Elimination of Wasteful Operations in Natural Language Accesses to Relational Databases, Using a Knowledge-Based Subsystem

by

Stefan W. Joseph

Diplom Ingenieur Elekrotechnik, Technische Universitat Berlin

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
April 1988

Approval

Name:	Stefan W. Joseph
Degree:	Master of Science
Title of Thesis:	Elimination of Wasteful Operations in Natural Language Access to Relational Databases, Using a Knowledge-Based Subsystem

Jia-Wei Han
Chairman

Nick Cercone
Senior Supervisor

Romas Aleliunas

Robert F. Hadley

Wo-Shun Luk
External Examiner

April 15, 1988
Date Approved

Title of Thesis/~~Project/Extended Essay~~

Elimination of Wasteful Operations in Natural Language Accesses to Relational Databases, Using a Knowledge-Based Subsystem

Author: [illegible]
(signature)

STEFAN W. JOSEPH
(name)

April 22. 1988
(date)

Acknowledgements

It is a pleasure to acknowledge the help and support I have received from many people during the efforts that have culminated in this thesis.

Nick Cercone is responsible for creating the Automated Academic Advisor project within which I have carried out this research. I am most grateful to Nick for his patience with me, for carefully reading earlier drafts of my thesis, for his valuable advice, and his financial, and moral support throughout my time at Simon Fraser University.

I am very grateful to Romas Aleliunas with whom I spent numerous hours discussing almost all aspects of my thesis. Romas carefully read earlier drafts, and his constructive criticism and his uncompromising demand for clarity have greatly helped to improve my thesis.

I am very grateful to Bob Hadley, who also carefully read earlier drafts of my thesis and gave me valuable comments on several aspects, especially on the four valued 'logic'.

Tomek Strzalkowski carefully read an early draft of my thesis and sent me valuable comments on it. Jia-Wei Han and Wo-Shun Luk carefully read my thesis and also gave me some valuable comments.

I had long discussions about my work with many of the graduate students of the School of Computing Science, mostly with Mimi Kao who did her Master's degree in a directly related area within the AAA project, and with Sanjeev Mahajan. All those discussions contributed to clarifying different aspects of my thesis.

I am also grateful to the people taking care of the computing facilities at the School of Computing Science, especially to Steve Cumming for his help whenever it was needed. Keith Vincent created some of the special characters which were not available in the normal version of scribe.

Abstract

We present an approach to eliminate wasteful operations in natural language accesses to relational databases, using a knowledge-based subsystem. The wasteful operations to be eliminated are mainly those that would be carried out if no special actions were taken, in case null events of type '*property inapplicable*' that is, type mismatches, were bound to occur.

We first design a database with an academic environment as domain. The design is based on the extended relational model RM/T for which we present a diagramming technique in Appendix A.. The RM/T model is introduced and slightly modified and further extended to our own version: RM/T*. The modifications are of a general nature, and not specific to the particular application. The extensions primarily affect the catalog, which is the intensional part of the model, they are particularly important for an efficient treatment of type mismatches.

We identify and classify the different types of null events according to the stage of query processing at which they can be detected with and without the assistance of a knowledge-based subsystem. We clarify the fundamental distinction between these two types of null events, which has been overlooked in the current literature.

We then outline an efficient way of processing the queries such that null events of type '*property inapplicable*' can be detected with only limited database access, or without any database access.

Finally we extend Codd's notion of a three valued logic to deal with null events of type '*value at present unknown*' to a fourth value in order to include the null events of type '*property inapplicable*'. Our approach is distinct from Codd's recent approach [Codd 87] to a four-valued logic.

Throughout the thesis we compare our work to previous work that has been done in the area, and propose some further extensions as appropriate.

Wie nur dem Kopf nicht alle Hoffnung schwindet,
Der immerfort an schalem Zeuge klebt,
Mit gier' ger Hand nach Schaetzen graebt,
Und froh ist, wenn er Regenwuermer findet !

(Faust, NACHT;
Johann Wolfgang Goethe)

How can such hope still dwell with him,
whose mind tenaciously adheres to rubbish,
who digs with eager hands for treasure
and is delighted when he finds a worm !

(Faust, NIGHT;
Johann Wolfgang Goethe)

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The use of natural language as a means of communication between a database system and its human users has become increasingly important since database systems have become widespread and their accessibility to nonexpert users is desirable, if not essential, to facilitate full use of the database system. Although natural language tends to be ambiguous and/or unspecific in a number of situations, [1] it can be seen as the ideal language for the communication between the database system and its human users since it is available to virtually every human. The user's effort to learn a formal query language is thus converted into the effort it takes to make the system accept and generate natural language sentences.

According to the *natural language front-end* paradigm, [Cercone and McCalla 86], the natural language access to a database system can be broken down into two major parts, the linguistic component and the database component. The database component performs the part corresponding to the traditional database management system, whereas the linguistic component is responsible for translating the natural language input into a formal query and generating a natural language response based on the results from the database search. Thus we have the structure depicted in Figure 1-1, where the linguistic component is represented by the solid vertical paths and the database component by the solid horizontal path. The lexicon is basically a table which is used to map the words of the natural language input onto the formal objects (relation names, attribute names, ...) of the database. Both the parser and the semantic interpreter make use of the lexicon. The natural language generator takes the formal response as its input, and also inspects the parse tree in order to generate an adequate natural language response.

More recent research [Winograd 83], [Cercone and McCalla 86] suggests we abandon this decomposed approach to a certain extent in favor of an integrated understanding system.

[1] for example, the question '*Does John or Mary have a phone ?*' could be interpreted as expecting a '*Yes/No*' answer or one or more numbers as a response

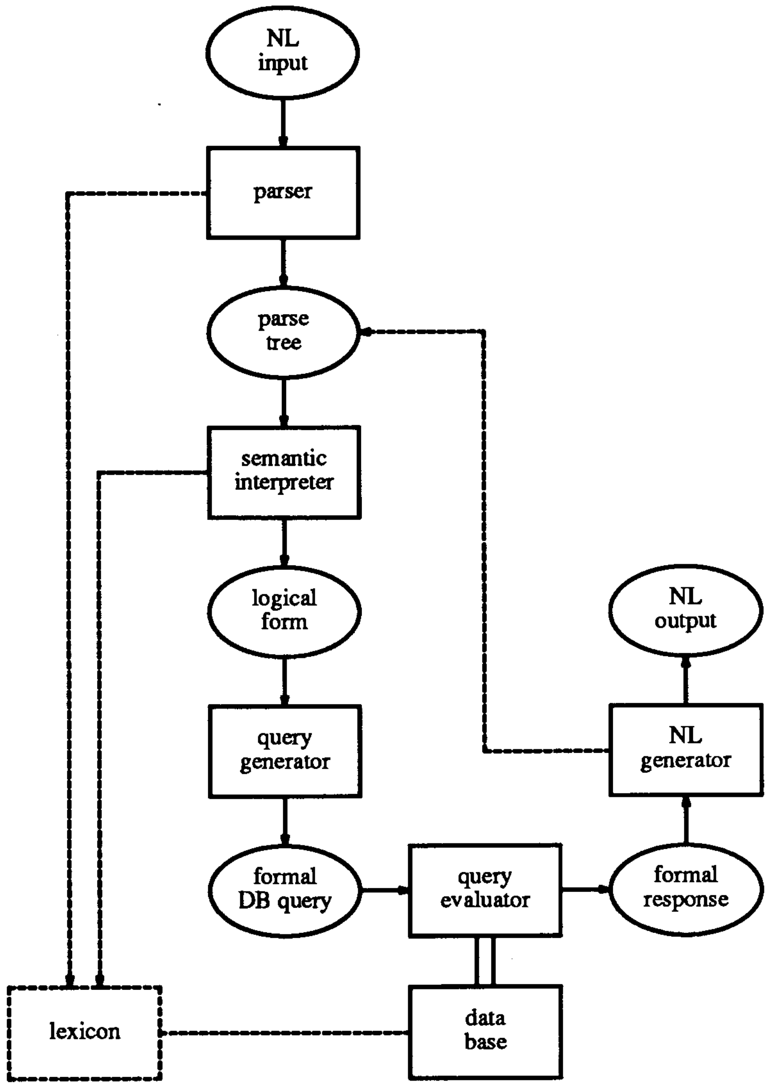


Figure 1-1: Traditional natural language — database system.

The main reason for this trend is based on the fact that *knowledge* is essential in order to allow the system to accept natural language input and to generate natural language output. Knowledge of the discourse issues and the surrounding context, for example, [Grosz 77], as well as of the domain and structure of the database must be included in the system in order to allow it to properly interpret and possibly disambiguate the queries.

Somewhat similar to the way humans make use of their knowledge about syntactic rules of the language spoken and about the context of discourse when trying to decipher distorted radio transmissions, natural language database systems could make use of syntactic knowledge and of knowledge about the actual database in order to properly relate the natural language input to the structure and contents of that database. Of course, the system will expect the user to ask questions pertaining to the domain of the database, which, in turn, represents some aspect of the real world. The syntactic knowledge usually resides in the linguistic component of the system, in particular in the syntax analyzer whereas knowledge about the actual database resides to some extent in the semantic data model used.

Knowledge about the user and the goals of his *speech acts* (see [Martinich 85],[2] [Winograd and Flores 86][3]) are especially important if a user friendly dialogue is to be carried through and Grice's cooperative principles of conversation (see [Martinich 85]) are to be followed.

A part of the knowledge that has to be implemented in order to improve the overall performance of the system might be of direct interest to the user. For example, knowledge about the structure and the domain of the database and knowledge about the rules in the knowledge-base could sometimes help the user to use the system more effectively. Therefore it makes sense to let the user have access to at least a part of the knowledge that is included in the system.

The next step towards an integrated understanding system is to include additional knowledge to represent aspects of the real world that could not be captured in an acceptable way by the traditional database system. One such aspect could be the representation of rules, for example, the rule that in the School of Computing Science a specific course, say CMPT 810, is always taught by either Dr. Hell, Dr. Liestman, or Dr. Peters and offered at least once every year.

Cercone and McCalla [Cercone and McCalla 86] have identified six issues that are particularly important to achieve an integrated understanding system. They are:

1. The full complexity of English is overwhelming, which means that the kind of language used when interfacing with a database are usually constrained; ways must be found of *expanding the linguistic coverage* of natural language systems.
2. The "stratified approach" of doing syntactic analysis, then semantic interpretation, then query evaluation, is ineffective; more *flexible parsing strategies* must be created, in particular techniques to integrate syntax, semantics, and pragmatics so that whatever action is appropriate at a given time can be done.
3. The separation of the linguistic component from the database components sets up an arbitrary barrier which may have become counterproductive; a means of reintegrating data and language, and ultimately of *integrating knowledge and language* must be found.

[2] Part 2, containing six articles by Austin, Grice, Searle, and Vendler, all related to speech acts

[3] Chapter 5: Language, Listening, and Commitment

4. Traditional (relational) database structures are not necessarily conductive to promoting the kinds of inferences which need to be made for the query to be comprehended or answered properly; more sophisticated structures i.e. appropriate *knowledge representation schemes* must be devised.
5. The user's understanding of the capabilities of the complete system and the current level to which he is or thinks to be informed is an important aspect of the man-machine communication which must be taken into account; *modeling the user* is important and cannot be ignored.
6. Even in a restricted linguistic domain such as natural language database interfacing, many *discourse phenomena* arise which must be accounted for if the natural language system is to behave cooperatively.

This thesis focuses on one aspect of such an integrated understanding system for which the implementation of additional knowledge is required, namely the elimination of wasteful operations that may occur during query processing, and in particular the treatment of *null events* and directly related issues.

We distinguish between *null events*, *null values* and *null responses* as follows:

- A *null value* is an actual value occupying some memory space, that would otherwise be used for a genuine value. The null value represents the fact that some information is missing in the field, where the null value is found.
- A *null response* (or empty response) is one that contains no information.
- A *null event* is an event that may happen sometime during query processing and that results in the generation of a null response unless special actions are taken.

Given a query to a particular database, there are different possible reasons which explain why a null event may occur. One possible reason could be the fact that a null value was found during query processing, that is, the information stored was incomplete with respect to the query at the time the query was entered to the system. For example, the phone numbers of several persons might have been missing although the rest of their personal data such as their names and addresses were present. Some of these persons whose phone numbers were missing, that is, substituted by null values in the corresponding attribute field in the database, might have a phone in real life, others might not have one.

Another possible reason could be a misconception on the part of the database user. Contrary to the user's expectations (reflected in the way he words his query), the requested information might be represented in a different way, or not at all. In this case a query would result in a null event although no actual null value is detected in the database, rather the structure of the database does not allow the interrelation of entities and properties as suggested by the wording of the query. In the above example, this type of null event would be the case if the database designer had chosen to include information about phone numbers for some of the persons represented in the database (say STAFF members), but not for others (say STUDENTS), or if he had chosen to exclude information about phone numbers altogether.

Knowledge, in addition to the extensional data in the database, can help to detect null events and generate appropriate responses to the user. This additional knowledge can be knowledge about the structure of the database, or about the real world. The part that deals with knowledge about the database structure alone, is probably easier to adapt to different systems, than the part that deals with real world knowledge about the aspect of the world being represented by the system.

In our approach to the problem of null events and directly related issues, the complete natural language database system still adheres to the stratified structure as illustrated in Figure 1-1. However a knowledge-based subsystem interacts with different parts of this system and supervises the different processing stages, in particular the query generation and the query interpretation stages. Thus, this approach represents a step towards an integrated system, while leaving the clearly decomposable parts basically unaltered as separate modules. Figure 1-2 shows the general structure of our approach. Here the dashed arrows originating from the knowledge-base indicate the interactions of the knowledge-based subsystem as proposed in this thesis. The dotted arrow to the semantic interpreter module indicates a possible assistance of the knowledge-based subsystem to disambiguate some natural language input with respect to the database at hand. The dotted arrow to the natural language generator indicates a control function of the knowledge-based subsystem in case it can run in parallel with the rest of the system. In this case it could sometimes be necessary to abort the natural language generator in its current processing in order to process an alternative formal response, which turns out to be more appropriate.

The underlying database is structured according to the RM/T model [Codd 79] and represents a part of the domain of an academic advisory environment. A knowledge-based subsystem will exploit an extended version of the *catalog* to predict the occurrence of null events and thus to avoid wasteful searches through the database. The same information will be used to generate answers to help the user understand the reasons why a search was wasteful.

The less a user knows about the database domain and logical data independence, the higher the chances of null events and inputs that cannot be properly processed by the system. An unsophisticated database system might

- not be able to avoid wasteful searches,
- simply reply with a null response when a null value is encountered, leaving the correct interpretation up to the user.
- prevent misleading responses when null events occur due to user misconceptions, by rejecting input that could somehow be determined to be illegal without educating the user about the reasons for doing so,

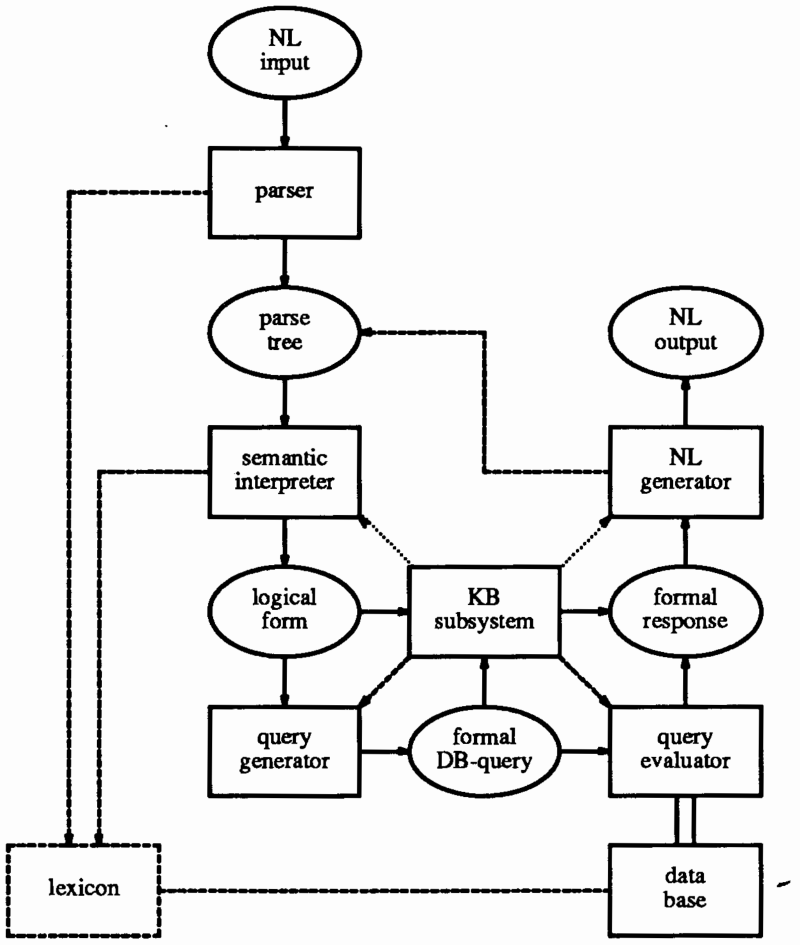


Figure 1-2: A natural language interface to a database, supported by a knowledge-based subsystem.

