUSELIST;
```

Given the tree from Figure 5-12, and the logical form in Figure 5-11, TRANSLATE_TREE will create the list of clauses shown in Figure 5-13.

```
( (RELATION CLASS
      (CNAME FAC#)
      (x10 x20)
      (= =) )
  (RELATION COURSE
      (CNAME UNITS)
      (x10 3)
      (= =) )
  (RELATION FACULTY
      (FAC# FNAME)
      (x20 x4) ) )
```

Figure 5-13: Translation of Tree in Figure 5-12

Note that the subroutines ADD_VALUE_LIST, ADD_ATTRIBUTE_LIST and UPDATE_MASTERLIST can be merged into one routine so that only one pass through the attribute list of the current relation clause is made.

Chapter 6

Observations and Future Research

The observations made in this chapter are speculative and require further investigation. However, they indicate what appears to be an exciting, natural and promising extension of the work reported in the thesis thus far.

6.1. Using Lexical Information to Disambiguate Verbs

The minimally connected access path is not always the correct access path. Consider the following query to the UNIVERSITY database.

"Which faculty members teach in the Math department?"

The minimum tree in the key join graph corresponds to the base relation APPOINT and this interpretation results in the return of the Math department appointees. It is possible, however, to use information attached to verbs in the lexicon to make better choices in cases such as this.

As mentioned in Chapter 3, the TQA team found that attaching detailed grammatical information to verbs in the lexicon was too great an expectation for linguistically naive database admininstrators. However, it is not unreasonable to expect that the customization process will include the attachment, to the lexical entries for verbs, of the names of the attributes corresponding to the noun phrase arguments of those verbs. For example, one entry for the verb "teach" would include the information that it takes two arguments, corresponding to FAC# and CLID names and/or values. This information is required to decide that the pronoun in the following query refers to faculty members.

"Who teaches for the Math department?"

This capability is essential for a practical natural language interface. The creation of the lexical entries for the verbs corresponding to the functional dependencies between the key and non-key attributes of the base relations is relatively simple. There are relatively few such dependencies, one for each edge in the dependency graph. The database admininstrator, upon examining these dependencies, possibly being prompted with them by the customization system, could enter the relevent verbs and attach the information about their arguments.

Given this information about the arguments of the verbs which correspond to the functional dependencies in the base relations, the system would be able to make more reasonable choices when faced with those verbs in substructures containing NPs corresponding to attributes from more than one base relation. For example, when searching for the correct access path corresponding to the verb in the following query,

"Which faculty members teach in the Math department?",

it would prefer a path which includes the functional dependency CLID → FAC# that is attached to the lexical entry for "teach". Thus it would select

DEPT ← CNAME ← CLID → FAC# over
DEPT ← {DEPT,FAC#} → FAC#.

Note that the correct choice is a co-determinancy relationship that is based on the functional dependency between CLID and FAC#. In the English query, "Math department" refers to a set of courses, which in turn refer to a set of classes to which the faculty members in question are to be in the "teach" relationship. By keying on the functional dependency indicated by the verb the system selects the co-determinancy corresponding to the query over the alternative which is more minimal.

There are a number of ways to implement this heuristic. The system could find all trees and then, if there were more than one, choose the minimum tree containing the desired edge. Otherwise, it could include the nodes adjacent to the preferred edge in the target graph and thus ignore more minimal trees which do not include it.

Similarly, it would be useful and practical to attach to the lexical entry for a noun which refers to only one possible role of an attibute sufficient information to allow the system to select the corresponding path. For example, consider the attribute FAC#. This attribute represents an entity which has more than one role represented in the database. That is, the attribute is at the tail of more than one edge in the dependency graph. The two roles corresponding to the two edges incident on FAC# are "teacher" and "appointee". Some of the natural language noun phrases corresponding to this attribute do not distinguish clearly between the two roles, eg., "faculty member" and "professor". Attached to the lexical entries for these two phrases would be the information that they correspond to the attribute FAC#. However, noun phrases such as "appointee" and "teacher" specify only one of the possible roles. In these cases, the name of the node at the tail of the edge would be included in the lexical entry, {DEPT,FAC#} for "appointee" and CLID for "teacher". When interpreting a query containing one of these noun phrases, both corresponding nodes would be included in the list of target nodes. In this manner, the system would select the longer access path for the query

"Who are the Math teachers?"

and the shorter one for

"Who are the Math appointees?".

6.2. Managing Multivalued Co-dependencies

Up until now we have ignored multivalued dependencies in our examination of the correspondence between natural language predicates and database predicates. We consider multivalued dependencies in this section.

A *multivalued dependence* is defined in (Smith, 1985) as follows: "There is a multivalued dependence from A to B ($A \twoheadrightarrow B$) if, at any point in time, a fact about A determines a *set* of facts about B". For example, a class may have many students. Each combination of A and B must be unique, that is, {A,B} must be a key in a relation. For example, {CLID, STUD#} is the key for GRADES. A given student determines a set of classes, therefore STUD# $\twoheadrightarrow$ CLID. A given class determines a set of students, therefore CLID $\twoheadrightarrow$ STUD. As pointed out by Smith, this definition is not in exact accord with previous definitions.[9] However, this definition is derived from and is similar to the previous definitions; it makes good common sense and suits our purpose.

[9] See Appendix A for a previous definition of multivalued dependencies.

6.2.1. Multivalued Co-dependencies

Let us call the members of a set of attributes (entities) **X**, which are all multivalued dependents of the same attribute (entity) *A*, *multivalued co-dependents* and the relationship among the members of **X** and *A* a *multivalued co-dependency*. In order to be clear, we replace the term "co-dependents" and "co-dependency" with *functional co-dependents* and *functional co-dependency* when we refer to attributes which are functionally dependent on the same key and to the relationship among the attributes and the key. The following relation schemes represent a canonical multivalued co-dependency.

$R_1 = \{\underline{A},\underline{B}\}$,
$R_2 = \{\underline{A},\underline{C}\}$.

We represent multivalued dependencies in dependency diagrams by light double headed arrows from the determiner to the dependents. The dependency diagram for R_1 and R_2 is given in Figure 6-1. Since a given *A*-value in R_1 determines a set of *B*-values, and vice versa, there is a light double headed arrow from node A to node B in the dependency diagram, and vice versa. Similarly, since a given *A*-value in R_2 determines a set of *C*-values, and vice versa, there is a light double headed arrow from node A to node C in the dependency diagram, and vice versa.

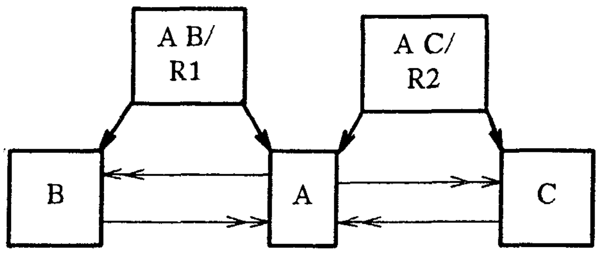


Figure 6-1: Canonical Multivalued Co-dependency

Note that the relationship between the multivalued co-dependents B and C is infelicitous, that is, B and C are not linked in a functional dependency or a functional co-dependency.

We will use the following GRADSCHOOL database scheme to inform our discussion of multivalued co-dependencies.

STUDENT = {STUD#, SNAME, SSEX},
FACULTY = {FAC#, FNAME, FSEX},
CLASS = {CLID, TERM},
ASSIGN = {CLID, FAC#},
GRADES = {CLID, STUD#, GRADE},
SUPERVISE = {FAC#, STUD#}.

This scheme contains the following multivalued dependencies,

CLID $\twoheadrightarrow$ STUD#,
CLID $\twoheadrightarrow$ FAC#,
STUD# $\twoheadrightarrow$ CLID,
FAC# $\twoheadrightarrow$ CLID,
STUD# $\twoheadrightarrow$ FAC#, and
FAC# $\twoheadrightarrow$ STUD#

Dependency diagrams for the three multivalued co-dependencies contained in the GRADSCHOOL database are shown in Figure 6-2. Intuitively, the GRADSCHOOL database is a representation of a school where there is more than one student in a class and where there can be more than one teacher of a class. In this school students may attend more than one class, and faculty members may teach more than one class. Furthermore, a faculty member supervises more than one student and students have more than one supervisor.

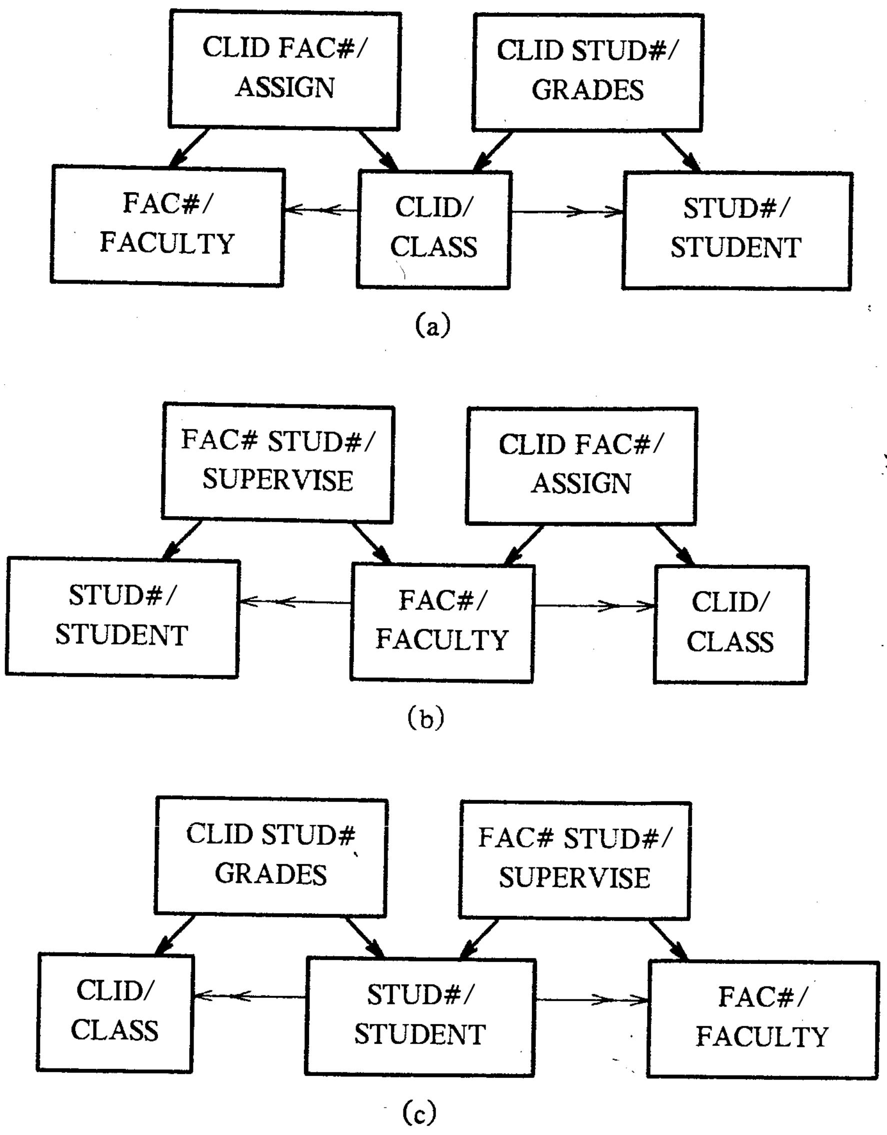


Figure 6-2: Multivalued Co-dependencies in the GRADSCHOOL Database

6.2.2. Multivalued Co-dependencies and Natural Language

Although the multivalued co-dependency is an infelicitous relationship and weaker than the functional co-dependency, it is possible to have simple natural language expressions corresponding to some multivalued co-dependencies. Consider the functional co-dependency $\mathrm{FAC\#} \leftarrow \{\mathrm{FAC\#}, \mathrm{STUD\#}\} \rightarrow \mathrm{STUD\#}$, where $\{\mathrm{FAC\#}, \mathrm{STUD\#}\}$ is the key for the relation SUPERVISE. A given FAC#-value in the SUPERVISE relation specifies a set of {STUD#, FAC#}-values of which is predicated a set of STUD#-values. This functional co-dependency reflects the supervisory relationship between faculty members and students. Compare this functional co-dependency with the multivalued co-dependency $\text{FAC\#} \twoheadleftarrow \text{CLID} \twoheadrightarrow \text{STUD\#}$. A given FAC#-value specifies a set of CLID-values of which is predicated a set of STUD#-values. This multivalued co-dependency reflects the teaching relationship between faculty members and students. The multivalued co-dependency relationship is not a strong as the functional co-dependency relationship because in the functional co-dependency the relationship between the determiner and its corresponding set of dependents is one-to-one, while in the multivalued dependency the relationship between the determiner and its corresponding set of dependents is one-to-many. For example, in the functional co-dependency, $\text{FAC\#} \leftarrow \{\text{FAC\#},\text{STUD\#}\} \rightarrow \text{STUD\#}$, a given FAC#-value specifies a set of supervisory relationships, and there is a one-to-one relationship between each supervisory relationship and the student which comprises it. In the multivalued co-dependency, $\text{FAC\#} \twoheadleftarrow \text{CLID} \twoheadrightarrow \text{STUD\#}$, a given FAC#-value specifies a set of classes, and there is a one-to-many relationship between each class and the students which comprise it. Thus, when we speak of "Jones' students" in the graduate school context we are more likely to assume that "Jones" is specifying the stronger relationship between faculty members and classes, that is, the supervision relationship. However, if we speak of "Jones' Math 891 students" we know that we are referring to the teaching relationship.