The problem of turning null responses into cooperative responses has been addressed extensively in [Kaplan 79] and [Kao 86]. Here null responses obtained from the database after an exhaustive search through the extensional data are turned into quality responses in order to improve the communication between the database system and the human user.

The objective of this research is to prevent the exhaustive and time consuming searches and to give a direct but helpful answer to the user. We cannot deal with all null events in this manner. Some null events occur because of a lack of extensional data and can be detected only after an exhaustive search has been carried out. In these cases, approaches such as Kao's will continue to be useful to improve the overall system performance. Our objective then is to identify those null events that can be detected with only limited database access or without any database access, and to find an efficient way to perform this detection. For the analysis part of the problem and the identification of the predictable null events, we design a database wich is based on a well recognized semantic model.

Since a natural language system is most probably used in an environment where the percentage of database expert users is small, a high number of queries has to be expected that could result in wasteful searches and null events if no special precautions are being taken. In general the search through the database is the most time consuming part of the system, even if a sophisticated natural language interface is included. For example in the CO-OP system [Kaplan 79], a natural language database system that provides cooperative responses to simple questions requesting data retrieval, about 90 % of the real time required to get a response is spent in the database system itself, not in the natural language components. Avoiding dispensable searches through the database would mean a valuable improvement in performance, especially in systems, where the domains are sufficiently complex to let the users of the natural language facility make incorrect presumptions in their queries.

A higher system acceptance of the users can be expected for two reasons:

1. because of the higher efficiency of processing the queries (the user has to wait less long for responses) and
2. because of the higher quality of the responses (an explanation on any misconceptions on the side of the user is better than a null response).

The quality of our approach is arguable with respect to point 1 since the effects of the inclusion of the knowledge-based subsystem will be twofold: the response time to queries that would otherwise result in null responses will decrease, but the response time to the other queries will increase. For the two goals, optimization of response time, and expansion of linguistic coverage, priority should be given to an expansion of the linguistic coverage as long as queries that a human user sees as pertaining to the domain of the database, might result in null responses. With our approach we attempt to expand the linguistic coverage with a minimum increase in response time.

This short introduction into the general area of research in which the thesis is embedded, the development of integrated knowledge-based systems, serves to orient the reader. We now present a brief overview of the following Chapters.

The example database referred to throughout this thesis is introduced in Chapter 2. The domain of this database covers a representative part of an academic environment. The structure of the database follows the RM/T* model, which is basically the extended RM/T model with a number of yet further extensions and constraints. The description of the database goes hand in hand with a general introduction to the RM/T model and the changes and additions made in the RM/T* model. The relevant concepts are introduced step by step and as they are required in the database.

A detailed description of the catalog of the database is introduced in Chapter 3. The catalog is actually a part of the RM/T database, but clearly separable from the extensional data; it represents the intensional aspect of the world as represented by the database. Since the information in the catalog is time independent, both, the catalog structure and its complete contents are presented. The catalog of the RM/T* model has some further extensions concerning integrity constraints and representing further information, especially about the domains of the attributes.

In Chapter 4 we first clarify the distinction between null values and null events. Based on this distinction, we show the fundamental distinction between null events of type 'value at present unknown' and of type 'property inapplicable'. This distinction has been overlooked in the current literature, and has significant influence on database design. Finally we present a classification of null events, which goes hand in hand with an identification of those null events that can be detected with limited database access, or no database access at all.

In Chapter 5 we present the relevant aspects required to process the queries in such a way, that null events can be detected with as little database access as possible: We outline algorithms used to inspect each particular query and retrieve applicable information from the catalog before accessing the actual database. With these algorithms wasteful operations can be eliminated which would otherwise be carried out and result in null events of type 'property inapplicable'.

In Chapter 6 we present a digression on Codd's notion of a three-valued logic to deal with null events of type 'value at present unknown' to a fourth value in order to include null events of type 'property inapplicable'. The elements of the four-valued logic reflect the pragmatic nature of the approach proposed in this thesis.

A summary of what has been achieved is given in Chapter 7. Open issues as well as directions for further, future work are presented. We finally suggest some further extensions to the RM/T* model, which allow the representation of partial information. This approach can be seen as a realization of a pseudo-indexed representation of null values of type 'value at present unknown': only those null values are indexed, for which some further information is available.

In Appendix A we present a diagraming technique for the RM/T (and RM/T *) model. Based on this diagraming technique, we present a diagram of the logical structure of our example database. Frequent reference to this diagram might prove helpful while reading most parts of the thesis.

In Appendix B we present all relations of the example database, including some explanation on their intended meaning. This appendix is contains information at a level of detail, that is only rarely required while reading the thesis and is mainly added for reasons of completeness.

In Appendix C we present the relations of the catalog, as well as the contents of those relations. The contents reflects the structure of the database.

In Appendix D we present the parts of the knowledge-base, which are not represented in the catalog.

In Appendix E we give a brief overview of some mathematical terms used in the description of the RM/T model.

Chapter 2
The RM/T* database

In order to show the interactions of the knowledge-based subsystem with a database, we use a database representing aspects of an academic environment as a practical example. This choice is mainly due to the fact that some work in this area, the AAA-project (AAA stands for automated academic advisor), is currently being pursued at Simon Fraser University [Cercone et al. 83], [Cercone et al. 84], [Hall 86], [Kao 86], [McFetridge et al. 88].

An inadequate database design could create problems which could then (partially) be solved by additional processing. Our intention is to show that even a good, or at least adequate, database design will leave some unsolved problems which can successfully and efficiently be handled using a knowledge-based subsystem. The design methodology for the database is based on the RM/T model [4] [Codd 79], [Date 83]. RM/T is based on the relational model, but imposes additional structure on the comparatively unstructured collection of information of a 'normal' relational database, and introduces some discipline into the integrity enforcement scheme. In [Date 86a] [5] C. J. Date sees a direct parallel

> "... to the basic relational model, which was used for logical database design long before any relational DBMS was ever available. Even if no RM/T system per se is ever developed, its use in design may nonetheless prove an important contribution."

In addition to providing a set of objects (entities of different types, properties, etc.) and rules (integrity constraints), RM/T also provides a set of high-level operators over and above the operators of the basic relational model. However, we use the RM/T model solely as a basis of the semantic data model of our database, and are not concerned with its manipulative aspect. Also note that in [Date 86b], when referring to the operational aspect of the RM/T model in his overview, Date points out that

> "much additional work remains to be done in this area".

An introduction to the relevant concepts of the RM/T model and a few elaborations on the design

[4] RM/T stands for Relational Model Tasmania: Codd presented this model for the first time during an invited talk presented at the Australian Computer Science Conference in Hobart, Tasmania, in February 1979.

[5] The Relational Future, page 489.

decisions concerning our database are given in this chapter. A detailed description of the RM/T model and the extended RM/T model can be found in [Codd 79] and [Date 83], respectively. We introduce a number of further extensions to the RM/T model and to the extended RM/T model and refer to our extended model as RM/T*. As an initial overview, a diagram showing the complete database is given in Appendix A. In Appendix B individual relations are specified in detail. The description of the catalog of the database will be given in Chapter 3.

2.1. The RM/T model

We give a brief overview of the RM/T model to orient the reader. The different RM/T concepts involved will all be explained in more detail in subsequent sections.

According to the RM/T model, a micro-world of interest is represented in terms of entities, their properties, and nothing else. Each entity can be arranged along two distinct dimensions: its type and the class it belongs to. The type gives some indication as to what type of real world object the entity corresponds to; the class indicates some of the integrity constraints that apply to the entity.

Real-world objects, relationships among them, as well as relationships among those relationships, are all represented as entities. Informally, we may say that an entity is any distinguishable object — where the 'object' in question may be as concrete or as abstract as we please. Internally, all entities are identified by system generated surrogates which are invisible to the user who uses his own key attributes to identify the tuples of interest.

References from an entity type A to an entity type B (A and B need not be distinct) are represented by a special attribute field in a property relation (introduced in Section 2.2.2) for entity type A, containing a surrogate identifying an entity of type B. Independent of the entity type of A, or B, all references are classified into a set of disjoint *classes* of references. Each class of reference is subject to a specific *integrity constraint*. Thus each entity is characterized by its type, the classes of references involved in the entity, and the set of single valued properties that apply to it.

2.2. The two kinds of relations in RM/T

Before describing the three classes of entities and their corresponding integrity constraints in detail, we introduce the two kinds of relations that may exist in an RM/T database, entity relations and property relations, and the two integrity constraints that hold for all entity relations and for all property relations, respectively.

2.2.1. Entity relations

Just as in the basic relational model, entities are categorized into different *entity types*. For example, we use the entity type STUDENT to represent students and their properties in our database.

In order to keep track of the instances of a particular entity type that are currently represented in the database, a specific kind of relation, an *E-relation*, is used. An E-relation is a unary relation, which is given the same name as the entity type it represents. For example, Figure 2-1 shows the entity relation for the entity type STUDENT. There is an E-relation for each entity type in the database. Its sole attribute is called an *E-attribute* and named by appending the character '¢' at the end of the relation name, for example, STUDENT¢.

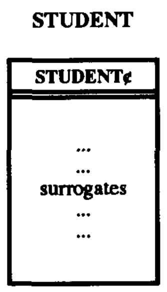


Figure 2-1: Entity relation for entity type STUDENT.

Every instance of an entity is uniquely identified by a system assigned *surrogate*. The surrogates are used for system internal identification only; they are invisible to the user. A special domain, the *E-domain*, serves as a source of all surrogates. The mapping from real-world entities to the surrogates in the corresponding E-relation is partial and bijective (see Appendix E). For example, for every student in the real-world there is at most one surrogate in the E-relation STUDENT, and if there is such a surrogate, then it uniquely identifies that student. Each E-relation lists the surrogates of all the instances of the corresponding entity type that are currently represented in the database. [6] As stated in [Codd 79], [Date 83] and [Date 86b], all E-relations are subject to the following integrity constraint:

- **Entity Integrity:** E-relations do not accept null values; E-relations accept insertions and deletions but not updates.

The first part of this rule conforms to the entity integrity rule of the basic relational model, stating that no primary key of a base relation is allowed to be null or have a null component. The second part is meant to conform with the ground rules for surrogates and is somewhat misleading. Since an E-relation is a unary relation, an update of any of its tuples is in effect identical to a deletion—insertion pair.

The system uses single E-attributes as primary keys for all relations (both E-relations and property relations), that is, internally all entities in the database are identified by their surrogates and by nothing else. The users may use their own (possibly composite) primary key attributes, [7] which will be treated as property attributes (see Section 2.2.2) by the system, but the users do not see the system internal E-attribute primary keys.

Every reference in RM/T is a reference from an E-attribute of some relation to the E-relation of some entity type.

2.2.2. Property relations

Each entity in RM/T may have a set of zero or more immediate, single-valued properties, represented by a corresponding set of attribute fields. For example, as shown in Figure 2-2, the three property types of a student, identification number, name of the student, and sex of the student, are represented by the property attributes NUMBER, NAME, and SEX, respectively, of the STUDENT entity.

The property attributes of a particular entity type are represented in a set of n-ary property relations (or P-relations) for that entity type. The set of P-relations for a particular entity type satisfies the following

[6] The RM/T model does not assume a *closed world*, that is, more real-world entities than the ones currently represented may exist.

[7] However, users need not invent artificial key attributes, in case no 'natural' key exists for a relation.

STUDENT

STUDENT¢
aadxg997
aadxg998
aadxg999
aadxh000
aadxh001
aadxh002

STUDENT_PROPERTIES

STUDENT¢	NUMBER	NAME	SEX
aadxg997	843901737	JONES	M
aadxg998	833904097	WALKER	M
aadxg999	871234567	SMITH	F
aadxh000	818767654	JONES	F
aadxh001	787656545	WHITE	M

Figure 2-2: Entity relation for entity type STUDENT and a corresponding property relation.

- **Naming Integrity:**
  - the primary key of every P-relation in the set is an E-attribute with the same name as the single E-attribute of the corresponding E-relation;
  - no two P-relations in the set have any attribute names in common except those E-attribute names mentioned above.

For example, as shown in Figure 2-2, the set of P-relations for entity type STUDENT consists of the single element STUDENT_PROPERTIES, whose key attribute is the E-attribute STUDENT¢, and whose remaining attributes, NUMBER, NAME, and SEX represent immediate, single-valued properties of a STUDENT entity [8]

A property reference is a reference from the E-attribute primary key of a P-relation to the corresponding E-relation. This reference is a total and injective function: the domain from which the values in the key attribute field of a P-relation are taken, is a subset[9] of surrogate values represented in the corresponding E-relation. Expressed in the terminology of database theory the integrity constraint corresponding to the property references is the following

- **Property Integrity:** No tuple can exist in a P-relation, unless the primary key value of that tuple is identical to some value in the E-relation corresponding to that P-relation.

For example, no tuple can exist in the P-relation STUDENT_PROPERTIES unless its key attribute STUDENT¢ contains a (surrogate) value identical to some existing value in the E-relation STUDENT. Informally, this means that in the P-relation STUDENT_PROPERTIES we cannot represent the id-number, and/or the name and/or the sex of some entity, unless that entity is known to be of type STUDENT.

[8] The same property attributes could also be represented using two, or three P-relations. For example, one P-relation containing the single property attribute NUMBER, and one P-relation containing the two property attributes NAME and SEX.

[9] In general, the complete set.

Although the RM/T model uses exclusively E-attributes as primary keys for all relations, any attribute of a P-relation can be declared as *user key*. All attributes declared to be user keys are subject to the

- **User-key Integrity:** User key attributes do not accept null values and duplicates are not permitted.

The P-relations contained in our database are given in detail in Appendix B. We summarize the list of property attributes for each entity in the database.

Entity type	*Property attributes*
ACADEMIC	STATUS
ADMINISTR	FOR¢, POSITN
AFFILIATION	ACADEMIC¢, DEPARTMENT¢
AREA	NAME
BOOK	TITLE
BOOK_MVP	BOOK¢, AUTHOR
CLASS	COURSE¢, SEMESTER¢, INSTRUCTOR¢, FINAL
COMMITTEE	ACADEMIC¢, GRAD¢
COURSE	FIELD¢, NUMBER, UNITS
CURNT_DATE	DATE
DEPARTMENT	CHAIR¢, NAME, FACLTY
ENROLLMENT	STUDENT¢, CLASS¢, GRADE
FACILITY	OF¢, DIREC¢, NAME
GRAD	SUPRV¢, PROG
INSTRUCTOR	ACADEMIC¢, DEPARTMENT¢
OFFERED	AREA¢, DEPARTMENT¢
PRE_REQ	FOR_COURSE¢, IS_COURSE¢
ROOM	OFFICE¢, BUILDING, NUMBER
ROOM_MVP	ROOM¢, PHONE
SCHEDULE	TIME_TABLE¢, ROOM¢, CLASS¢
SEMESTER	TERM, YEAR
STAFF	DEPT¢, NUMBER, NAME, SEX
STUDENT	NUMBER, NAME, SEX
TEXT	BOOK¢, COURSE¢
TIME_TABLE	DAY, HOUR
UNDER_GRAD	
UNDER_MVP1	UNDER_GRAD¢, MAJOR
UNDER_MVP2	UNDER_GRAD¢, MINOR

Some of the property attributes in the above list have names ending in the special character '¢', that is, they are at the same time E-attributes. These property attributes represent references to other entity types. The specific roles they play will be explained in the following sections.

2.3. Entity classes

All RM/T entity types are classified into three disjoint *classes*: kernel, characteristic, and associative. For expository reasons Codd also introduces the concept of a 'nonentity association' [Codd 79], which has no E-relation and therefore no existence in its own right. Codd points out that

"... RM/T may be applied to database design completely avoiding the nonentity association concept altogether."

An additional constraint applies to those nonentity associations. We do not make use of this kind of association in our database.

2.3.1. Kernel entities

Kernel entities are entities that have totally independent existence. A kernel entity is one that is neither characteristic nor associative. For example, STUDENT and COURSE are kernel entities.

Aside from the entity integrity and the property integrity, no further constraints need to apply to kernel entities. However, some additional integrity constraints may apply if certain criteria are met, as we will see later in this section and in Section 2.4.