The virtual relation corresponding to the phrase "Jones' Math 891 students" is

$\pi_{\text{STUD\#}}\sigma_{\text{FAC\#}=85110\text{-}1000,\text{CLID}=m891/861/s1}\mathit{teach} \bowtie \mathit{grades}.$

We are using the faculty number to represent Jones and the class id of the current offering of Math 891 to represent the course for simplicity. Note that the access path corresponding to the virtual relation joins *assign* with *grades*. In our canonical example in Figure 6-1, the access path which yields the virtual relation containing the multivalued co-dependency contains $R_1 \bowtie R_2$. As we discover below, the relationship between the entities represented by the relations which are joined to create the virtual relation containing an multivalued co-dependency determines whether the multivalued co-dependency has a corresponding simple natural language expression.

Not all multivalued co-dependencies have corresponding simple natural language expressions. For example, it seems that neither of the multivalued co-dependencies represented by the dependency diagrams in Figures 6-2(b) and (c) can be expressed by a simple natural language expression. The relationship between a student and the classes taught by his supervisor and the relationship between a faculty member and the classes attended by his supervisees are not strong or direct enough to warrant such an expression. However, as Figure 6-2 mades clear, the dependency relationships in a multivalued co-dependency which has a corresponding natural language expression appear identical to the dependency relationships in the multivalued co-dependencies which have no such expression.

6.2.3. Strong Multivalued Co-dependencies

Let us call a multivalued co-dependency which has a corresponding natural language expression a *strong multivalued co-dependency.* We refer to a multivalued co-dependency which does not have a corresponding natural language expresssion a *weak multivalued co-dependency.* Strong multivalued co-dependencies do have different dependency relationships from the relationships in weak multivalued co-dependencies. They differ in a way which the diagrams of Figure 6-2 do not show. We have said that the tuples in a *grades* relation represent undertakings. Let us refer to the relationships that are represented in the relation ASSIGN as "assignments". Let us refer to the relationships between faculty members and students which are represented in the relation SUPERVISE as 'supervisions'. One way of describing the significant difference between strong multivalued co-dependencies and weak multivalued co-dependencies is to say that there is a dependency between the composite entities that make up strong multivalued co-dependencies, and this is not the case with weak multivalued co-dependencies. For example, consider the multivalued co-dependency of Figure 6-2(a), which represents the teaching relationship between faculty members and students. This relationship is based on the intermediating relationship between the composite entities assignments and undertakings. There is a dependency between assignments and undertakings. For any assignment there must be a corresponding non-empty set of undertakings, given the make-up of the graduate school. Given an undertaking there must be a corresponding non-empty set of assignments. These multivalued dependencies are dependencies that exist between entities in the world. One can imagine that an assignment of a teacher to a class takes place before any registration of students in that class. Therefore, in the database there may be a *assign* tuple without any corresponding *grades* tuples. However, in the world an assignment implies undertakings, real or imminent. Thus, we say that an assignment determines a set of undertakings, and an undertaking determines a set of assignments. There is no strong relationship between the composite entities which make up the multivalued co-dependency depicted in Figure 6-2(b). Given an assignment there need not be any corresponding supervisions. Given an supervision there need not be any corresponding assignments. The same holds true for supervisions and undertakings, the composite entities which comprise the multivalued co-dependency depicted in Figure 6-2(c).

Another way of describing the difference between strong and weak multivalued co-dependencies is to say that the entities represented by the attributes composing strong multivalued co-dependencies together comprise a third entity, whereas this is not the case with weak multivalued co-dependencies. In the strong multivalued co-dependency represented in Figure 6-2(a), undertakings and assignments cohere to form a third entity, which we might refer to as an "instruction". An instruction is the product of a bringing together of a teacher and a student in the same class. An instruction is an entity which, although abstract, is easily recognizable. Consider now the multivalued co-dependency represented in Figure 6-2(b). The entities represented in this diagram, that is, assignments and supervisions, do not cohere to form a recognizable entity. It is the strength and closeness of the coherence between undertakings and assignments that leads to the fact that there are simple natural language expressions which refer to relationships between the entities which comprise them. It is a lack of coherence between assignments and supervisions which accounts for the fact that there are no natural language expressions corresponding to the relationships between the entities that comprise assignments and supervisions.

6.2.4. Representing Strong Multivalued Co-dependencies

The dependencies which distinguish strong multivalued co-dependencies from weak ones often are not represented in a database scheme or its corresponding key join graph. If access paths are to be generated to yield virtual relations containing strong multivalued co-dependencies, these dependencies must be represented. If we are to use the method developed in Chapter 5, the representation of these dependencies must be compatible with that method. We present two approaches for representing these "hidden multivalued co-dependencies".

The first approach requires a change in our algorithm for translating dependency graph trees into logical form access path specifications. The second approach requires a change to the database itself. Both approaches have an impact on the key join graph.

Our first approach follows from the observation that there are multivalued dependencies between the composite entities which comprise the strong multivalued co-dependency. We represent these dependencies in the key join graph by light double headed edges between the nodes representing the corresponding composite keys. The key join graph for the GRADSCHOOL database is depicted in Figure 6-3. Since we prefer functional co-dependencies over multivalued co-dependencies we weight the edges, giving the multivalued dependency edges a greater weight than the functional dependency edges. Thus the problem of finding the minimum tree in the key join

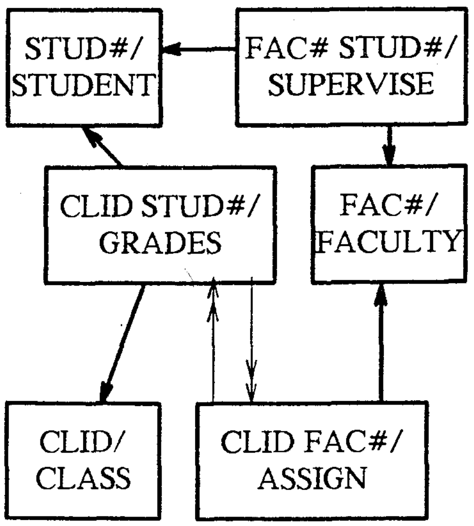


Figure 6-3: GRADSCHOOL Key Join Graph

graph corresponding to the virtual relation in the database specified by a simple natural language predicate in the query becomes the Minimum Directed Cost Steiner Tree (MDCST) problem. This problem subsumes the Minimum Cost Steiner Problem. The algorithm in (Liestman & Richards, 1986) solves the MDCST in linear time when the target nodes number less than seven. Therefore we may use this same algorithm after including the multivalued dependency edges. Further research is required to assess how much weight to assign to the two different types of edge. Below we offer an argument for assigning a weight of 2(weight of the functional dependency edges) to the multivalued dependency edges. The algorithm for translating the resulting trees into logical form specifications of access paths must change if we use this approach because a join represented by a multivalued dependency edge does not take place on the attribute corresponding to the node at the tail of the edge. The join takes place on the attribute which is the intersection of the composite keys represented by the nodes adjacent to the edge.

The second method for representing hidden multivalued dependencies follows from the observation that the composite entities which comprise the multivalued co-dependency cohere to form a third entity. A relation is added to the database to represent instances of this third entity. In order to abide by the restrictions of the surrogate key assumption, surrogate keys are created for the composite keys of the relations representing the original composite entities. The key for the new relation is a composite of these surrogates. The following relation scheme reflect the modifications that this approach has on the GRADSCHOOL database.

ASSIGN = {<u>ASSIGNID</u>,CLID,FAC#}
GRADES = {<u>GRADEID</u>,CLID,STUD#,GRADE}
INSTRUCT = {<u>GRADEID</u>,<u>ASSIGNID</u>}

Figure 6-4 shows the modified GRADSCHOOL dependency graph. Note that the relationship between STUD# and FAC# is now a functional co-dependency. Thus, the analysis that strong multivalued co-dependencies are strong because the composite entities compising them together form a third entity implies that strong multivalued co-dependencies are implicit functional co-dependencies. If this analysis is valid, then if we were to use the first approach the appropriate weight for the multivalued dependency edges is 2(weight of the functional dependency edges), because these edges represent the implicit joins of the two relations represented by their adjacent nodes to the third, implicit relation.

Creation of the new relation could be done automatically by a sophisticated DBMS such as would be necessary to implement RM/T (Codd, 1979). The DBMS would be required to update and delete tuples from the new relation whenever additions and deletions from the original composite relations warranted. Note that to create the

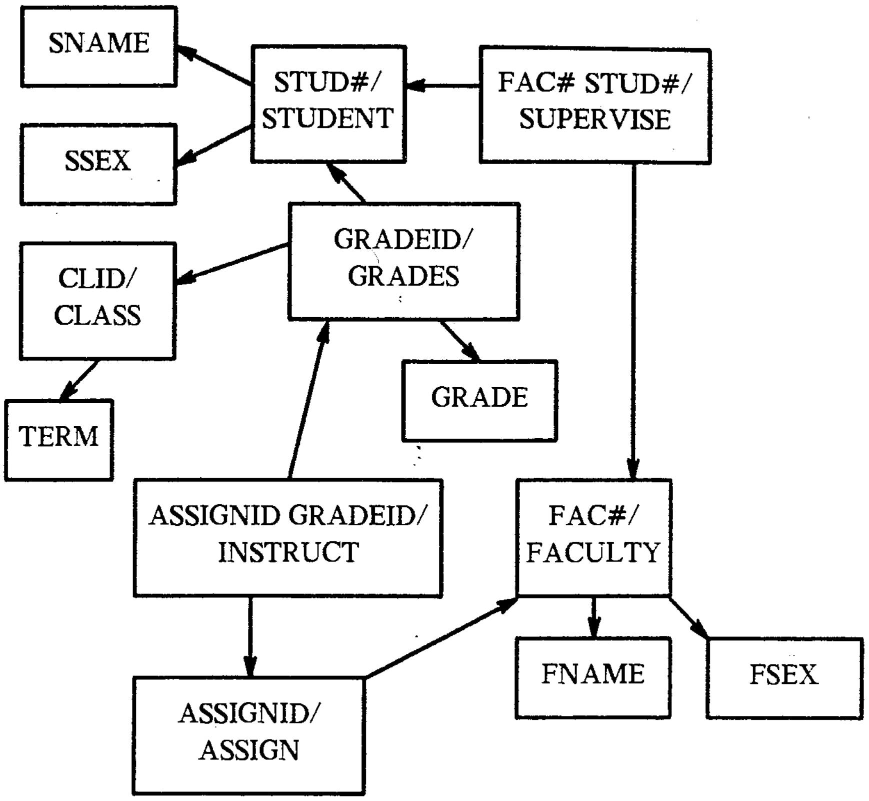


Figure 6-4: Modified GRADSCHOOL Dependency Graph

new relations the system must be informed by the database admininstrator of the existence of the multivalued dependencies between the composite entities, for example, of the multivalued dependency between assignments and undertakings.