Our database contains the following kernel entities, see Appendix B:

Kernel entity	*Property attribute*
ACADEMIC	STATUS
ADMINISTR	FOR¢, POSITN
AREA	NAME
BOOK	TITLE
COURSE	FIELD¢, NUMBER, UNITS
CURNT_DATE	DATE
DEPARTMENT	CHAIR¢, NAME, FACLTY
FACILITY	OF¢, DIREC¢, NAME
GRAD	SUPRV¢, PROG
ROOM	OFFICE¢, BUILDING, NUMBER
SEMESTER	TERM, YEAR
STAFF	DEPT¢, NUMBER, NAME, SEX
STUDENT	NUMBER, NAME, SEX
TIME_TABLE	DAY, HOUR
UNDER_GRAD	

The entities GRAD and UNDER_GRAD are sub-entities of the STUDENT entity in the sense that an instance of any of these sub-entities is automatically also an instance of the superior entity STUDENT (see Appendix A, which contains an overview of the database structure). Figure 2-3 illustrates the representation of the superior kernel entity STUDENT and its two subtype kernel entities GRAD and UNDER_GRAD.

STUDENT

STUDENT¢
aadxg997
aadxg998
aadxg999
aadxh000
aadxh001
aadxh002

STUDENT_PROPERTIES

STUDENT¢	NUMBER	NAME	SEX
aadxg997	843901737	JONES	M
aadxg998	833904097	WALKER	M
aadxg999	871234567	SMITH	F
aadxh000	818767654	JONES	F
aadxh001	787656545	WHITE	M

GRAD

GRAD¢
aadxg997
aadxg998
aadxg999

GRAD_PROPERTIES

GRAD¢	SUPRV¢	PROG
aadxg997	aaprq250	CMPT
aadxg998	aaprr334	MATH
aadxg999		

UNDER_GRAD

UNDER_GRAD¢
aadxh000
aadxh001
aadxh002

Figure 2-3: Superior kernel entity type STUDENT
and its two subtype kernel entities GRAD and UNDER_GRAD

Properties of the superior entity STUDENT (for example, the student name NAME) are *inherited* by the subordinate entities. The analogue applies to the entities of type STAFF and their subordinate entities of types ADMINISTR and ACADEMIC. This requires a special constraint that applies to the database design.

■ **Attribute-Naming Integrity:** the P-relations for a given entity type do not have any attribute names in common with the P-relations for any supertype, at any level, of that entity type.

This constraint allows supertype properties to be automatically inherited by subtypes, without any risk of ambiguity. For example, entity type STUDENT is the only supertype (at any level) of entity type GRAD, and the only P-relations for the two entity types are STUDENT_PROPERTIES and GRAD_PROPERTY, respectively. The names SUPRV¢ and PROG of P-relation GRAD_PROPERTY are distinct from all the names of the property attributes of the P-relation STUDENT_PROPERTIES (NUMBER, NAME, and SEX). Thus the corresponding inheritance of properties cannot cause ambiguities.

A subtype reference is a reference from the E-relation for a subtype to the E-relation for an immediate supertype of that subtype. For example, the reference from the E-relation GRAD to the E-relation STUDENT is a subtype reference. A subtype reference is a total injective function, that is, the domain of the E-relation of a subtype is a subset of the domain of the superior E-relation. Expressed in the terminology of database theory, the following integrity constraint applies to subtype entities:

■ **Subtype Integrity:** Whenever a surrogate, say e, belongs to the E-relation for an entity of type E, e must also belong to the E-relation for each entity type for which E is a subtype.

For example, no (surrogate) value can exist in the E-attribute GRAD¢ of the E-relation GRAD, unless the same value exists in the E-attribute STUDENT¢ of the E-relation STUDENT (see the example in Figure 2-3). Informally this means that an entity cannot be represented as graduate student without being known to be a student in the first place.

The set of subtypes (GRAD and UNDER_GRAD) *spans* the supertype STUDENT per category STATUS,[10] that is, the union of the domains of the subtypes GRAD and UNDER_GRAD is identical to the domain of their supertype STUDENT. Expressed in the terminology of database theory, we have the following

■ **Spanning constraint:** If a set of subtypes spans a supertype per some category C, then every instance of the supertype must also be an instance of some subtype in that category C.

The subtypes ADMINISTR and ACADEMIC of the supertype STAFF do not span the supertype STAFF per category JOB, that is, an instance of an entity of type STAFF might not be an instance of any subtype per category JOB; this would be the case, for example, with technical personnel.

In RM/T* we add the notion of 'mutually exclusive subtypes'. For example, the two subtypes GRAD and UNDER_GRAD are mutually exclusive: any STUDENT can (and must due to the spanning constraint) only be either a GRAD or an UNDER_GRAD, but never both. However, exceptions are allowed among STAFF members, who could at times be administrators and academics at the same time. The subtypes ADMINISTR and ACADEMIC are not mutually exclusive. We obtain an additional

■ **Mutex Integrity:** No instance of a supertype entity can be an instance of two distinct subtypes if these subtypes are mutually exclusive.

Note that the concept of subtypes does not apply to kernel entities only. Entities of characteristic or associative type could have subtypes as well.

[10] The notion of a 'category' serves to identify a particular ramification within an arbitrarily complex hierarchy.

The motivation for including SEMESTER, TIME_TABLE and CURNT_DATE as kernel entities is because this makes it easier to deal with temporal constraints, as shown in Chapter 5. The entity type CURNT_DATE is a trivial entity in the sense that it always contains exactly one tuple specifying the current date and it gets updated automatically.

2.3.2. Associative entities

An associative entity is one whose function is to represent a many-to-many, or many-to-many-to-many, etc., relationship between two or more entities. For example, an ENROLLMENT represents an association between a STUDENT and a CLASS. Many students can be enrolled in one class, and each individual student can be enrolled in many classes.

As the name indicates, associative entities involve association references. An association reference is a reference from an E-attribute of a P-relation to the E-relation for a participant in that association. The participants of an association are the entity types that are associated via the association. For example, as Figure 2-4 illustrates, the participants of the association ENROLLMENT are the two entity types STUDENT and CLASS. Each value of the E-attribute STUDENT¢ of the P-relation ENROLLMENT_INSTANCE refers to the E-relation for the entity type DEPARTMENT; each value of the E-attribute CLASS¢ of the P-relation ENROLLMENT_INSTANCE refers to the E-relation for the entity type CLASS. Each associative entity has at least two, and may have more than two associative references among its immediate properties.

The following integrity constraint applies to associative entities:

■ **First Association Integrity:** Let A be an entity type belonging to the class of associative entities, and let E be the set of E-attributes that identifies the participants in A. Then a given instance of A can exist in the database only if, for that instance, each E-attribute in E either

1. has the value *E-null*, or
2. identifies an existing entity of the appropriate type.

In other words, the domain of each of those E-attributes is a subset of the surrogate values currently existing in the E-relation to which the particular E-attribute refers, together with the value E-null. The value *E-null* represents a null value in an E-attribute; that is, a null value where a system assigned surrogate was expected. [11]

[11] *E-null* values can only appear in a non-key E-attribute of a P-relation, that is, in a property attribute that refers to some E-relation; they cannot appear in the E-attribute primary key of a relation.

STUDENT

STUDENT¢
aadxg997
aadxg998
aadxh001
aadxh002

CLASS

CLASS¢
ccdxg317
ccdxg318
ccdxg319

ENROLLMENT

ENROLLMENT¢
eedxq657
eedxq658
eedxq659
eedxq660
eedxq661
eedxq662

ENROLLMENT_INSTANCE

ENROLLMENT¢	STUDENT¢	CLASS¢
eedxq657	aadxg997	ccdxg317
eedxq658	aadxg998	ccdxg317
eedxq659	aadxg998	ccdxg318
eedxq660	aadxh001	ccdxg318
eedxq661	aadxh001	ccdxg319
eedxq662	aadxh002	ccdxg317

ENROLLMENT_GRADE

ENROLLMENT¢	GRADE
eedxq657	A
eedxq658	B
eedxq662	A

Figure 2-4: The E-relation ENROLLMENT and the P-relation ENROLLMENT_INSTANCE for the associative entity type ENROLLMENT, together with the E-relations for the entity types STUDENT and CLASS, which are associated by ENROLLMENT.

For example, no entity of type ENROLLMENT can exist in the database, unless the following two conditions hold:

1. the E-attribute STUDENT¢ of the corresponding tuple in the P-relation ENROLLMENT_INSTANCE, contains a (surrogate) value, which is either
   - the value *E-null*, or
   - identical to some existing value in the E-relation STUDENT.
2. the E-attribute CLASS¢ of the corresponding tuple in the P-relation ENROLLMENT_INSTANCE, contains a (surrogate) value, which is either
   - the value *E-null*, or

• identical to some existing value in the E-relation CLASS.

Informally this means that no entity of type ENROLLMENT can be represented in the database, unless the two entities associated via the attributes STUDENT¢ and CLASS¢ are known to be of the appropriate entity types STUDENT and CLASS, respectively.

At least two of the attributes of the P-relations of an associative entity are E-attributes and refer to other entities, namely, to the ones that are associated. Additionally, the P-relations of an associative entity may contain attributes, which represent some 'normal' single-valued properties of the association. For example, the entity type ENROLLMENT, has the 'normal' single-valued property GRADE, representing the grade of a particular student in a particular class (see Figure 2-4).

In addition to the associative integrity rule, each associative entity is constrained by the

■ **Second Association Integrity:** for the P-relation containing the set of participants of the association, this set of participants constitutes a composite alternate key, [12] unless this set contains one or more E-null values.

For example, no two tuples in the P-relation ENROLLMENT_INSTANCE can have the same pair of values in their E-attributes STUDENT¢ and CLASS¢. If two tuples have the same values in their STUDENT¢ attribute, and one of them has an E-null value in its CLASS¢ attribute, then this E-null value substitutes some value that is necessarily distinct from the value in the CLASS¢ attribute of the other tuple. Codd introduces so called MAYBE versions of all the common relational database operators (such as SELECT, PROJECT, JOIN, etc.) in order to appropriately deal with these situations ([Codd 79]). For example, a MAYBE-SELECT applied to some attribute, would select all tuples containing a null value in the respective attribute field. Based on those MAYBE operators Codd introduces further operators, such as OUTER-UNION, OUTER-JOIN, etc. For our purposes these operators are irrelevant as we will see in Chapter 5.

Our database contains the following associative entities (for a complete description see Appendix B):

Associative entity	*Participants*
AFFILIATION	ACADEMIC¢, DEPARTMENT¢
CLASS	COURSE¢, SEMESTER¢, INSTRUCTOR¢
COMMITTEE	ACADEMIC¢, GRAD¢

[12] Here we exclude the exceptional and misleading case in which the participants of the association are distributed over several P-relations for the associative entity type.

ENROLLMENT	STUDENT¢, CLASS¢
INSTRUCTOR	ACADEMIC¢, DEPARTMENT¢
OFFERED	AREA¢, DEPARTMENT¢
PRE_REQ	FOR_COURSE¢, IS_COURSE¢
SCHEDULE	TIME_TABLE¢, ROOM¢
TEXT	BOOK¢, COURSE¢

Some of the associative entities exclusively associate kernel entities, while others associate kernel entities and other associative entities.

The names of the E-attributes in the P-relations for the associative entities indicate which entity type they are referring to (see Appendix B). In fact, they are all identical to the E-attribute names of the respective entity types, except for the attribute names of the PRE_REQ entity. The reason is that the PRE_REQ entity associates two entities of the same type (COURSE): one COURSE is a prerequisite of the other COURSE; two distinct names are necessary to differentiate between the two E-attributes referring to the same entity type.

2.3.3. Characteristic entities

A characteristic entity is one whose sole function is to qualify or describe some other, superior entity. It is a database construct used solely to represent multi-valued properties of the superior entity being characterized. A characteristic reference is a reference from an E-attribute of a P-relation of a characteristic entity to the E-relation of the immediately superior entity type.

As Figure 2-5 illustrates, an entity of type BOOK has the single valued property TITLE and the multi-valued property AUTHOR. To represent this multi-valued property (MVP), the characteristic entity BOOK_MVP is introduced. For each of the multiple values of AUTHOR that a particular BOOK has, there is one tuple in the E-relation BOOK_MVP and one tuple in the corresponding property relation BOOK_AUTHOR. The property relation BOOK_AUTHOR of the characteristic entity type BOOK_MVP contains two property attributes: the attribute AUTHOR containing the name of one author, and the E-attribute BOOK¢ identifying the particular BOOK of which one AUTHOR is represented in the same tuple. Each value in the E-attribute BOOK¢ of the relation BOOK_AUTHOR represents a characteristic reference to the superior entity type BOOK.

No E-null values are allowed in an E-attribute representing a characteristic reference, since a null value in such an attribute would in a sense be equivalent to a null value in the key attribute of a relation. The domain of an E-attribute representing a characteristic reference, is a subset of the values currently existing in the E-relation of the entity type being characterized. The mapping from the E-attribute representing a characteristic reference to the E-relation of the entity type being characterized is a total function:

BOOK

BOOK¢
bbdxg502
bbdxg503
bbdxg504
bbdxg505

BOOK_PROPERTY

BOOK¢	**TITLE**
bbdxg502	The Handbook of Artificial Intelligence
bbdxg503	The Knowledge Frontier
bbdxg504	Parallel Distributed Processing
bbdxg505	

BOOK_MVP

BOOK_MVP¢
eeyxr657
eeyxr658
eeyxr659
eeyxq280
eeyxq281
eekxq716
eekxq717

BOOK_AUTHOR

BOOK_MVP¢	**BOOK¢**	**AUTHOR**
eeyxr657	bbdxg502	Avron Barr
eeyxr658	bbdxg502	Paul R. Cohen
eeyxr659	bbdxg502	Edward A. Feigenbaum
eeyxq280	bbdxg503	Nick Cercone
eeyxq281	bbdxg503	Gordon McCalla
eekxq716	bbdxg504	James L. McClelland
eekxq717	bbdxg504	David E. Rumelhart

Figure 2-5: The E-relation and the P-relation for kernel entity type BOOK and the E-relation and the P-relation for the characteristic entity type BOOK_MVP, characterizing the superior entity type BOOK.

- **Characteristic Integrity:** A characteristic entity cannot exist in the database unless the entity it describes is also in the database.

For example, no entity of type BOOK_MVP can exist in the database, unless the E-attribute BOOK¢ of the corresponding tuple in the P-relation BOOK_AUTHOR contains a (surrogate) value identical to some currently existing value in the E-relation BOOK. Informally this means that no author can be represented in the database, unless the entity described by the author (via the E-attribute BOOK) is known to be of the appropriate type BOOK.

The correspondence of the entity described to its characteristic entity is one-to-many, whereas the inverse correspondence is one-to-one. For example, a BOOK could have several AUTHORS, but any given AUTHOR can characterize one BOOK only. This might seem surprising at first sight, since we would expect some authors to have written a number of distinct books. The important thing to notice is that due to a deliberate choice of the database designer to make BOOK_AUTHORS existence-dependent on BOOKS, they are represented as distinct entities as they describe distinct courses. In our database an AUTHOR is a property, not an entity. In RM/T* we use as a naming convention the suffix '_MVP' for all characteristic entities to clarify this concept. Naming the characteristic entity 'AUTHOR' and the property 'NAME' (analogous to examples given in [Codd 79] and [Date 83]) would disguise the concept.

An additional constraint of our RM/T* model is the restriction in the use of characteristic entities to a single level, that is, characteristic entities should not be allowed to in turn have characteristic entities describing them. Such a construct would in fact be one representing a property, which in turn had properties and would be prone to update anomalies. For example, suppose we represent a book as a characteristic entity, describing a course, that is, a BOOK will be a multi-valued property of a COURSE. A particular BOOK may be used for several courses, and thus be represented several times. For each individual representation the set of AUTHORs has to be entered in the system. It is now possible, that due to a mistake during data entry, 'the same book' has two distinct sets of authors. Since the user need not be aware of the distinct surrogates involved in the distinct representations (this is an important feature of the RM/T model), the distinct representations of a particular book, with distinct sets of authors might cause misinterpretations by the user. In short, once a property type is allowed to in turn have properties, it should no longer be considered as property type, but turned into an entity type.

All versions of the RM/T model offer all the constructs required for this approach: the entity type BOOK can be represented as kernel entity instead of characteristic entity, and associated with the entity type COURSE via an associative entity type. We use the associative entity type TEXT for this purpose. A particular COURSE can be associated to many distinct BOOKs and a particular BOOK can be associated to many distinct COURSEs. Thus a particular BOOK is no longer represented several times and the corresponding anomalies cannot occur.