6.3. Multiple Candidate Keys

One of the most unrealistic assumptions on which our method is based is the assumption that there is only one candidate key for a given relation. This assumption could be relaxed by assuming instead that only primary keys are used to represent the tuple entity in other relations and treating other candidate keys like any other non-key attributes. It may be that the method could handle having all candidate keys acting like primary keys in the sense that they could be used as connections between the relations in which they are keys and other relations in the database. This would complicate the dependency graph and research is needed to see what, if any, alterations to our approach to choosing access paths is necessary. Similarly, further research is needed to see how relaxing the 4NF assumption affects the results.

6.4. Generating Natural Language

Paraphrasing access paths in natural language and feeding them back to the user is a common technique for validating query interpretations. Responding to user queries in natural language may be more appropriate then listing data at times. More natural and comprehensible discourse may be generated by using lexical information about natural language predicates and knowledge about the database structure as contained in the dependency graph. For example, paraphrases such as "Does Prof Jones teach John Wong?" might be generated as an alternative to "Does Prof Jones teach classes undertaken by John Wong?"

Chapter 7
Conclusions

In this thesis, we have developed an approach to solving the MAPP in the context of natural language interfaces to relational databases.

We have shown that the MAPP in the TQA system is confined to choosing paths which yield virtual relations that correspond to simple natural language predicates. We have argued that the rules governing propositional speech acts will cause human users of natural languages interfaces to express their queries in such a manner that the MAPP will be confined in this way for natural language interfaces in general.

We have argued that the entities referred to in simple natural language predicates have a definite, direct relationship with each other, and that this relationship is of significance to the human users of the natural language. We have argued further that this type of definite, direct relationship between entities will be reflected in the database by the relationship of the attributes corresponding to the entities.

We have characterized the relationships between attributes in the database based on an informal semantics for relational databases that we have developed. We have shown how a certain class of these attribute relationships, the felicitous relationships, which arose from our characterization, correspond to simple natural language predicates. Based on the correspondence between simple natural language predicates and attribute relationships in the database, we developed a strategy for making reasonable guesses of the access paths which yield the virtual relations referred to in natural language database queries by simple predicates.

We developed a rich representation of the database which embodies the syntactic and semantic relationships among attributes and relations in the database. We provided a heuristic method to map a simple natural language predication to this representation and thereby derive an image of an access path in the database which yields a virtual relation corresponding to the predication. We provided an algorithm to translate the image into a logical form specification of the access path.

Our method for deriving access paths is suitable for cyclic databases, which is an improvement over previous methods used in natural language interface systems. It does not require an undue or impractical amount of customization of the natural language interface to a particular application.

Appendix A

A.1. Database Basics

A computer *database* is a collection of data. A set of programs which allows a user to manipulate, store and retrieve the data in a database is called a *database management system* (DBMS).

One major purpose of a DBMS is to eliminate the requirement that users know how data is physically stored in the computer. A DBMS which provides this feature provides *physical data independence*. Physical data independence is accomplished by presenting users with a view of the data, called the *user view*, which represents the database in terms of data structures which are easier for users to manipulate. Intermediate between the external views and the physical database is the *conceptual database*. The conceptual database may be merely the amalgamation of the external views, or it may include additional information useful to others who are involved in the enterprise which the database supports, such as the database administrator (DBA).

Historically, there have been three main approaches to the design of the conceptual database and the user views: the *relational*, the *network* and the *hierarchical* approaches. The hierarchical approach is often viewed as a restricted case of the network approach (Date, 1981). The relational approach is generally accepted to have advantages over the network and hierarchical approaches (Date, 1981), (Ullman, 1982a). One of its advantages is its greater ease of use; relational databases are easier to understand and manipulate. In addition, the relational approach has a strong theoretical basis, including mathematical set theory as well as a large body of its own theory. For these reasons, most database research since 1970 follows the relational approach. This is the approach that we observe in this thesis.

A.2. Relational Databases

We follow the notation that is described in (Maier, 1983). Much of what follows is a paraphrase of part of the first six chapters of that work.

A.2.1. Intuitions

The simplicity of the relational approach results from viewing all data as being stored in tables, with each row in a table having the same format. Each row in a table summarizes some object or relationship in the world. Let us consider an enterprise which buys certain parts from certain suppliers at different times. Part of the world of this enterprise might be represented in the tables which appear in Figure A-1. Each supplier is represented by a row in the SUPPLIER table, each type of part by a row in the PARTS table and each shipment of one type of part by a supplier is represented by a row in the SHIPMENT table. Suppliers are summarized by their number, name and location. Parts are summarized by their number, name and location of use. Shipments are summarized by their supplier, part and the quantity shipped. The kind of information that can appear in any one column is restricted. The SNAME column must contain the names of suppliers. The order of the columns in any one table is irrelevant to the information contained. Each row is unique in each table; no object (that is, supplier, part or shipment) is represented by more than one row.

The tables shown in Figure A-1 are examples of relations. The column names of each table provide the format for that table. The column names are called attribute names. Each attribute name has a corresponding domain of permissable values for the associated column. The domain of QTY might be the set of positive integers up to 1,000,000. The rows in the relations are called tuples. The tuples in each relation form a set, that is, there are no duplicate tuples. There is a subset of attribute names in each relation with the property that the tuples in that relation can be distinguished by looking at the attribute values corresponding to the names in that subset. This subset is called the key for the corresponding relation. S# is the key for the SUPPLIER relation, S# and P# together form the key for the SHIPMENT relation.

SUPPLIER

S#	SNAME	CITY
s1	ajax	london
s2	acme	paris
s3	aone	rome
s4	best	paris

(a)

PART

P#	PNAME	CITY
p1	nut	london
p2	bolt	madrid
p3	screw	milan

(b)

SHIPMENT

S#	P#	QTY
s1	p1	300
s2	p1	100
s2	p3	200
s3	p2	100
s4	p3	300

(c)

Figure A-1: The PARTSUPPLIER Database

A.2.2. Relations

A *relation scheme* R is a finite set of *attribute names* $\{A_1, A_2, \ldots, A_n\}$. Corresponding to each attribute name A_i is a set D_i, $1 \leqslant i \leqslant n$ called the *domain* of A_i, sometimes denoted $dom(A_i)$. Attribute names are sometimes called *attributes*. Let $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_n$. A *relation* r on relation scheme R is a finite set of mappings $\{t_1, t_2, \ldots, t_p\}$ from R to $\mathbf{D}$ with the restriction that each mapping $t \in r$, $t(A_i)$ must be in D_i, $1 \leqslant i \leqslant n$. These mappings are called tuples.

In Figure A-1 there are three relation schemes.

SUPPLIER = {S#, SNAME, CITY}
PART ={P#, PNAME, CITY}
SHIPMENT = {S#, P#, QTY}

Some of the domains might be:

dom(SNAME) = {ajax, acme, aone, best, num1, quality, cutrate}
dom(P#) = {p1, p2, p3, p4, p5, p6, p8, p9}
dom(QTY) = the set of all positive integers less than 1000000

The SHIPMENT relation has 5 tuples. One of them is t defined as t(S#) = s1, t(P#) = p1, t(QTY) = 300.

The tuples are defined as mappings in order to avoid any commitment to the order of the attributes in the relations. However, for convenience we will denote relations by listing attributes in a certain order and tuples by listing values in the same order.

The *value* of tuple t on attribute A is called the A-*value* of t. If t is interpreted as a mapping the A-value of t is $t(A)$. If t is interpreted as a row in a table, the A-value of t is the entry of t in the column headed by A. For any subset X of R, we can restrict t to those attribute values corresponding to X; the restriction is denoted $t(X)$ and called the X-value of t.

Let t be the tuple defined in the preceding example. The S#-value of t is t(S#) = s1. Let X = {P#, QTY}. The X-value of t is {p1, 300}.

A.2.3. Keys

A *key* of relation r on relation scheme R is a subset $K = \{B_1, B_2, ..., B_m\}$ of R with the property that no two tuples have the same values on all the attributes of K; that is, $t_1(K) \neq t_2(K)$, where t_1 and t_2 are distinct tuples in r. Therefore, it is sufficient to know the K-value of a tuple to identify the tuple uniquely.

Let us formulate some notation for relations, schemes and keys. We will use uppercase letters from the beginning of the alphabet for attributes, uppercase letters from the back of the alphabet for relation schemes, and lowercase letters for relations. A relation scheme $R = \{A_1, A_2, ..., A_n\}$ may be denoted by $R[A_1A_2...A_n]$, or sometimes $[A_1A_2...A_n]$ when the name is unimportant. (Concatenation is used to stand for set union between of sets of attributes.) A relation r is written $r(R)$ or $r(A_1A_2...A_n)$. If a relation has more than one key the various keys are referred to as *candidate keys* and one of the candidate keys is designated to be the *primary key*. To denote the primary key, (hereafter referred to as the key if there is no need to distinguish it from other candidates), of a relation or relation scheme we underline the attributes in the key.

Rewriting the relation schemes for Figure A-1 we have:

SUPPLIER = {S#, SNAME, CITY}
PART ={P#, PNAME, CITY}
SHIPMENT = {S#, P#, QTY}

The definition of key has to be narrowed somewhat to include a concept of minimality. A *key* of a relation *r(R)* is a subset K of R such that for any distinct tuples t_1 and t_2 in r, $t_1(K) \neq t_2(K)$ and no proper subset K' of K shares this property. K is a *superkey* of r if K contains a key of r.

A.2.4. Operations

Two relations on the same scheme can be considered sets over the same universe. Thus, Boolean operations can be applied to two such relations. The set $r \cap s$ is the relation *q(R)* containing all tuples that are in both s and r. The set $r \cup s$ is the relation *q(R)* containing all tuples that are in either s and r. The set $r - s$ is the relation *q(R)* containing those tuples that are in r but not in s.

Select is a unary operation on relations. When applied to a relation r, it yields another relation that is the subset of tuples in r with a certain value on a specified attribute. Let r be a relation on scheme R, A an attribute in R, and a an element of *dom(A)*. Using mapping notation, $\sigma_{A=a}(r)$ ("select A equal to a on r") is the relation $r'(R) = \{t \in r | t(A) = a\}$.

The relation in Figure A-2.b is $\sigma_{\text{CITY}=\text{"paris"}}(supplier)$, where *supplier* is the relation in Figure A-1.a.

Project is also a unary operator on relations. The *projection of* r *onto* X, written $\pi_X(r)$, is the relation $r'(X)$ obtained by striking out the columns corresponding to attributes in $R-X$ and removing duplicate tuples in what remains. In mapping notation, $\pi_X(r)$ is the relation $r'(X) = \{t(X) | t \in r\}$.

The projection of the *part* relation from Figure A-1.b onto the attributes P# and PNAME, $\pi_{\{P\#,PNAME\}}(part)$, is shown in Figure A-2.a.

p1	nut
p2	bolt
p3	screw

(a)

s2	acme	paris
s4	best	paris

(b)

s1	ajax	london	p1	nut

(c)

Figure A-2: Examples of Database Operations

Join is a binary operator for combining two relations. In general, join combines two relations on all their common attributes. Start with relations $r(R)$ and $s(S)$ with $RS = T$. The *join* of r and s, written $r \bowtie s$, is the relation $q(T)$ of all tuples t over T such that there are tuples $t_r \in r$ and $t_s \in s$ with $t_r = t(R)$ and $t_s = t(S)$. Thus, q is made up of all possible tuples that can be derived by combining the tuples of r and the tuples of s. Since $R \cap S$ is a subset of both R and S, as a consequence of the definition, $t_r(R \cap S) = t_s(R \cap S)$. Thus, every tuple in q is a combination of a tuple from r and a tuple from s with equal $(R \cap S)$-values. Join is sometimes referred to as *natural join* to distinguish it from some other types of join operations that will be briefly described below.