Our database contains the following characteristic entities (see Appendix B):

Characteristic entity	*Entity being characterized*	*Multi-valued property*
BOOK_MVP	BOOK	AUTHOR
ROOM_MVP	ROOM	PHONE
UNDER_MVP1	UNDER_GRAD	MAJOR
UNDER_MVP2	UNDER_GRAD	MINOR

2.4. Designative references

Independently of the class (kernel, associative, or characteristic) an entity type belongs to, it can also *designate* another entity type. For example, as illustrated in Figure 2-6, the kernel entity type GRAD designates the kernel entity type ACADEMIC. Values of the E-attribute SUPRV¢ of the P-relation GRAD_PROPERTY refer to the E-relation of entity type ACADEMIC. The corresponding reference is called a *designative reference.*

ACADEMIC

ACADEMIC¢
aafyg997
aafyg998
aafyh001
aafyh002

ACADEMIC_PROPERTY

ACADEMIC¢	STATUS
aafyg997	PROF
aafyg998	ASSI
aafyh001	PROF
aafyh002	ASSO

GRAD

GRAD¢
aggxq652
aggxq653
aggxq654
aggxq655
aggxq656
aggxq657

GRAD_PROPERTIES

GRAD¢	SUPRV¢	PROG
aggxq652	aafyh001	PhD
aggxq653	aafyh001	MSc
aggxq654	aafyg997	PhD
aggxq655	aafyg997	PhD
aggxq656	aafyh002	MSc
aggxq657	aafyg998	MSc

Figure 2-6: The E-relation and P-relation for kernel entity type ACADEMIC and the E-relation and the P-relation for the designative kernel entity type GRAD, designating the entity type ACADEMIC.

An ACADEMIC could supervise several GRADs, but any given GRAD can have only one supervisor and thus designate only one ACADEMIC.

The domain of an E-attribute referring to the E-relation of the entity type being designated is a subset of the surrogate values currently existing in that E-relation, united with the value E-null. Expressed in the terminology of database theory, the following constraint applies to designative entities:

- **Designative Integrity:** Let D be a designative entity type, and let E be the set of E-attributes representing designations by D. Then a given instance of D can exist in the database only if, for that instance, each E-attribute in E either

1. has the value *E-null*, or
2. identifies an existing entity of the appropriate type.

For example, no entity of type GRAD can exist in the database unless the E-attribute SUPRV¢ of the corresponding tuple in the P-relation GRAD_PROPERTIES contains a value which is either
1. the value *E-null*, or
2. identical to some currently existing value in the E-relation ACADEMIC.

Informally this means that no graduate student can be represented in the database, unless the entity described by that student via the E-attribute SUPRV¢ is either completely unknown, or known to be of the appropriate type ACADEMIC.

There is an obvious similarity between designative entities and characteristic entities. In both cases some other entity is indirectly described by the properties of the characteristic entity or the properties of the designative entity. In [Date 86b] (footnote on page 616) Date states that

"A characteristic entity is in fact a special case of a designative entity; it is really nothing more than a designating entity that happens to be existence-dependent on the entity it designates."

However, this statement is not correct, as we now show. Many instances of a characteristic entity type can describe one particular instance of the superior entity type; and in exactly the same way, many instances of a designative entity type can describe one particular instance of the entity type being designated. Similarly, one particular instance of a characteristic entity type can characterize only one instance of the superior entity type; and in exactly the same way, one particular instance of a designative entity type can designate only one instance of the entity type being designated.

The fundamental difference is this: Since the sole purpose of a characteristic entity is to represent multi-valued properties of some superior entity (the one on which it is existence-dependent), those properties that characterize several instances of the superior entity, are represented several times. No equivalent is possible with a non-characteristic entity type.

For example, consider the characteristic entity type BOOK_MVP used to represent the multi-valued property AUTHOR in our database. If one particular AUTHOR has co-authored *n* distinct BOOKs, then this AUTHOR is represented *n* times. If instead we had used a kernel entity AUTHOR designating the entity type BOOK, then each AUTHOR could only designate one particular BOOK, or else several distinct surrogates would identify the same real-world author, which would undermine the whole idea of surrogates as ***unique*** identifiers.

The somewhat misleading (or at least missing) naming conventions used by both Codd [Codd 79] and

Date [Date 83] [13] might have led Date to his conclusion cited above. It is important to notice that among all entities, characteristic *entities* are unique in the sense that they are in fact constructs to represent nothing but *properties*.

Designations do have some particulars in common with associations. An associative entity could be seen as *designating* the entities it associated. For example, the associative entity type ENROLLMENT could be seen as designating the kernel entity type STUDENT and also designating the associative entity type CLASS. However, in order to obtain a completely equivalent representation using designations instead of the association, we would have to add the additional constraint, that the two attributes STUDENT¢ and CLASS¢ must constitute a composite primary key of the P-relation ENROLLMENT_INSTANCE (unless E-null values are involved). Otherwise we could obtain, for example, two distinct representations of one particular student enrolled in one particular class, having distinct grades in the two representations.

The database designer uses his discretion to determine what classes of references to use in order to represent the interrelationships among real-world entities and their properties, and what classes of entities to use in order to represent the real world entities themselves. The choices made reflect the database designer's understanding of the world.

Our database contains the following designative entities (see Appendix B):

Entity type	*Designative E-attribute*	*Designated entity type*
ADMINISTR	FOR¢	FACILITY
COURSE	FIELD¢	AREA
DEPARTMENT	CHAIR¢	ACADEMIC
FACILITY	OF¢	DEPARTMENT
FACILITY	DIREC¢	ACADEMIC
GRAD	SUPRV¢	ACADEMIC
ROOM	OFFICE¢	STAFF
SCHEDULE	CLASS¢	CLASS
STAFF	DEPT¢	DEPARTMENT

The names of the E-attributes in the P-relations designating other entity types, indicate which role the designation plays.

We could alternatively declare the entity type SCHEDULE as purely associative, that is, with a set of

[13] Both Codd and Date would probably have used the name AUTHOR for the characteristic entity BOOK_MVP, and the name NAME for the property AUTHOR.

three participants: ROOM, TIME_TABLE, and CLASS. However, this choice has unfavorable consequences. Since the composite alternate key now consist of the three attributes ROOM¢, TIME_TABLE¢, and CLASS¢, two distinct classes could be assigned to the same room at the same time. An additional integrity constraint for this particular relation would be required.

Similarly, if we had declared the entity type CLASS (see Section 2.3.2) to associate only the two entity types COURSE and SEMESTER, and to designate the entity type INSTRUCTOR, then only one INSTRUCTOR could teach a particular COURSE in a particular SEMESTER.

2.5. Additional integrity constraints

Integrity constraints reflect a part of the database designer's understanding of the micro-world represented. All the constraints specified so far, are direct consequences of the RM/T model. Every database system adhering to the RM/T model is subject to those constraints. Thus the database designer is forced to structure the micro-world to be represented, according to rigid guidelines. This task is at times not trivial, but has great advantages. Due to the rigid structure of the RM/T model, once the database is designed according to this model, only very few additional constraints are required in order to obtain an accurate representation of the particular micro-world. Those additional constraints will generally turn out to be very specific.

We delay the introduction of additional integrity constraints until Chapter 5, where some of the RM/T-integrity constraints and some of those additional integrity constraints will be exploited to eliminate wasteful operations in the extensional database.

The description of the extensional part of the database is now complete. In Chapter 3 we describe the catalog, which contains information about the structure of the database. The catalog contains information about what entity types exist in the database, to which classes (kernel, associative, characteristic) they belong, what properties they have, and so on. The contents of the extensional database changes after each update, whereas the contents of the catalog remains constant at all times, except when the structure of the database is changed. In this latter case we could also speak of a new database with its new catalog, which again would remain constant throughout the lifetime of that database.

Chapter 3
The catalog of the RM/T* database

Part of the RM/T model is a ***catalog*** which describes the structure and the functional dependencies of the actual database. The catalog can be considered as a database in its own right. It will serve as a crucial source of information required to predict the occurrence of null events.

Since the catalog represents the time invariant part of the database, both its structure and contents can be specified. These details are given in Appendix C. We introduce the different concepts involved with the catalog and explain how to interpret its contents. Date's extended version of the RM/T catalog [Date 83] slightly deviates from Codd's original proposal [Codd 79]: The primary keys of the catalog relations are not E-attributes; a designation graph relation, which was not present in Codd's original version, is added; and the notational efficacy is improved. Our RM/T* version basically follows Date's version, but includes some further extensions to the model as we show in this chapter.

3.1. DOMAINS, RELATIONS and ATTRIBUTES relations

The relations of the catalog can be divided into two types: graph relations and non-graph relations. We begin our presentation with the three non-graph relations: DOMAINS, RELATIONS and ATTRIBUTES.

3.1.1. The CATLG_DOMAINS relation

- CATLG_DOMAINS (DOMNAME, DATA_TYPE, QUALIFIER, ORDERING)

The CATLG_DOMAINS relation (see Appendix C.1) contains a tuple for each domain in the database, giving the name (DOMNAME) and data-type (DATA_TYPE) for the domain in question and indicating whether the '>' predicate is applicable between values of that domain (ORDERING is either YES or NO).

In RM/T*, we add the attribute QUALIFIER to the catalog relation CATLG_DOMAINS in order to qualify the data-types specified in the DATA_TYPE attribute field.

The interpretation of the QUALIFIER attribute field depends on the value in the DATA_TYPE field. The following list shows the six data-types used in our database, together with the interpretation of the corresponding qualifier attribute.

DATATYPE	QUALIFIER	Comments
boolean	RelName	the name of the relation containing the two boolean constants
enumeration	RelName	the name of the relation enumerating the members of the domain
integer	min — max	the smallest and the largest member of the domain
real	min — max	the smallest and the largest member of the domain
string	Number	the number of elements (characters) in the string
surrogate		no qualifier

For example, the qualifier RelName applies to enumeration types and specifyes the name of the relation in which the members are enumerated. This data-type can, but need not be ordered. If ORDERING is 'YES', then the relation RelName enumerates the members in ascending order, that is, the first element specified in the relation RelName is the 'lowest' element. For example, the DATA_TYPE for the domain GRADE is 'GRADE'; in relation GRADE the tuple with ELEMENT# = '1' contains the lowest value of the GRADE domain.

The qualifier for boolean data-types is also RelName. The respective relation contains the two boolean constants, that is, the two values, say A and B, for which '$\neg A = B$' and '$\neg B = A$' holds. In our database we have only one such data-type, namely SEX, with '$\neg$ *Male* = *Female*' and '$\neg$ *Female* = *Male*'.

The relations named in the QUALIFIER attribute of relation CATLG_DOMAINS in Appendix C.1 are rendered in Appendix C.2.

The domain of all E-attributes in the RM/T models is the E-domain, containing system generated surrogate values, each uniquely identifying an instance of some entity type. The E-domain carries no information with respect to the different entity types. Such information can be extracted from the names of the E-attributes and from the catalog. For example, the E-attribute GRADS contains those surrogate values that uniquely identify instances of entities of type GRAD. The subtype graph (see Section 3.2.5) carries further information about this set of surrogate values with respect to the set of surrogate values identifying entities of type STUDENT and the set of surrogate values identifying entities of type UNDER_GRAD.

In RM/T* we add the relation ATTRIBUTE_DOMAINS (see Appendix C.3) to represent further information about the different domains of the P-attributes in the database.

- ATTRIBUTE_DOMAINS
  (RELNAME, ATTNAME, DOMNAME, CARDINALITY, EL_KEY, INTRV_KEY)

The attributes of relation ATTRIBUTE_DOMAINS have the following function:

- RELNAME and ATTNAME form the composite primary key of the relation and identify a particular attribute within a particular relation in the database.
- DOMNAME specifies the corresponding domain name.
- CARDINALITY specifies the cardinality of the set of domain values, if this is possible.
- EL_KEY contains one component of a composite primary key of the relation ELEMENTS. This relation is used to represent domains, which are subdomains of some other domains. The concept is similar to that of the hierarchical aspect among entity types. Here the domain of a particular property can be specified as the subdomain of another domain. For example, in our database more fields are being offered, than there are departments. Consequently, the domain of the P-attribute DEPARTMENT_PROPERTIES.NAME is a subset of the domain of the P-attribute AREA_NAME.NAME. The index to the elements (listed in relation FIELD, see Appendix C.2) of this subset are listed in the relation ELEMENTS together with the value of EL_KEY (see Appendix C.4).
- INTRV_KEY has the same purpose as EL_KEY, but for those domains that have ordered elements, such that they can be specified via minimum and maximum values. INTRV_KEY contains one component of a composite primary key of the relation INTERVALS.

There are several sets of domains that use the same data type. The distinct names (DOMNAME) facilitate the checking required to avoid nonsensical join operations. For example, although the two attributes BOOK.TITLE and STUDENT.NAME have the same data type ('string'), a join operation applied to these two attributes is nonsensical. By naming the respective domains distinctly and by recording the distinction in the ATTRIBUTES relation (see Section 3.1.3) such a nonsensical operation can easily be avoided.

3.1.2. The CATLG_RELATIONS relation

- CATLG_RELATIONS (<u>RELNAME</u>, RELTYPE)

The CATLG_RELATIONS relation (see Appendix C.6) contains a tuple for every relation in the database, giving the name (RELNAME) and type (RELTYPE) of the relation in question. The attribute (RELTYPE) specifies one of the two kinds of relations, and for E-relations, the class of the respective entity type, whether it is also designative, and whether or not it is a subtype. The specification is done by concatenating the appropriate letters from the following list:

P	property relation
E	E-relation
I	inner kernel entity type relation
K	kernel entity type
A	associative entity type
C	characteristic entity type
D	designative

Inner kernel entities are entities that are not subtypes of any other entity.

3.1.3. The CATLG_ATTRIBUTES relation

- CATLG_ATTRIBUTES
  (RELNAME, ATTNAME, DOMNAME, PKEY, UKEY, NULLS, EXC)

The CATLG_ATTRIBUTES relation (see Appendix C.7) contains a tuple for each attribute of each relation in the database, giving the relation-name, attribute-name, and underlying domain-name, and also indicating whether the attribute

- participates in the primary key of the relation concerned (PKEY is YES or NO);
- participates in a user key for the entity type concerned (UKEY is YES or NO); and
- can accept null values (NULLS is YES or NO).

As mentioned in Section 3.1.1, the distinct domain names recorded in attribute DOMNAME facilitate the elimination of nonsensical join operations: a join operation can be applied only to attributes that have identical domain names.

In RM/T* we add the attribute EXC, which serves to identify constraints which are activated whenever the respective attribute is to be accessed in the database. These constraints are recorded in the special table EXCEPTION_RULES, see Appendix D.1, which is basically a list of lists, where each sublist contains one constraint. The details concerning the contents and use of the attribute EXC and of the table EXCEPTION_RULES will be explained in Chapter 5.

3.2. Graph relations

The catalog also includes a set of ***graph relations***, whose function is to represent the various connections among relations in the database — for example, the connection between an E-relation and its corresponding P-relation(s). In order to increase the notational efficacy, we change some of the names proposed in [Codd 79] and [Date 83], but the respective concepts and applicable constraints remain unaltered. The graph relations are the following:

PROPT_GRAPH	:	the property graph relation
CHARC_GRAPH	:	the characteristic graph relation
ASSOC_GRAPH	:	the association graph relation
DESIG_GRAPH	:	the designation graph relation
SUBTP_GRAPH	:	the subtype graph relation

Neither the concept of a designation nor the DESIG_GRAPH relation is mentioned in [Codd 79] but they are introduced as an extension in [Date 83]. On the other hand, Date considers only unconditional generalizations with the SUBTP_GRAPH relation, while Codd used two relations in place of the SUBTP_GRAGH relation, one for unconditional generalization and one for alternate generalization.

3.2.1. The property graph relation

• PROPT_GRAPH (P_RELNAME, E_RELNAME)

The PROPT_GRAPH relation (see Appendix C.8) contains a tuple for every P-relation in the database, giving the name of that P-relation and the name of the corresponding E-relation, as indicated by the respective attribute names.

The property graph is a collection of disjoint trees, in the sense that no two E-relations have a P-relation in common. Each of those trees consists of a 'parent' (the E-relation) and a set of 'children' (the P-relations), and thus involves exactly two levels.

3.2.2. The association graph relation

• ASSOC_GRAPH
(ASSOCIATION_E_RELNAME, ASSOCIATION_P_ATTNAME, PARTICIPANT_E_RELNAME)

The ASSOC_GRAPH relation (see Appendix C.9) contains a tuple for every participant in every association in the database. This graph cannot be represented by a binary relation since such a relation would lead to ambiguities in certain cases. For example consider the associative E-relation PRE_REQ in the database (see Appendix B). This relation associates two entities of the same type. In order to distinguish between the two attributes and the role they play in the association, we need to give them two distinct names. Thus, for each participant in the association we must record the participant name and the name of the entity it refers to. The original RM/T proposal [Codd 79] did not take this case into account, however, the extended model [Date 83] does so.

Each tuple of the association graph relation gives the name of an associative E-relation, the name of the attribute that identifies a participant in the corresponding P-relation of the association, together with the name of the E-relation of that participant. The association graph is not a collection of disjoint trees, in general: a given entity type can participate several times in a given association and/or in multiple associations (for example, the entity type COURSE in our database participates twice in the associative entity type PRE_REQ and it also participates in the associative entity types CLASS and TEXT).