The join of the *supplier* and *part* relations of Figure A-1, *supplier*$\bowtie$*part*, is shown in Figure A-2.c. *SUPPLIER*$\cap$*PART* = CITY, and the first tuples from each of the two corresponding relations are the only pair which have equal CITY-values. Thus, the only tuple in the join is the one created by the combination of these two tuples.

The resulting relation represents a relationship between a supplier and a part, that is, the supplier is located in the same city in which the part is used.

It can be seen that the join operation is commutative from the symmetry in its definition. It is also associative. Given relations q, r and s,

$$(q \bowtie r) \bowtie s = q \bowtie (s \bowtie r).$$

The *Θ-join* of r and s is the relation which may be formed by the selection of those tuples in the cartesian product of r and s such that the A-value of R stands in relation Θ to the B-value of S. A Θ-join where Θ is '=' is called an *equijoin*. An equijoin where $A = B$ is the same as the join of r and s.

A.2.5. Functional Dependencies

Two primary purposes of databases are to avoid data redundancy and to improve data reliability. Any *a priori* knowledge of restrictions or constraints on permissible sets of data is useful when designing databases to achieve these purposes. The knowledge of what is possible in the world allows designers to impose restrictions on the data which in turn impose structure on the database. Data dependencies are formulations of such advance knowledge. Dependencies constrain relations in such a way that they reflect only those states of affairs in the real world that are possible.

For example, in the world that is partially represented by the UNIVERSITY database described in Section 2.1.4, there are a number facts about the entities and relationships in that world that impose constraints on the database. For example,

Classes are offerings of exactly one course.
Courses are offered by only one department.
For each class, students receive only one grade.

These restrictions are examples of *functional dependencies*. The functional dependency is the most important of the data dependencies.[10] Informally, a functional dependency occurs when the values of a tuple on one set of attributes uniquely determine the values on another set of attributes. The above restrictions can be phrased as

CNAME functionally depends on CLID,
DEPT functionally depends on CNAME, and
GRADE functionally depends on {CLID, STUD#},

where CLID is the class identifier attribute and CNAME is the course name attribute. Generally the order of the sets is reversed and we write CLID, STUD# *functionally determines* GRADE, or {CLID, STUD#} $\rightarrow$ GRADE.

We now state this notion formally. Let r be a relation on scheme R, where X and Y are subsets of R. Relation r *satisfies* the *functional dependency* (FD) $X \rightarrow Y$ if for every X-value x, $\pi_Y(\sigma_{X=x}(r))$ has at most one tuple. In other words, for two tuples t_1 and t_2 in r, if $t_1(X) = t_2(X)$, then $t_1(Y) = t_2(Y)$.

Given a relation scheme R composed of the set of attributes S and containing the set of candidate keys $\mathbf{K}$, we say R *embodies* the FD $K \rightarrow S$ if $K \in \mathbf{K}$.

Functional dependence is transitive. For example, since CLID $\rightarrow$ CNAME and

[10]Other dependencies identified by DB theorists are multivalued dependencies (MVDs) and join dependencies (JDs)

CNAME $\rightarrow$ DEPT, CLID $\rightarrow$ DEPT; i.e., CLID $\rightarrow$ CNAME $\rightarrow$ DEPT. We say that the set of dependencies {CLID $\rightarrow$ CNAME, CNAME $\rightarrow$ DEPT} *logically implies* the dependency CLID $\rightarrow$ DEPT. In general, if F is a set of dependencies for a relation scheme R and $X \rightarrow Y$ is a functional dependency we say F *logically implies* $X \rightarrow Y$, if every relation r on R that satisfies F also satisfies $X \rightarrow Y$. The *closure* of F, written F^+, is the set of FD's that are logically implied by F. There is a sound and complete set of axioms, known as *Armstrong's axioms*, which derive logical implications from sets of FD's (Armstrong, 1974). We rephrase and repeat them here. If W, X, Y, and Z are subsets of R, for any relation r on R:

F1. Reflexity: $X \rightarrow X$.
F2. Augmentation: $X \rightarrow Y$ implies $XZ \rightarrow Y$.
F3. Pseudotransitivity: $X \rightarrow Y$ and $YZ \rightarrow W$ implies $XZ \rightarrow W$.

There is a linear time algorithm due to (Beeri and Bernstein, 1979) to test whether a FD is a member of F^+ for a given set of FD's F.

A.2.6. Multivalued Dependencies

Suppose we are given a relation scheme R and A and B are subsets of R. A *multidetermines* B, $A \twoheadrightarrow B$, if, given values for the attributes of A, there is a set of zero or more associated values for the attributes of B, and this set of B-values is not connected in any way to values of the attributes in $R-A-B$. Formally, we say $A \twoheadrightarrow B$ holds in R if whenever r is a relation for R, and t and s are two tuples in r, with $t(A) = s(A)$, then r also contains tuples u and v, where

1. $u(A) = v(A) = t(A) = s(A)$, and
2. $u(B) = t(B) = u(R-A-B) = s(R-A-B)$.

The relation in Figure A-3 contains two multivalued dependencies, COURSE$\twoheadrightarrow$PROF and COURSE$\twoheadrightarrow$TEXT.

COURSE	PROF	TEXT
math 100	Smith	Calculus
math 100	Smith	Calculus1
math 100	Jones	Calculus
math 100	Jones	Calculus1

Figure A-3: Example of Multivalued Dependency

A.2.7. Normal Forms

A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. A relation scheme R is in *first normal form* (1NF) if the values in $dom(A)$ are atomic for every attribute A in R. That is, the values in each domain are not lists, sets of values, or composite values. A relation scheme R is in *second normal form* (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key. (Attribute A is *fully dependent* on the set of attributes B if it is functionally dependent on B and not functionally dependent on any proper subset of B.) A relation scheme R is in *third normal form* (3NF) if and only if it is in 2NF and every non-key attribute is nontransitively dependent on the primary key. A relation scheme R is in *Boyce/Codd Normal Form* (BCNF) if and only if every determinant is a candidate key. (A *determinant* is an attribute on which some other attribute is fully dependent). BCNF is a stronger version of 3NF devised to handle the case of a relation possessing two or more composite and overlapping candidate keys. Sometimes relations can be in BCNF and still be redundant. For example, the relation in Figure A-3 is in BCNF, (the key consists of all three attributes). The redundancy results from the multivalued dependencies COURSE$\twoheadrightarrow$PROF and COURSE$\twoheadrightarrow$TEXT. A relation R is in *fourth normal form* (4NF) if and only if, the only dependencies (FDs or MVDs) in R are functional dependencies

COURSE	PROF
math 100	Smith
math 100	Jones

COURSE	TEXT
math 100	Calculus
math 100	Calculus1

Figure A-4: Examples of 4NF Relations

from a candidate key to some other attribute. The relations in Figure A-4 contain the same information as the one in Figure A-3 and are in fourth normal form.