3.2.3. The characteristic graph relation

• CHARC_GRAPH (CHARACTERISTIC_E_RELNAME, SUPERIOR_E_RELNAME)

The CHARC_GRAPH relation (see Appendix C.10) contains a tuple for every characteristic E-relation in the database, giving the name of that E-relation and the name of the immediately superior E-relation.

The characteristic graph, like the property graph, is a collection of disjoint trees. According to Codd's [Codd 79] and Date's [Date 83] definition, individual trees are not necessarily restricted to two levels (superior entity type T1 may have a characteristic entity type T2, which in turn may have a lower level characteristic entity type T3, and so on). However, as described in Section 2.3.3, we do not allow characteristic entities to have in turn characteristic entities describing them. Thus in our database all trees in the CHARC_GRAPH relation have only two levels.

3.2.4. The designation graph relation

• DESIG_GRAPH
(DESIGNATIVE_E_RELNAME, DESIGNATIVE_P_ATTNAME, DESIGNATED_E_RELNAME)

The DESIG_GRAPH relation (see Appendix C.11) contains a tuple for every designation in the database. Each tuple of the designation graph relation gives the name of a designative E-relation, the name of the E-attribute that identifies the designated entity type in the corresponding P-relation of the designation, together with the name of the designated entity type. Neither in general, nor in our specific case is the designation graph a collection of disjoint trees.

3.2.5. The subtype graph relation

• SUBTP_GRAPH (SUBTYPE_E_RELNAME, SUPERTYPE_E_RELNAME, CATEGORY, SPANNING_SUPERTYPE, MUTUALLY_EXCLUSIVE)

The SUBTP_GRAPH relation (see Appendix C.12) contains a tuple for every immediate subtype/supertype relationship in the database. Each such tuple gives the names of the E-relations for the subtype and the supertype, together with the name of the applicable category. The subtype graph is not a collection of disjoint trees, in general. It corresponds to Codd's *unconditional generalization by inclusion relation* (UGI-relation) [Codd 79]. In RM/T* we add the attribute SPANNING_SUPERTYPE to indicate whether or not the respective category spans the supertype and the attribute MUTUALLY_EXCLUSIVE to indicate whether or not the subtypes are mutually exclusive.

3.2.6. Alternative generalizations

Alternative generalizations are necessary if an entity type may have two or more distinct supertypes. For example, we might want to represent the fact that an instructor could be either an academic staff member, or a graduate student. We could then add a kernel entity type LECTURER, which would be declared to be a subtype of either the kernel entity type ACADEMIC, or, alternatively, a subtype of the kernel entity type GRAD. This would be recorded in an *alternative generalization by inclusion relation* (AGI-relation). The associative entity type INSTRUCTOR would then associate entities of type LECTURER and entities of type DEPARTMENT.

Consider further the possibility of having an additional kernel entity type PERSON, and the entity types STAFF and STUDENT to be declared as subtypes of PERSON. This would be recorded in the SUBT_GRAPH-relation. Consider now the introduction of a new instance of type LECTURER into the database. The AGI-relation tells us that the LECTURER must either be an ACADEMIC, or a GRAD, but we cannot deduce which one. However, the SUBT_GRAPH-relation allows us to deduce that the surrogate added to the E-relation LECTURER to identify the new lecturer, can also be added to the E-relation PERSON, since no matter whether the particular LECTURER is an ACADEMIC, or a GRAD, he must also be a PERSON.

Our database does not make use of the concept of alternative generalization, therefore we do not include an AGI-relation in the graph relations of our catalog.

Our database also does not make use of the concept of *cover aggregation* and therefore the graph relations of the catalog do not include a *cover membership relation*.

3.2.7. Temporal constraints

In [Codd 79] Codd proposes four different graph relations to deal with temporal constraints. These are

- the *unconditional successor relation* (US-relation),
- the *alternative successor relation* (AS-relation),
- the *unconditional precedence relation* (UP-relation), and
- the *alternative precedence relation* (AP-relation).

The only temporal constraint imposed on the world represented by our database refers to the properties FINAL and GRADE of the entities CLASS and ENROLLMENT, respectively: The value of the DATE attribute of the CURRENT_DATE must have 'passed' the value of the FINAL attribute of a CLASS, before the GRADE attribute of an ENROLLMENT which associates that class to some STUDENT, can have a non-null value. Informally, this means that the final exam of a class must have been written, before a grade can be expected.

We are mainly dealing with properties and not with event type entities as conceived in [Codd 79]. None of the graph relations proposed there is appropriate to deal with the situation in our database. Therefore, the graph relations of our catalog do not include any of the four graph relations mentioned above. The knowledge-based subsystem is used to deal with our particular situation, as will be shown in Chapter 6.

In this and the previous chapter, we have presented our database and its catalog. We are now in a better position to approach the specific problems concerning wasteful operations that might occur in this database, and in relational databases in general. In Chapter 4 we clarify a fundamental distinction between the two major kinds of null events and subsequently present a classification of null events as they occur during query processing.

Chapter 4
Classification of null events

Since null events generally indicate that the user's request for information cannot be satisfied, they constitute one primary area for query optimization. Thus a database system should be designed in such a way that null events can be detected quickly and with least effort. In Chapters 2 and 3 we presented the basic database and its associated catalog which will serve as a concrete example for the general study of null events that might occur in a natural language — database system. In the present chapter we examine the different possible manifestations of null events as they occur during query processing.

For convenience, we repeat our distinction between null events, null values, and null responses, as stated in Chapter 1:

- A *null value* is an actual value occupying some memory space that would otherwise be used for a genuine value. The null value represents the fact that some information is missing in the field, where the null value is found.
- A *null response* (or empty response) is one that contains no information.
- A *null event* is an event that may happen sometime during query processing, and that results in the generation of a null response unless special actions are taken.

Thus a null response is the result of some null event. A null event, in turn, is often erroneously seen as the result of some null value (for example, see [Codd 79], [Vassiliou 79], [Atzeni and Parker 82], [Codd 86], [Date 86b], [Codd 87]). As a consequence of this view, phenomena such as type mismatches, which also cause null events, are erroneously equated to null values.

According to SPARC/DBMS Study Group of the American National Standards Committee [ANSI 75], there are 14 different manifestations of null, see Figure 4-1.

The different meanings of null values are usually classified into two distinct types, with the meaning '*value at present unknown*' and '*property inapplicable*', respectively. Thus the first null value in Figure 4-1 would be of type '*property inapplicable*'; null values 2 through 13 of type '*value at present unknown*'; and null value 14 of either type, depending on the type of null value from which is has been derived. Notice that

1. Not valid for this individual (e.g., maiden name of male employee)
2. Valid, but does not yet exist for this individual (e.g., married name of female unmarried employee)
3. Exists, but not permitted to be logically stored (e.g., religion of this employee)
4. Exists, but not knowable for this individual (e.g., last efficiency rating of an employee who worked for another company)
5. Exists, but not yet logically stored for this individual (e.g., medical history of newly hired employee)
6. Logically stored, but subsequently logically deleted
7. Logically stored, but not yet available
8. Available, but undergoing change (may be no longer valid)
    - Change begun, but new values not yet computed
    - Change incomplete, committed values are part new, part old, may be inconsistent
    - Change incomplete, but part new values not yet committed
    - Change complete, but new values not yet committed
9. Available, but of suspect validity (unreliable)
    - Possible failure in conceptual data acquisition
    - Possible failure in internal data maintenance
10. Available, but invalid
    - Not too bad
    - Too bad
11. Secured for this class of conceptual data
12. Secured for this individual object
13. Secured at this time
14. Derived from null conceptual data (any of the above)

Figure 4-1: Different manifestations of null, as given by the ANSI Study Group on DBMS [ANSI 75]

this is not the only possible interpretation of the manifestation of nulls in Figure 4-1. For example, the second manifestation could also be interpreted as *'property currently inapplicable'*.

Although this classification has found wide acceptance (for example, see [ANSI 75], [Codd 79], [Vassiliou 79], [Atzeni and Parker 82], [Reiter 84], [Codd 86], [Date 86b], [Codd 87]), we feel uncomfortable with it, since it seems to lead to less rigorous solutions to the problem of dealing with null events.

With respect to null values of type *'value at present unknown'* Reiter ([Reiter 84]) proposes an approach using Skolem constants instead of uniform null values, that is, existentially quantified variables. Thus a distinct null value would be used for each occurrence of some missing piece of information. Referring to this approach, Codd ([Codd 86]) points out that

> Under certain circumstances the database management system **might be able** to deduce equality or inequality between two distinctly named variables — or it might be able to deduce certain other constraints on the variables. However, it would rarely be possible for the database management system to deduce the actual values of these variables. Instead, most of the missing or unknown items are eventually supplied by users in the form of late-arriving input.

To this argument we could add the fact that in case it is possible to deduce the actual values of missing information, the explicit storage of this 'missing' information is redundant and might reflect some flaw in the database design. We are faced with a different situation than in deductive databases, where the information represented allows the deduction of further information which is not to be explicitly represented in the database. Neither approach to deal with null values of type *'value at present unknown'* seems to be perfect; we follow Codd's approach.

The notion of a null value of type '*property inapplicable*' is clearly misleading. This classification suggests an analogy between null events of type '*value at present unknown*' caused by null values of type '*value at present unknown*' and null events of type '*property inapplicable*' caused by null values of type '*property inapplicable*'. However, null events of type '*property inapplicable*' are, in general, not caused by a corresponding null value, but by a '*type mismatch*': the property MAIDEN_NAME is inapplicable to a MALE_EMPLOYEE not because the corresponding MAIDEN_NAME attribute field contains some specific value, which is inapplicable, but because the property 'maiden name' does not match any of the properties of a male person in general. The representation of misconceptions in the database will only cause additional problems, as we will show in Chapter 5 and should never be allowed in a proper database design.

From now on we make use of the following notation to clearly distinguish between four distinct concepts: The upper case greek letters Ω and Ξ denote null events of type '*value at present unknown*' and '*property inapplicable*', respectively; the lower case greek letters ω and ξ denote null values of type '*value at present unknown*' and '*property inapplicable*', respectively.

If information concerning the maiden names of female employees is to be represented in the database, MALE_EMPLOYEEs and FEMALE_EMPLOYEEs could be represented as distinct subtypes of the entity type EMPLOYEE. The distinction can be seen as analogous to the one between UNDER_GRAD students and GRAD students with respect to the attribute SUPRVisor: it is applicable to GRAD students, and inapplicable to UNDER_GRAD students.

Another proper representation would simply split up the different property attributes such that only those properties are grouped together in one relation, which are guaranteed to be applicable at the same time. We would then use two distinct relations, both having the same primary key, and one of them would represent such properties as the number, the name, and the sex of the employee, [14] while the other would represent the maiden_name of the employee. This latter relation would contain tuples only for female employees, for which the maiden_name is applicable, and none for male employees. This approach is similar to the one taken in our database to represent the property GRADE of entity type ENROLLMENT. An ENROLLMENT associating a STUDENT and a CLASS might exist in the database at a time, when the property GRADE is inapplicable to that ENROLLMENT. Therefore the GRADE is represented in a separate P-relation, whenever it is applicable, and not represented at all if it is inapplicable.

If the database is not properly designed, ξ-type null values could be entered in the those attribute fields that are inapplicable to the entity represented by the complete tuple (for example in the MAIDEN_NAME field of a male EMPLOYEE, or the SUPRV field of an undergraduate STUDENT), but even in that case they are not required: No matter what value is entered in the MAIDEN_NAME attribute field of a male EMPLOYEE, be it an ω-type null value, an ξ-type null value, or any normal value, the property MAIDEN_NAME always remains inapplicable to a male EMPLOYEE. An Ξ-type null event caused by a user query, which request the retrieval of some inapplicable information, can be seen as a misconception on the part of the user. An Ξ-type null event caused by an ξ-type null value represented in the database, can be seen as a misconception on the part of the database designer. As we have shown above, a solution to the latter problem can easily be achieved by a proper database design and is not worth extensive elaboration. A solution to the former problem will be presented in Chapter 5.

Notice, that also Ω-type null events might occur without a corresponding ω-type null value in the database. For example, in our database a query such as

- Q.: *Is John Doe an 'academic' ?*

could result in an Ω-type null event since John Doe could be represented as STAFF in the database but he might not be represented as either ADMINISTR or ACADEMIC. Making use of the 'open world' assumption, we can interpret a missing value in the E-attribute primary key of some relation, that is, a missing surrogate, [15] as an Ω-type null value. Thus both a missing property and a missing entity can be the cause of an Ω-type null event.

For pragmatic reasons, we use the capabilities and knowledge of the database system as a guideline for our classification. The null events are strictly dependent on the interrelation between the query to the

[14] Some of which might be 'currently unknown', but none of which would be 'inapplicable' at any time.

[15] notice that all surrogates are used to identify entities

particular system and the system itself. One query can result in different responses from different systems. Since the classification at hand is for null events occurring in a relational database, the criteria for classification are directly related to the way the query that triggers the null event is (or can be) processed in a relational database. Thus, the classification can in turn be used as a guideline for a strategy to efficiently process the query.

Our classification is primarily based on minimizing the amount of operations required on the database and on the knowledge-base in order to detect the null event. We use the term 'knowledge-base' for the ensemble of the RM/T* catalog and some additional, as yet unspecified representations of relevant knowledge. The term 'database' will be used for the basic database excluding the catalog. Thus, the database contains all the extensional facts, whereas the knowledge-base contains all the intensional facts.

We assume a system structure as depicted in Figure 1-2. The NL-query is parsed by a parser and analyzed by a semantic interpreter. Both, the parser and the semantic interpreter have access to the lexicon of the database. The output of the semantic interpreter is the logical form of the query. The logical form serves as input to both the query generator of the database system and the knowledge-based subsystem. The knowledge-based subsystem supervises the query generator and the query evaluator.

We divide the database processing required to generate an answer to a query into three groups:

1. no database access,
2. database access
    a. - including select and project operations, but excluding join operations,
    b. - database access including join (and all other) operations

An access to the database might or might not be necessary to detect a null event. Similarly, after some database accesses have been performed, subsequent join operations might or might not be necessary to detect the null event. Our classification reflects these distinctions. Figure 4-2 shows an overview of the classification.

Null event can be detected... :

increase in processing →

increase in KB-support ↓

	prior to DB access	after some DB access	
		without join	with join
without KB-subs. support	1.1	2.1.1	2.2
with KB-subsystem support	1.2	2.1.2	

Figure 4-2: Overview of the classification of null events

Null values in class 1.1 can be detected without database access. A knowledge-based subsystem is not necessary to detect those null values.

Null values in class 1.2 can be detected without database access, if the support of some suitable knowledge-based subsystem is available. Without a suitable knowledge-based subsystem, these null values cannot be detected prior to a database access.

Null values of type 2.1.1 require some database accesses but no join operations, in order to be detected. The database accesses cannot be dispensed with whether or not a knowledge-based subsystem is available; but once the database accesses have been performed, the null values can be detected without support of a knowledge-based subsystem.

Null values in class 2.1.2 require some database accesses but no join operations, in order to be detected, if the support of some suitable knowledge-based subsystem is available. Without a suitable knowledge-based subsystem, these null values require some join operations in order to be detected; even with the support of a knowledge-based subsystem, the database accesses cannot be dispensed with.

Null values of type 2.2 require some database accesses and some join operations, in order to be detected. Neither the database accesses, nor the join operations can be dispensed with, whether or not a knowledge-based subsystem is available.

We explain of the classification in detail and include some examples.

4.1. Null events that can be detected without database access

Class 1 of our classification contains those null events that can be predicted or detected without access to the database. Since no database access is involved, these null events are all of type Ξ. An Ξ-type null event reveals some mismatch between the words (recognized as objects) of the natural language input and the objects of the database. Depending on the point of view taken, this mismatch can be attributed to a misconception on the side of the user who tries to relate two or more objects in some inapplicable way, or to an imperfect database representation of the micro-world of interest, or both. Since this distinction can only be seen from outside the self-contained database system, it does not result in a further subclassification. We do however subclassify according to whether or not a knowledge-based subsystem is required in order to detect or predict the null events at this stage of query processing, that is, prior to any access to the database.

4.1.1. Null events that can be detected without support by the knowledge-based subsystem

Some of the null events are taken care of by the semantic interpreter (see Figure 1-2) which uses the lexicon to establish the proper cross reference between the words of the NL-query and the formal objects represented by the database. A failure of this process results in a null event. The semantic interpreter can then generate some appropriate message to the user. The corresponding null events form subclass 1.1 of our classification.

Example:

- Q: *What is the address of student Mary Lou ?*

The semantic interpreter fails to find any matching keyword to the input '*address*' in the lexicon. Therefore '*address*' cannot be mapped into any entity or property represented in the database.

The user obviously expected an aspect of his conception of the real world to be represented in the database, but this was not the case. From the user's point of view, the null event is caused by some inapplicability with respect to the imperfect database representation of the micro-world. In our database students do not have addresses. However, from the database system's point of view the query is just as inapplicable as the query '*What is the address of the fall 87 semester ?*'. The user would see this latter query as inapplicable not only with respect to the database representation, but also with respect to the real world, since a semester doesn't have an address.

A human observer would probably see the object '*address*' in the above queries as a missing property of the entity types STUDENT and SEMESTER, respectively. In the next example the missing object is more likely to be seen as an entity.

Example:

- Q: *Who are the classmates of John Smith ?*

The semantic interpreter fails to find any matching keyword to the input '*classmate*' in the lexicon. Therefore '*classmate*' cannot be mapped into any entity or property represented in the database.

Again the user expected an aspect of his conception of the real world to be represented in the database, but this was not the case.

Although the use of a knowledge-based subsystem is not crucial in predicting this class of null events, it can have other beneficial effects. As we show in Section 5.1 (where the concept of a CLASSMATE is added to the system via a simple rule in the knowledge-base), the knowledge-based subsystem can sometimes help to eliminate the problems with null events by expanding the linguistic coverage of the system.

4.1.2. Null events that cannot be detected without support by the knowledge-based subsystem

Subclass 1.2 of the class of null events that can be detected prior to a database access consists of those null events that require the knowledge-based subsystem which makes use of the information represented in the RM/T* catalog, in order to be detected at this stage of query processing. The knowledge-base inspects the logical form of the query, and determines whether or not any inapplicability with respect to the database described by the extended catalog can be detected. If some inapplicability is detected, the knowledge-based subsystem generates an appropriate response to the user, otherwise the query processing goes on to the next 'normal' stage.

The feasibility of this approach varies drastically, depending on the particular case, as the following two examples will show. For the first example assume that the database contains a MAIDEN_NAME attribute for entities of type STUDENT. It is a trivial question when addressed to a human listener whose native language is English.

Example:

- Q: *What is the maiden name of student John Smith ?*

  Common sense tells us that this query will result in a null event even though the database contains information about maiden names since maiden names only make sense for female persons. We do know this fact and we are also capable of deducing from his name the fact that John Smith must be a male person. If the same knowledge and capabilities were implemented in the knowledge-based subsystem, then this null event could be predicted without database access. However, a reliable algorithm to determine the sex of a person, given the persons name, would probably require more time to execute than can be saved by avoiding the database access.

  A solution using null values of type '*property inapplicable*' and no assistance of a knowledge-based subsystem would require a database access before the null event could be detected. A normalization of the database such that male and female persons are distinct entity types might save some space in memory, but would also require a database access before the null event could be detected, since only after such a database access could the sex of the instance specified via the name '*John Smith*' be determined.

The above example is one that a human listener could easily cope with, but which could hardly be handled efficiently by a knowledge-based system. The next example could be solved by a knowledge-based system, that has access to the catalog and that includes some appropriate rules.

Example:

- Q: *What are the minor fields of the students who got A grades in last semester's CMPT 810 course ?*

  Assuming that '*CMPT 810*' was in fact offered in the last semester, and that — as presumed by the query — some students did get 'A' grades in it, this query still turns out to be inapplicable, because the respective students must all be a graduate students (that is, instances of entity type GRAD): *CMPT 810* refers to a graduate course and only graduate students are allowed to take graduate courses. The attributes MAJOR and MINOR are applicable to entities of type UNDER_GRAD only.

  Null values of type '*property inapplicable*' do not help in this case. Not only is the attribute MINOR inapplicable to the specified students, it is — as a sensible consequent — not represented at all. From the database point of view this query is similar to the query '*What is the address of the fall 87 semester ?*', of Section , with the difference that now the semantic interpreter is no longer capable of detecting the inapplicability.

  A detection of this null event without prior access to the database requires the implementation of additional knowledge along with associated capabilities, namely
  - the knowledge that only graduate students can take graduate courses, as well as
  - some means to detect and differentiate between undergraduate and graduate courses by their number. This is feasible since the course number allows a direct determination of whether the respective course is an ungraduate course or a graduate course.

  An approach based on normalization would divide the entity type COURSE into two new types, one for undergraduate courses and one for graduate courses. However, to detect the null value prior to any database access, this approach would still require the knowledge-based subsystem. It would also require the additional representation of the fact that graduate students can still take undergraduate courses, but not vice versa. In short, normalization seems to be an inappropriate approach in this case.

4.2. Null events that can be detected only after some database access

Class 2, the second major class of null events consists of those whose prediction or detection requires some database access. In contrast with the previous class, we can now expect both Ξ-type, and Ω-type null events to occur.

A Ξ-type null event reveals some mismatch between the words (recognized as objects) of the natural language input and the objects of the database. Depending on the point of view, this mismatch can be attributed to a misconception on the side of the user who tries to relate two or more objects in some inapplicable way, or to an imperfect database representation of the micro-world of interest, or on both.

In case of an Ω-type null event the requested information is not present in the database: an applicable entity or a property value is missing. If a property value is missing, a tuple can be retrieved, but the property attribute of interest contains an ω-type null value. If an entity is missing, no matching surrogate exists in the appropriate E-relation, that is, a whole tuple is missing.

Whether or not either type of null event, Ω or Ξ, occurs, depends on the structure of the database and the particular state the database is in. Depending on the complexity of the query relative to the structure of the database, join operations might, or might not be necessary before a null event can be predicted or detected. We further subclassify according to these two alternatives.

4.2.1. Null events that can be detected without the need of join operations on the database

Subclass **2.1** contains those null events, whose detection requires no join operations to be performed in the database. Again both Ξ-type, and Ω-type null events can be expected. We further subclassify according to whether or not the knowledge-based subsystem is required in order to detect the null event at this stage of query processing.

4.2.1.1. Null events that can be detected without support by the knowledge-based subsystem

Null events whose detection requires some database access, but neither requires join operations to be performed, nor the assistance of the knowledge-based subsystem in order to be detected, form class **2.1.1**, which is at the lowest level of our classification. Again both Ξ-type, and Ω-type null events can be expected.

One particular query might result in either an Ω-type null event, or a Ξ-type null event of this class, depending on the current state of the database.

Example:

- Q: *Is John Smith in a PhD program ?*

  Suppose '*John Smith*' refers to an identifiable instance of type GRAD, and furthermore, that the respective tuple in the GRAD_PROPERTIES relation contains an ω-type null value in its PROGRAM attribute. This would constitute a typical case of an Ω-type null event.

  On the other hand, if '*John Smith*' is not represented at all in the database, that is, neither as STUDENT nor as STAFF, nor as AUTHOR, we would again obtain an Ω-type null event because of missing information. This time not the attribute value, but a whole tuple is missing. The difference should be visible in the distinct responses generated for the distinct cases.

  For yet another result suppose '*John Smith*' refers to an identifiable instance of type UNDER_GRAD. This would constitute a typical case of a Ξ-type null event, since the attribute PROGRAM is inapplicable to entities of type UNDER_GRAD.

  In all the above cases only a search through the database was necessary, and the assistance of a special knowledge-based subsystem might not be required: The natural language interface with its access to the lexicon (see Figure 1-1) might be enough.

The Ξ-type null event is a borderline case with respect to our classification. Normally the query evaluator (see Figure 1-2) would search through the STUDENT_PROPERTIES relation and find a surrogate value for the instance referred by *'John Smith'*. However, a search for the same surrogate in the GRAD relation, or the GRAD_PROPERTIES relation would fail. A 'dumb' query evaluator might interpret this as an Ω-type null event, a 'smart' one might detect a possible misconception.

In any case a knowledge-based subsystem could at least be helpful in generating a cooperative response to the user, such as *'John Smith is not a graduate student and therefore cannot be in any program'*, or preferably *'John Smith is an undergraduate student and therefore cannot be in any program'*. Although the entity type STUDENT is categorized in only two subentity types, the latter choice is particularly informative since *John Smith* could also be a GRAD student, or a STAFF member, or an AUTHOR. In general, we cannot require a user of a natural language front-end to specify the appropriate database relation in order to eliminate this ambiguity. The generation of a (highly) cooperative response would then require some processing which was not requested by the query, namely a determination of the subtype *'John Smith'* belongs to.

In the next example, a knowledge-based subsystem would again be helpful to detect the null event at the present stage of query processing, but does not represent the only possible solution.

Example:

- Q: *What grade did John Smith get in CMPT 100 this semester ?*

Suppose that at the time the query is entered to the system, the specified course is still in progress and the grades could not yet be computed. Thus, the query would be *inapplicable* and should result in a Ξ-type null event. The knowledge-based subsystem could predict this after a mere search through the database by comparing the FINAL attribute value of the specified course with the CURRENT_DATE, and give an appropriate response to the user. A cooperative response should also inform the user about the value of the applicable FINAL. If the FINAL attribute of the corresponding tuple contains an ω-type null value, the null event will be considered as being of type Ω instead of Ξ.

An approach using null values of type *'property inapplicable'* in the GRADE attribute would require an automatic update of those values to ω-type null values in case no manual update has been performed by the appropriate due date. This automatic update would again require the assistance of a knowledge-based subsystem in some form or another. However, the detection of the Ξ-type null event would then not require the assistance of the knowledge-based subsystem and therefore the null event under consideration belongs to the current class in our classification.

We conclude our elaborations on this class of null events with a final example:

- Q: *What are the minor fields of the students who got A grades in all courses taught by Newton last semester ?*

Suppose *Newton* was on sabbatical leave last semester and therefore didn't teach at all. Since our database does not represent information on whether or not an instructor is (or was) on sabbatical leave, an initial search revealing the Ω-type null event cannot be avoided.

4.2.1.2. Null events that can be detected only with support by the knowledge-based subsystem

Null events that can be detected after a database access and with the assistance of the knowledge-based subsystem only, form subclass 2.1.2 of the class of null events that can be detected without the need of any join operations. In Section we gave an extensive list of examples in order to show the possible advantages of a knowledge-based subsystem in that class. Some of those examples involved borderline cases already with respect to the need of a knowledge-based subsystem. In the present class the advantages are obvious; a single example will suffice to illustrate this.

- Q: *What are the minor fields of the students who got A grades in all courses taught by Newton last semester ?*

  Suppose one of the courses *Newton* taught last semester, was a graduate course. Then any student who got A grades in all those courses must be a graduate student to whom the attribute MINOR is not applicable. In order to detect this null event before any join operations are performed, the knowledge-based subsystem can direct the query evaluation in such a way, that initially only the respective course numbers are determined. As soon as the information about the course numbers is available, the knowledge-based subsystem can detect the null event and generate an informative response to the user.

4.2.2. Null events that can be detected only after some join operations have been performed on the database

Some null events can be detected only after an access to the database and some join operations on the data have been performed. Those null values belong to subclass 2.2 of our classification. Since in this class we release all restrictions on searches and join operations, the null event will eventually be detected with or without the support of a knowledge-based subsystem.

Although it would still be possible to distinguish between the stages of query processing at which a null event could or could not be detected with or without the support of a knowledge-based subsystem, we do not further subclassify this class.

In this subclass again, both Ξ-type and Ω-type null events can be expected. As before, the Ξ-type null events reflect some mismatch between the elements of the natural language query and the corresponding objects of the database, whereas the Ω-type null events reflect some current lack of information within the database which otherwise appropriately represents the relevant aspect of the real world.

The following is a typical example of an Ω-type null event. It can be detected only after searches and join operations in the database.

Example:

- Q: *What are the minor fields of the students who got A grades in last semester's CMPT 410 and CMPT 411 courses ?*

  Assuming that both *CMPT 410* and *CMPT 411* were in fact offered in the last semester, and that — as presumed by the query — some students did get *A* grades in both courses, extensive searches and join operations on the database might still result in Ω-type null events, namely in case the the minor fields of the respective students are not recorded in the database, that is, substituted by ω-type null values.

  A cooperative response clarifying the user's misconception should also be generated in case the set of students who got *A* grades in both courses is empty. In this case the null event would also be of type Ω, but distinct from the previous one. The distinction should be apparent in the different responses generated.

The following is a typical example of a Ξ-type null event, where the user suggests with his query to relate some property to an entity, to which it is not applicable at the time the query is entered.

Example:

- Q: *What are the minor fields of the students who got A grades in last semester's CMPT 410 and CMPT 411 courses ?*

  Assuming again that both *CMPT 410* and *CMPT 411* were in fact offered in the last semester, and that — as presumed by the query — some students did get *A* grades in both courses, query processing might still result in Ξ-type null events following extensive searches and join operations on the database, namely in case some of the respective students have graduated in the mean time. The support of a knowledge-based subsystem is the most appropriate approach to handle these null events, that is, to detect the possibility of a user misconception and to generate an informative response. However, the searches and relational operations on the database to find all applicable students in the first place cannot be avoided.

In this chapter we have clarified the important distinction between null events of type '*value at present unknown*' (Ω-type null events) and null events of type '*property inapplicable*' (Ξ-type null events). Based on this distinction, we have shown that the representation of null values of type '*property inapplicable*' (ξ-type null values) reflects a poor database design and can in general be seen as the manifestation of misconceptions on the part of the database designer. We have then presented an identification and classification of null events, which is primarily based on minimizing the amount of database access required to detect the null events. In Chapter 5 we present an approach which puts the ideas developed so far into practice: We investigate on the required representation of additional information about the intensional contents of the database, and outline a way to process the queries such that null events are detected with a minimized amount of database access.

Chapter 5
Null event detection in the RM/T* model

In this chapter we describe the knowledge-based subsystem in action. The goal is to determine the existence of an Ξ-type null event with as little database access as possible. The main idea embodied in the knowledge-based subsystem is to extract information from the catalog, specifically information about the domains of the attributes involved in each particular query.

From the catalog we stepwise retrieve available information about the domains of the attributes involved during query processing. The retrieval sometimes activates constraints that are then added to the current environment of processing. The constraints are formally specified in Section 5.1. For the use of the algorithms to be outlined, the corresponding rules are listed and indexed in the table EXCEPTION_RULES in the knowledge-base. Activation of a rule occurs, whenever a relational attribute is accessed in the extended CATLG_ATTRIBUTES relation, whose corresponding tuple contains a reference to an exception rule. The respective rule is added to the current environment. An environment extends over all attributes of a relation and over all the attributes of those relations that are connected to the relation via a join operator. All constraints that are added to a specific environment, propagate to all attributes involved in that environment.

We assume the query is stated in relational calculus which will be introduced briefly in Section 5.2. In Section 5.3 we describe the processing of the query. Rather than giving the actual program for all functions and procedures, we give an outline of the algorithms used and the pseudo code for some important functions. In Section 5.4 we finally give a detailed example of the processing of a query which results in null events.

5.1. Additional integrity constraints

In this Section we formally specify additional integrity constraints and other rules, that apply to the micro-world modelled in our database. Most of the integrity constraints are represented as rules in the table EXCEPTION_RULES in the knowledge-base and used for the detection of null events of type 'property inapplicable'.

We use the following notation for the formal specification:

- REL_NAM (t) declares t to be a tuple of relation REL_NAM
- t[ATTR_NAM] specifies the value of attribute ATTR_NAM in the tuple t
- '*dom_val*' denotes an arbitrary domain_value
- θ denotes any one of the comparison operators $<, \leq, =, \geq, >$, and $\neq$

```
(∀ q) (∀ r) (∀ s) (∀ t) {   ENROLLMENT_GRADE (q)
                        ∧   ENROLLMENT_INSTANCE (r)
                        ∧   CLASS_INSTANCE (s)
                        ∧   TODAY (t)

                        ∧   r[CLASS¢]  =  s[CLASS¢]
                        ∧   s[FINAL]  ≥  t[DATE]

                        →   ( (q[GRADE]  θ  dom_val) ≡ Ξ )  }
```

Informally this means that unless the final date of a class has passed, the assignment of any grade is inapplicable.

```
(∀ q) (∀ r) (∀ s) {   COURSE_PROPERTIES (q)
                  ∧   CLASS_INSTANCE (r)
                  ∧   ENROLLMENT_INSTANCE (s)

                  ∧   q[NUMBER]  ≥  800
                  ∧   q[COURSE¢]  =  r[COURSE¢]
                  ∧   r[CLASS¢  =  s[CLASS¢]

                  →   { (∃ t) (     GRAD (t)
                              ∧     t[GRAD¢]  =  s[STUDENT¢] )} }
```

Informally this means that students enrolled in 800- and higher level courses must be graduate students.

$$
\begin{aligned}
(\forall q)(\forall r)(\forall s)(\forall t)\{\quad & \text{ENROLLMENT\_GRADE}(q)\\
\wedge\quad & \text{ENROLLMENT\_INSTANCE}(r)\\
\wedge\quad & \text{CLASS\_INSTANCE}(s)\\
\wedge\quad & \text{TODAY}(t)\\
\wedge\quad & q[\text{ENROLLMENT¢}] = r[\text{ENROLLMENT¢}]\\
\wedge\quad & r[\text{CLASS¢}] = s[\text{CLASS¢}]\\
\rightarrow\quad & s[\text{FINAL}] < t[\text{DAY}]\ \}
\end{aligned}
$$

Informally this means that the final date of a class for which the grades are recorded already, must be in the past.

$$
\begin{aligned}
(\forall s)\{\quad & \text{ROOM\_OFFICE}(s)\\
\rightarrow\quad & \neg\{(\exists t)(\ \text{SCHEDULE\_INSTANCE}(t)\\
& \qquad \wedge\ t[\text{ROOM¢}] = s[\text{ROOM¢}]\ ))\}\ \}
\end{aligned}
$$

Informally this means that a room which is used as an office, cannot be scheduled for a class.

$$
\begin{aligned}
(\forall s)\{\quad & \text{ROOM\_PHONE}(s)\\
\rightarrow\quad & \neg\{(\exists t)(\ \text{SCHEDULE\_INSTANCE}(t)\\
& \qquad \wedge\ t[\text{ROOM¢}] = s[\text{ROOM¢}]\ ))\}\ \}
\end{aligned}
$$

Informally this means that a room which has a phone (and is thus used as an office), cannot be scheduled for a class.

$$
\begin{aligned}
(\forall s)\{\quad & \text{SCHEDULE\_INSTANCE}(s)\\
\rightarrow\quad & \neg\{(\exists t)(\ \text{ROOM\_OFFICE}(t)\\
& \qquad \wedge\ t[\text{ROOM¢}] = s[\text{ROOM¢}]\ ))\}\ \}
\end{aligned}
$$

Informally this means that a room in which a class is scheduled, cannot be an office.

$$
\begin{aligned}
(\forall s)\{\quad & \text{SCHEDULE\_INSTANCE}(s)\\
\rightarrow\quad & \neg\{(\exists t)(\ \text{ROOM\_PHONE}(t)\\
& \qquad \wedge\ t[\text{ROOM¢}] = s[\text{ROOM¢}]\ ))\}\ \}
\end{aligned}
$$

Informally this means that a room in which a class is scheduled, does not have a phone in it.