References

Armstrong, W.W. *Dependency Structures of Data Base Relationships*, pages 580-583. Proceedings 1974 IFIP Congress, Amsterdam, 1974.

Atzeni, P. and Parker, D.S. *Assumptions in Relational Database Theory*, pages 1-9. Proceedings of the ACM Symposium on Principles of Database Systems, Los Angeles, CA., 1982.

Ballard, Bruce W., Lusth, John C. and Tinkham, Nancy L. LDC-1: A Transportable, Knowledge-Based Natural Language Processor for Office Environments. *ACM Transactions on Office Information Systems*, 1985, *3(2)*, 1-25.

Barr, Avron & Edward A. Feigenbaum. *The Handbook of Artificial Intelligence, Volume 1*. Los Altos, California:William Kaufmann, Inc., 1981.

Beeri, C., and Bernstein, P.A. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Transactions on Databases*, 1979, *4(1)*, 30-59.

Biller, H. On the Notion of Irreducible Relations. In G. Bracchi and G.M. Nijssen (Eds.), *Data Base Architecture*, Amsterdam: North-Holland Pub. Co., 1979.

Carlson, C.R. & R.S. Kaplan. *A Generalized Access Path Model and its Application to a Relational Data Base System*, pages 143-154. ACM SIGMOD, Washington, D.C., 1976.

Codd, E.F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 1970, *13(6)*, 377-387.

Codd, E.F. Relational Completeness of Data Base Sublanguages. In R. Rustin (Ed.), *Database Systems*, Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Codd, E.F. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 1979, *4(4)*, 397-434.

Damerau, Fred J. Operating Statistics for the Transformational Question Answering System. *American Journal of Computational Linguistics*, 1981, *7(1)*, 30-42.

Damerau, F.J. Problems and Some Solutions in Customization of Natural Language Data Base Front Ends. *ACM Transactions on Office Information Systems*, 1985, *3(2)*, 165-184.

Date, C.J. *An Introduction to Database Systems, Third Edition.* Reading, Mass.:Addison-Wesley, 1981.

Hafner, Carole D. and Godden, Kurt. Portablility of Syntax and Semantics in Datalog. *ACM Transactions on Office Information Systems*, 1985, *3(2)*, 141-164.

Johnson, D.E. *Design of a Portable Natural Language Interface Grammar.* Technical Report 10767, IBM Thomas J. Watson Research Laboratory, 1984.

Karp, R.M. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher (Eds.), *Complexity of Computer Computations*, New York: Plenum Press, 1972.

Kent, W. Consequences of Assuming a Universal Relation. *ACM Transactions on Database Systems*, 1981, *6(4)*, 539-556.

Liestman, Arthur L., and Richards, Dana. A linear algorithm for small instances of the Steiner tree problem in graphs. To be published. 1986.

Maier, David. *The Theory of Relational Databases.* Rockville, Maryland:Computer Science Press, 1983.

Martin, P., Appelt, D., Grosz, B. and Periera, F. An Experimental Transportable Natural Language Interface. *Database Engineering*, 1985, *8(3)*, 10-22.

Osborn, S.L. *Towards a Universal Relation Interface*, pages 52-60. Proceedings of the Fifth International Conference on Very Large Data Bases, Rio de Janeiro, Brazil, 1979.

Perrault, C. Raymond and Cohen, Philip R. It's for your own good: a note on inaccurate reference. In Joshi, Aravind K., Webber, Bonnie L. and Sag, Ivan A. (Eds.), *Elements of Discourse Understanding*, Cambridge: Cambridge University Press, 1981.

Petrick. S.R. *Semantic Interpretation in the Request System.* Technical Report RC 4457, IBM Thomas J. Watson Research Laboratory, 1973.

Petrick. S.R. *Natural Language Database Query Systems.* Technical Report RC 10508, IBM Thomas J. Watson Research Laboratory, 1984.

Petrick, S.R. Personal communication. 1986.

Pylyshyn, Z. and Kittredge, R. Databases and Natural Language Processing. *Database Engineering*, 1985, *8(3)*, 2-9.

Sagiv, Yehoshua. *Can We Use The Universal Instance Assumption Without Using Nulls?*, pages 108-120. ACM SIGMOD, Univ. of Michigan, Ann Arbour, 1981.

Searle, John R. *Speech Acts An Essay in the Philosophy of Language.* Cambridge:Cambridge University Press, 1969.

Smith, H.C. Composing Fully Normalized Tables From a Rigourous Dependency Diagram. *CACM*, 1985, *28(8)*, 826-838.

Thompson, Bozena Henisz and Thompson, Frederick B. ASK Is Transportable in Half a Dozen Ways. *ACM Transactions on Office Information Systems*, 1985, *3(2)*, 185-203.

Ullman, Jeffrey D. *Principles of Database Systems, 2nd ed.* Rockville, Maryland:Computer Science Press, 1982.

Ullman, Jeffrey D. *The U. R. Strikes Back*, pages 10-22. Proceedings of the ACM Symposium on Principles of Database Systems, Los Angeles, CA., 1982.

Wald, J.A. and Sorenson, P.G. Resolving the Query Inference Problem Using Steiner Trees. *ACM-TODS*, 1984, *9(3)*, 348-368.

Zhang, Z-Q, & A.O. Mendelzon. A Graphical Query Language for Entity-Relationship Databases. In C.G. Davis, S. Jajodia, P.A. Ng & R.T. Yeh (Eds.), *Entity Relationship Approach to Software Engineering*, Amsterdam: North-Holland, 1983.