All the above integrity constraints are reflected in the Table EXCEPTION_RULES in the knowledge-base (see Appendix D). For example the first one of the integrity constraints above corresponds to the exception rule '(1)'. This rule is activated whenever the relation CLASS_PROPERTY or the relation TODAY appears in a query (see Appendix C.7). The Table EXCEPTION_RULES also contains rules that correspond to none of the above integrity constraint, but to some of those constraints imposed by the RM/T* model in general. For example, exception rule '(4)' corresponds to the mutex contraint imposed on the subentities GRAD and UNDER_GRAD.

We now present two more rules which are not directly related to the detection of null events and for which an actual implementation is not shown.

$$
\begin{array}{rl}
(\forall q)(\forall r)(\forall s)(\forall t)\ \{ & \text{ENROLLMENT\_GRADE}(q) \\
\wedge & \text{ENROLLMENT\_INSTANCE}(r) \\
\wedge & \text{CLASS\_INSTANCE}(s) \\
\wedge & \text{TODAY}(t) \\
 & \\
\wedge & r[\text{CLASS¢}] = s[\text{CLASS¢}] \\
\wedge & s[\text{FINAL}] \leq t[\text{DATE}] \\
\wedge & q[\text{GRADE}] = \text{`}\omega\text{'} \\
 & \\
\rightarrow & q[\text{GRADE}] = \text{`D'}\ \}
\end{array}
$$

Informally this means that if an enrollment exists for some course, and the final has passed, and a corresponding grade is not available (that is, substituted by an ω-type null value), we may deduce the value 'D' (deferred) for the respective grade.

$$
\begin{array}{rl}
(\forall r)(\forall s)(\forall t)\ \{ & \text{ENROLLMENT\_INSTANCE}(r) \\
\wedge & \text{ENROLLMENT\_INSTANCE}(s) \\
\wedge & \text{CLASS\_INSTANCE}(t) \\
 & \\
\wedge & r[\text{STUDENT¢}] \neq s[\text{STUDENT¢}] \\
\wedge & r[\text{CLASS¢}] = s[\text{CLASS¢}] \\
\wedge & r[\text{CLASS¢}] = t[\text{CLASS¢}] \\
 & \\
\rightarrow & \text{`}\mathit{Classmate\_In\_Course}\text{'}(r[\text{STUDENT¢}], s[\text{STUDENT¢}], t[\text{COURSE¢}])\ \}
\end{array}
$$

Informally this rule introduces the concept of a classmate: any two distinct students who are enrolled in the same class, are considered to be classmates in the respective course. This concept could now be added to the lexicon of the database.

5.2. Relational Operations

In this Section we present the relational operations, expressed in relational calculus, and give an example for each. The five basic operations that serve to define the relational algebra are union, set difference, cartesian product, projection, and selection. Further operations like intersection, quotient, and join, can be expressed in terms of these five basic operations. We use the following notational conventions:

- the greek letter θ denotes one of the binary relations $<, \leq, =, \geq, >, \neq$.
- a string of capital letters denotes a relation with ordered attributes
- a string of lower case letters denotes a name of an attribute
- superscripts denote arities
- subscripts denote attribute names
- lower case italic letters denote tuples
- $R(t)$ denotes that $(t \in R)$

- t[i] denotes the value of the ith attribute of tuple t
- t[ATT_NAME] denotes the value of attribute ATT_NAME in tuple t

For all operations, the domains of the corresponding attributes or constants must be subsets (not necessarily proper subsets) of the same domain.

Union: $Q^n \cup T^n =$

$\{ t \mid Q(t) \vee T(t) \}$

Q and T must have the same arity for the union operation to make sense. The resulting relation has then the same arity as Q and T.

Example: $\text{GRAD} = \text{GRAD}_{\text{GRAD¢}}$

$\text{UNDER\_GRAD} = \text{UNDER\_GRAD}_{\text{UNDER\_GRAD¢}}$

$\text{GRAD} \cup \text{UNDER\_GRAD} =$

$\{ t \mid \text{GRAD}(t) \vee \text{UNDER\_GRAD}(t) \}$

The result of the operation is a unary relation containing all surrogates that identify either entities of type GRAD, or entities of type UNDER_GRAD. Since the category STATUS divides the super-type STUDENT in precisely the two sub-types GRAD and UNDER_GRAD, and STATUS spans the super-type, the resulting relation is equal to the E-relation STUDENT.

Set difference: $Q^n - T^n =$

$\{ t \mid Q(t) \wedge \neg T(t) \}$

Example: $\text{STUDENT} = \text{STUDENT}_{\text{STUDENT¢}}$

$\text{GRAD} = \text{GRAD}_{\text{GRAD¢}}$

$\text{STUDENT} - \text{GRAD} =$

$\{ t \mid \text{STUDENT}(t) \wedge \neg \text{GRAD}(t) \}$

The result of this operation is a unary relation containing all surrogates, that identify entities of type STUDENT, but not entities of type GRAD. Since the category STATUS divides the super-type STUDENT in precisely the two sub-types GRAD and UNDER_GRAD, and STATUS spans the super-type, the resulting relation is equal to the E-relation UNDER_GRAD.

***Cartesian product*:** $Q^m \times T^n =$

$$\{ t^{m+n} \mid (\exists e^m)(\exists l^n)(Q(e) \wedge T(l) \wedge t[1] = e[1] \wedge, .., \wedge t[m] = e[m] \wedge \wedge t[m+1] = l[1] \wedge, .., \wedge t[m+n] = l[n])\}$$

Example: $\text{ROOM\_PROPERTIES} = \text{ROOM\_PROPERTIES}_{\text{ROOM¢, BUILDING, NUMBER}}$

$\text{TIME\_TABLE\_PROPERTIES} = \text{TIME\_TABLE\_PROPERTIES}_{\text{TIME\_TABLE¢, DAY, HOUR}}$

$\text{ROOM\_PROPERTIES} \times \text{TIME\_TABLE\_PROPERTIES} =$

$\{ t^6 \mid (\exists e^3)(\exists l^3)(\text{ROOM\_PROPERTIES}(e) \wedge \text{TIME\_TABLE\_PROPERTIES}(l)$

$\wedge\ t[\text{ROOM¢}] = e[\text{ROOM¢}]$

$\wedge\ t[\text{BUILDING}] = e[\text{BUILDING}]$

$\wedge\ t[\text{NUMBER}] = e[\text{NUMBER}]$

$\wedge\ t[\text{TIME\_TABLE¢}] = l[\text{TIME\_TABLE¢}]$

$\wedge\ t[\text{DAY}] = l[\text{DAY}]$

$\wedge\ t[\text{HOUR}] = l[\text{HOUR}]\ \}$

The result of this operation is a relation with six attributes, the first three are identical to the attributes of relation ROOM_PROPERTIES, the last three are identical to the attributes of relation TIME_TABLE_PROPERTIES. The contents of the relation can be seen as a list of tables, one for each room (including offices), listing all the possible hours at which a lecture could be scheduled.

***Projection*:** $\pi_{q_i, .., q_j}(Q_{q_1, .., q_n})$, where $\{q_i, .., q_j\} \subseteq \{q_1, .., q_n\}$

$= \{ t^m \mid (\exists e)(Q(e) \wedge t[q_i] = e[q_i] \wedge, .., \wedge t[q_j] = e[q_j]) \}$, where $|\{q_i, .., q_j\}| = m$

Example: $\text{STUDENT\_PROPERTIES} = \text{STUDENT\_PROPERTIES}_{\text{STUDENT¢, NUMBER, NAME, SEX}}$

$\pi_{\text{NUMBER, NAME}}(\text{STUDENT\_PROPERTIES}) =$

$\{ t^2 \mid (\exists e)(\text{STUDENT\_PROPERTIES}(e) \wedge t[\text{NUMBER}] = e[\text{NUMBER}] \wedge t[\text{NAME}] = e[\text{NAME}]) \}$

The result of this operation is a binary relation with student NUMBERs and their NAMEs for all entities of type STUDENT.

Selection: $\sigma_{q_i \theta q_j} (Q_{q_1, \ldots, q_n})$, where $q_i \in \{q_1, \ldots, q_n\}$; $q_j \in \{q_1, \ldots, q_n\}$

$= \{ t \mid Q(t) \wedge (t[q_i] \; \theta \; t[q_j]) \}$

or $\sigma_{q_i \theta C} (Q_{q_1, \ldots, q_n})$, where $q_i \in \{q_1, \ldots, q_n\}$; C is a constant.

$= \{ t \mid Q(t) \wedge (t[q_i] \; \theta \; C) \}$,

The operands of operator θ are either both attributes of relation Q, or one operand is an attribute of relation Q and the other operand is a constant.

Example: ROOM_PROPERTIES = ROOM_PROPERTIES$_{\text{ROOM¢, BUILDING, NUMBER}}$

$\sigma_{\text{BUILDING} = \text{'MPX'}}$ (ROOM_PROPERTIES) =

{ t | ROOM_PROPERTIES(t) $\wedge$ t[BUILDING] = 'MPX' }

The result of this operation is a ternary relation with identical attributes as relation ROOM_PROPERTIES. All tuples in this relation have the value 'MPX' in their BUILDING attribute; the numbers in the NUMBER attribute are those of all the rooms in the 'MPX' building.

The following operations are useful during query processing, but could also be expressed in terms of the five basic operations:

Intersection: $Q^n \cap T^n$ =

$Q - (Q - T)$ =

$\{ t \mid Q(t) \wedge T(t) \}$

Example: STUDENT = STUDENT$_{\text{STUDENT¢}}$

GRAD = GRAD$_{\text{GRAD¢}}$

STUDENT $\cap$ GRAD =

{ t | STUDENT(t) $\wedge$ GRAD(t) }

The result of this operation is a unary relation containing all surrogates that identify entities of type STUDENT and simultaneously identify entities of type GRAD. Since the category STATUS divides the super-type STUDENT in precisely the two sub-types GRAD and UNDER_GRAD, and STATUS spans the super-type, the resulting relation is equal to the E-relation GRAD.

Quotient: let $Q = Q_{q_1, ..., q_n}$; $T = T_{t_1, ..., t_m}$; $\{t_1, ..., t_m\} \subset \{q_1, ..., q_n\}$; then

$Q \div T$ =

$\pi_{q_i, ..., q_j}(Q) - \pi_{q_i, ..., q_j}(((\pi_{q_i, ..., q_j}(Q)) \times T) - Q)$, where $\{q_i, ..., q_j\} = \{q_1, ..., q_n\} - \{t_1, ..., t_m\}$

$= \{ t^{n-m} \mid (\exists e^m)(Q(te) \wedge T(e)) \}$

Example: $\text{TEXT\_INSTANCE} = \text{TEXT\_INSTANCE}_{\text{TEXT¢, COURSE¢, BOOK¢}}$

$\text{BOOK} = \text{BOOK}_{\text{BOOK¢}}$

$\text{TEXT\_INSTANCE} \div \text{BOOK}$ =

$\{ t_{\text{TEXT¢, COURSE¢}} \mid (\exists e_{\text{BOOK¢}}) (\text{TEXT\_INSTANCE}(te) \wedge \text{BOOK}(e)) \}$

The result of this operation is a binary relation with the two E-attributes TEXT¢ and COURSE¢; the E-attribute COURSE¢ contains only surrogates identifying courses, that use all available books as their textbooks.

Join: let $Q = Q_{q_1, ..., q_n}$; $T = T_{t_1, ..., t_n}$; $q_i \in \{q_1, ..., q_n\}$; $t_j \in \{t_1, ..., t_n\}$ then

$Q\,[q_i\, \theta\, t_j]\, T$ =

$\sigma_{q_i \theta t_j}(Q \times T)$

Example: $\text{DEPARTMENT\_PROPERTIES} = \text{DEPARTMENT\_PROPERTIES}_{\text{DEPARTMENT¢, CHAIR¢, NAME, FACLTY}}$

$\text{STAFF\_PROPERTIES} = \text{STAFF\_PROPERTIES}_{\text{STAFF¢, DEPT¢, NUMBER, NAME, SEX}}$

DEPARTMENT_PROPERTIES [CHAIR¢ = STAFF¢] STAFF_PROPERTIES is a relation with nine attributes: the first four are identical to the attributes of the relation DEPARTMENT_PROPERTIES, the last five are identical to the attributes of the relation STAFF_PROPERTIES. Each tuple has identical values in its CHAIR¢ and its STAFF¢ attribute, that is, the relation lists the departments together with their respective chairpersons.

Natural Join: $R \bowtie S$

A natural join is a a special kind of join operation: It is equivalent to a sequence of join operations with the operand θ being equality ('=') and applied to all attributes that appear in both relations R and S, followed by a removal of the redundant columns generated by the join operations.

None of these four additional operations can be expressed by the five basic operations without use of the set difference or the selection operation. Therefore all four operations can cause Ξ-type null events to happen, no matter how the database is structured. In the next section we present some additional catalog relations and a set of procedures which will allow us to detect the null events as early as possible. We assume a database normalized such as the one presented in this thesis, that is, tuples containing inapplicable attributes are not represented.

5.3. Detection of Ξ-type null events

In this section we outline the processing of the query stated in relational calculus as described in Section 5.2. The goal is to detect Ξ-type null events with a minimum amount of database access. To do this, we determine the set of domain values of each attribute that appears in a particular query and reduce each set as much as possible, but such that the reduced set is still guaranteed to contain all domain values that could possibly be present in that attribute field in the current state of the database. Whenever we determine that one of those reduced sets is empty, we have detected an Ξ-type null event, since the respective attribute field can then not contain any applicable value, and the query cannot result in any tuple being retrieved. As we have shown in Chapter 4, this might be possible without any database access, or only after some database access.

We assume that the query is stated in relational calculus, using the formulae presented in Section 5.2. For each relation occurring in a particular query, we first determine the set of attributes participating in that relation. For each of those attributes we create a list, referred to as ADOM-list. Thus a relation with *n* attributes will result in a set of *n* ADOM-lists. Each ADOM-list consists of four elements referred to as FLAG, ATT_NAME, DOMAIN, and ENVIRONMENT:

- FLAG: a list containing one of the three characters 'N', 'C', and 'D'.
- ATT_NAME: a list containing a full attribute name, that is, a relation name, followed by a period ('.'), followed by an attribute name.
- DOMAIN: a list, which is interpreted according to the character in the FLAG:
    - 'N': DOMAIN contains the name of the domain, as specified in the CATLG_ATTRIBUTES relation.
    - 'C': DOMAIN contains a set of domain values as obtained from the catalog of the database and possibly reduced by additional constraints of the applicable environment.
    - 'D': DOMAINS contains a set of domain values obtained from the database, that is, after a database retrieval of the corresponding relation and possibly reduced by additional constraints of the applicable environment.
- ENVIRONMENT: a list containing an integer value, identifying the environment which is applicable to the corresponding attribute.

During retrieval of the information necessary to built up these lists, we also activate the set of exception rules applicable to the current environment. An environment in this context is a set of attributes which is subject to the same set of exception rules. All attributes of a base relation belong to the same environment. An exception rule, which is activated in the current environment need not have any effect. Only if all antecedents of the respective formula are part of the current environment and true in that environment, and if at least one of the consequents is part of the current environment, does the particular rule 'fire'. This concept will become clearer in the example given in Section 5.4.

The next step is to apply functions to these lists, corresponding to the relational operators of the relational interpretation of the query. The functions transform the list elements in a way compatible to the relational interpretation of the query and the applicable exception rules. The functions corresponding to the selection and to the join operators cause the affected environments to be expanded: All attributes of two relations that are connected via a selection or join operator belong to the same environment; thus the set of exception rules activated in one relation can propagate to another relation. Other operators do not cause the set of exception rules to propagate from one relation to another, that is, the respective environments remain distinct. However, functions corresponding to the other relational operators can also cause a reduction of the set of domain values of the affected attributes. For example, the function corresponding to the set difference operator, can cause a reduction of the set of domain values, as we show later i this Section.

An empty set resulting from the application of the functions for any attribute domain is equivalent to the detection of a null event: The result guarantees, that under no circumstances will we find any value in the corresponding attribute field. Thus not a single tuple will satisfy the constraints imposed by the query in the current state of the database, or in any state, if no database access was required to produce the empty set.

If no empty set is produced, we start retrieving relations from the database and carry out further operations, based on the sets of domain values, which are then additionally restricted according to the current contents of the database.

We now describe the representation of the required information and some of the functions retrieving and manipulating the information. As mentioned in Chapter 3, in RM/T* we extend the catalog relation CALTG_ATTRIBUTES by one further attribute field: 'EXC'. This attribute field contains integer values identifying indexed exception rules which are listed in table EXCEPTION_RULES (see Appendix C.7 and Appendix D.1). The value '0' acts as a dummy, in case no exception rule applies. The table EXCEPTION_RULES is a list of lists. Each list element corresponds to one exception rule. The structure of each exception rule is also a list: The first element is the integer value identifying the rule; the second element is a formula, the third element is a comment. For example, exception rule '(2)' has the three elements

- (2)
- ((COURSE_PROPERTIES.NUMBER $\geq$ 800) $\rightarrow$
  ((UNDER_GRAD.UNDER_GRAD¢ := $\emptyset$)
  $\wedge$ ((UNDER_MAJOR.UNDER_GRAD¢ := $\emptyset$)
  $\wedge$ ((UNDER_MINOR.UNDER_GRAD¢ := $\emptyset$)))
- (*You specified some higher level course(s); undergraduate students cannot be enrolled in such courses.*)

Not all formulae of exception rules have antecedents: sometimes the mere fact that a specific relation participates in a query results in applicable constraints. For example, exception rule '(4)' corresponds to the mutex integrity constraint and has the three elements

- (4)
- ((UNDER_GRAD.UNDER_GRAD¢ := $\emptyset$)
  $\wedge$ ((UNDER_MAJOR.UNDER_GRAD¢ := $\emptyset$)
  $\wedge$ ((UNDER_MINOR.UNDER_GRAD¢ := $\emptyset$))
- (*You specified some graduate student(s); graduate students are distinct from undergraduate students; 'major' and 'minor' are inapplicable to them.*)

For each relation appearing in the query, we determine the list of attributes corresponding to that query, and add all exception rules identified by those attributes to the current environment. This task is done by function GET_ATTRIBUTES:

Function **GET_ATTRIBUTES** takes as input the two values 'RELNAM', specifying the name of a relation and 'E', identifying the current environment. GET_ATTRIBUTES returns a set of *n* ADOM-lists corresponding to the *n* attibutes of relation RELNAM. Function GET_ATTRIBUTES also adds all applicable exception rules to the current environment.

```
Get all N tuples from catalog relation CATLG_ATTRIBUTES,
    which have the value RELNAM in their first attribute field.
For each of the N tuples create an ADOM-list as follows:
    FLAG := 'N'
    ATT_NAME := RELNAM.ATTNAM
    DOMAIN := DOMNAME
    ENVIRONMENT := E
    add the value of attribute EXC to the set of rules
       activated in the current environment E
Return the N ADOM-lists.
```

We now describe the functions corresponding to the relational operators described in Section 5.2. We assume that the ADOM-lists of the respective relations are available to these functions. Each function first makes sure, that the corresponding attributes have the same domain name, since only then are they compatible. Attributes whose domains are subsets of the same domain are either E-attributes with the common domain name '¢', or P-attributes which do not differ in their domain names, but only in their

EL_KEY or INTRV_KEY attributes in the ATTRIBUTE_DOMAINS relation (see Appendix C.3). If the domain names of two attributes that need to be matched by a relational operator are distinct, then the set of domain values for both resulting attributes is the empty set.

The function corresponding to the ***union*** operator augments the set of domain values for each pair of attributes such that the set of domain values for both domains consists of the union of the two individual sets. These resulting sets are stored in the DOMAIN-list of the ADOM-lists corresponding to the affected attributes.

For example, applied to the attributes GRAD.GRAD¢ and STUDENT.STUDENT¢, the function corresponding to the union operator returns the set denoted 'STUDENT¢', which represents all surrogates identifying entities of type STUDENT. Applied to the attributes AREA_NAME.NAME and DEPARTMENT_PROPERTIES.NAME, the same function returns the set denoted FIELD, which contains the set of domain values identifying the different fields of study in the university.

The function corresponding to the ***set-difference*** operation Q - T depends on the FLAGs of the ADOM-lists corresponding to T. If the FLAG has the value 'D', the corresponding DOMAIN contains the set of values actually found in the database. In this case DOMAIN(Q) is reduced to
$\{e \mid ((e \in \text{DOMAIN(Q)}) \wedge (e \notin \text{DOMAIN(T)}))\}$. Otherwise DOMAIN(Q) remains unaltered. There are two exceptional cases to consider:

- If Q and T are of arity one, then DOMAIN(Q) is reduced to
  $\{e \mid ((e \in \text{DOMAIN(Q)}) \wedge (e \notin \text{DOMAIN(T)}))\}$.
- For Q - Q, the function returns the empty set for the DOMAINs of each pair of ADOM-lists.

The function corresponding to the ***cartesian product*** operation $Q^m \times T^n$ and the function corresponding to the ***projection*** operation do not cause any restrictions on the affected domains.

The function corresponding to the ***selection*** operation $\sigma_{q_i \theta q_j}(Q^n_{q_1,\dots,q_n})$ reduces the DOMAINs corresponding to q_i and q_j such that

$\text{DOMAIN}(q_i) = \{ e \mid (e \in \text{DOMAIN}(q_i)) \wedge ((\exists\, l \in \text{DOMAIN}(q_j))\ (e\ \theta\ l)) \}$ and

$\text{DOMAIN}(q_j) = \{ e \mid (e \in \text{DOMAIN}(q_j)) \wedge ((\exists\, l \in \text{DOMAIN}(q_i))\ (l\ \theta\ e)) \}$.

For $\sigma_{q_i \theta C}(Q^n_{q_1,\dots,q_n})$ the DOMAIN corresponding to q_i is reduced such that

$\text{DOMAIN}(q_i) = \{e \mid (e \in \text{DOMAIN}(q_i)) \wedge (e\ \theta\ C)\})\}$

The function corresponding to the ***intersection*** operation $Q \cap T$ reduces the DOMAINs such that for each pair of attributes

DOMAIN(q_i) := $\{ e \mid (e \in \text{DOMAIN}(q_i)) \wedge (e \in \text{DOMAIN}(t_i)) \}$

DOMAIN(t_i) := $\{ e \mid (e \in \text{DOMAIN}(q_i)) \wedge (e \in \text{DOMAIN}(t_i)) \}$

The function corresponding to the ***quotient*** operation $Q^{\,n} \div T^{\,m}$ with the respective attributes $\{t_1, .., t_m\} \in \{q_1, .., q_n\}$ leaves the DOMAINs corresponding to the attributes q_i, where $q_i \notin \{t_1, .., t_m\}$, untouched. The remaining DOMAINs get reduced only if the DOMAINs of T consists of those values actually retrieved from the database, that is, the corresponding FLAG is set to 'D'. If this is the case, we obtain:

if $((\exists\ (i \leq j \leq k))\ (\text{DOMAIN}(q_j) \subset \text{DOMAIN}(t_j)))$

then $(\forall\ (i \leq l \leq k)\ (\text{DOMAIN}(q_l) := \emptyset))$

else no reduction occurs.

The function corresponding to the ***join*** operation $Q\ [q_i\ \theta\ t_j]\ T$ reduces the DOMAINs of the ADOM-lists corresponding to q_i and t_j such that

DOMAIN(q_i) := $\{e \mid (e \in \text{DOMAINS}(q_i)) \wedge ((\exists\, l \in \text{DOMAINS}(t_j))\ (e\ \theta\ l)) \}$ and

DOMAIN(t_j) := $\{e \mid (e \in \text{DOMAIN}(t_j)) \wedge ((\exists\, l \in \text{DOMAIN}(q_i))\ (l\ \theta\ e)) \}$

The functions assume a database normalized in such a way as described in Chapter 4, that is, tuples containing inapplicable attributes are not represented. Consider a user-query which simply requires the retrieval of a complete relation, such as the relation TIME_TABLE_PROPERTIES, corresponding to the query '*List all time slots of the week, at which a class could be scheduled*'. In this case all tuples currently represented in that relation will be retrieved. A proper database design guarantees that none of those tuples represents a misconception of some sort or another.

As a result of our proper database design, Ξ-type null events can occur only, when the query, stated in relational calculus, includes operators which impose constraints on the tuples to be retrieved, since only those constraints could result in empty sets, reflecting the Ξ-type null event. Out of the five primitive operators, union, set-difference, cartesian product, projection and selection, only two, set-difference and selection, impose constraints on the tuples to be retrieved. Thus every query which uses only the other three operators is guaranteed to cause no Ξ-type null events. The operators intersection, quotient, and join require set-

difference and selection operators in order to be expressed with only primitive operators. As a result they also may cause Ξ-type null events.

5.4. Example

In this section we illustrate our solution with a sample query. The sample query chosen results in a null event which will be predicted prior to any access to the extensional data. All information required for the detection is represented in the catalog of the RM/T* model and in the EXCEPTION_RULES which apply to the specific micro-world modelled in our database. The processing of the query begins at the innermost level of the corresponding formula stated in relational calculus. During query processing a number of exception rules are activated; one of those rules eventually 'fires' and produces an empty set as the domain for two attributes. The empty sets correspond to the detection of a null event. The comment part of the specific rule that fired and produced the empy set is then used for the response generation for the query.

Our example is based on the following query:

- *'What are the minor fields of the students who got A grades in CMPT 810 ?'*

Expressed in relational calculus, this query reads:

$$
\begin{array}{l}
\pi_{\text{MINOR}} \\
\quad \{\text{UNDER\_MINOR [UNDERGRAD¢ = STUDENT¢]} \\
\quad \{ \quad \{\text{ENROLLMENT\_INSTANCE [CLASS¢ = CLASS¢]} \\
\qquad\quad \{\text{CLASS\_INSTANCE [COURSE¢ = COURSE¢]} \\
\qquad\quad \{\sigma_{\text{NUMBER = '810'}} \\
\qquad\qquad \{\text{COURSE\_PROPERTIES [FIELD¢ = AREA¢]} \\
\qquad\qquad\quad (\sigma_{\text{NAME = 'CMPT'}}(\text{AREA\_NAME}))\}\}\}\} \\
\qquad\quad \text{[ENROLLMENT¢ = ENROLLMENT¢]} \\
\qquad\quad \{\sigma_{\text{GRADE = 'A'}}(\text{ENROLLMENT\_GRADE})\}\}\}
\end{array}
$$

Starting the evaluation of the formula at the innermost level, for the relation

(AREA_NAME)

we inspect the catalog relation CATLG_ATTRIBUTES (see Appendix C.7) and obtain two ADOM-lists:

[(N) (AREA_NAME.AREA¢) (¢) (1)]
[(N) (AREA_NAME.NAME) (FIELD) (1)]
E(1) = ∅.

E(1) is the current environment and contains the set of rules that are active in this environment. At this stage E(1) is empty.

For the selection operation

($\sigma_{\text{NAME} = \text{'CMPT'}}$(AREA_NAME))

we access the catalog relation ATTRIBUTE_DOMAINS (see Appendix C.3) to determine the set of domain-values corresponding to AREA_NAME.NAME; this set is equal to the complete domain FIELD. We make sure that 'CMPT' is a member of FIELD (see Appendix C.2) and obtain:

[(N) (AREA_NAME.AREA¢) (¢) (1)]
[(C) AREA_NAME.NAME) ({'CMPT'}) (1)]
E(1) = Ø.

Adding the join operation

{COURSE_PROPERTIES [FIELD¢ = AREA¢]
($\sigma_{\text{NAME} = \text{'CMPT'}}$(AREA_NAME))}

we obtain:

[(N) (COURSE_PROPERTIES.COURSE¢) (¢) (1)]
[(C) (COURSE_PROPERTIES.FIELD¢) (AREA¢) (1)]
[(N) (COURSE_PROPERTIES.NUMBER) (cours_num) (1)] *
[(N) (COURSE_PROPERTIES.UNITS) (unit_num) (1)]
[(C) (AREA_NAME.AREA¢) (AREA¢) (1)]
[(C) AREA_NAME.NAME) ({'CMPT'}) (1)]
E(1) = {2}.

An exception rule has been activated by attribute COURSE_PROPERTIES.NUMBER (marked with a '*'). It is the exception rule number 2, as specified in the catalog relation CATLG_ATTRIBUTES (see Appendix C.7). The exception rule is specified in the table of EXCEPTION_RULES in the knowledge-base (see Appendix D.1). Due to the join operation, two ADOM-lists are identical; both denote the set of surrogates identifying instances of entities of type AREA. This information is obtained as follows: from the catalog relation CATLG_ATTRIBUTES we know that AREA_NAME.AREA¢ is the primary key of the relation. Thus the entity type described is AREA. The attribute COURSE_PROPERTIES.FIELD¢ is not the primary key of the relation. In this case we need some further information. Catalog relation CATLG_RELATIONS (see Appendix C.6) tells us that COURSE_PROPERTIES contains a designation. Relation DESIG_GRAPH (see Appendix C.11) tells us that the designated entity type is AREA.

For

{$\sigma_{\text{NUMBER} = \text{'810'}}$
{COURSE_PROPERTIES [FIELD¢ = AREA¢]
($\sigma_{\text{NAME} = \text{'CMPT'}}$(AREA_NAME))}}

we determine the set of domain-values of attribute COURSE_PROPERTIES.NUMBER, which corresponds to the entire domain 'cours_num' (see relation ATTRIBUTE_DOMAINS, Appendix C.3). We make sure that '810' is a member of 'cours_num' and replace the set of domain-values by the specified value. We obtain:

[(N) (COURSE_PROPERTIES.COURSE¢) (¢) (1)]
[(C) (COURSE_PROPERTIES.AREA¢) (AREA¢) (1)]

[(C) (COURSE_PROPERTIES.NUMBER) ({'810'}) (1)]
[(N) (COURSE_PROPERTIES.UNITS) (unit_num) (1)]
[(C) (AREA_NAME.AREA¢) (AREA¢) (1)]
[(C) (AREA_NAME.NAME) ({'CMPT'}) (1)]
E(1) = {2}.

For

{CLASS_INSTANCE [COURSE¢ = COURSE¢]
{$\sigma_{NUMBER = '810'}$
{COURSE_PROPERTIES [FIELD¢ = AREA¢]
($\sigma_{NAME = 'CMPT'}$(AREA_NAME))}}}

we obtain:

[(N) (CLASS_INSTANCE.CLASS¢) (¢) (1)]
[(N) (CLASS_INSTANCE.INSTRUCTOR¢) (¢) (1)]
[(C) (CLASS_INSTANCE.COURSE¢) (COURSE¢) (1)]
[(N) (CLASS_INSTANCE.SEMESTER¢) (¢) (1)]
[(N) (CLASS_INSTANCE.FINAL) (day_num) (1)] *
[(C) (COURSE_PROPERTIES.COURSE¢) (COURSE¢) (1)]
[(C) (COURSE_PROPERTIES.AREA¢) (AREA¢) (1)]
[(C) (COURSE_PROPERTIES.NUMBER) ({'810'}) (1)]
[(N) (COURSE_PROPERTIES.UNITS) (unit_num) (1)]
[(C) (AREA_NAME.AREA¢) (AREA¢) (1)]
[(C) (AREA_NAME.NAME) ({'CMPT'}) (1)]
E(1) = {2,1}

The attribute CLASS_INSTANCE.FINAL (marked with a '*') has activated another exception rule. It is the rule number 1, specified in the table of EXCEPTION_RULES in the knowledge-base. Due to the join operation, the DOMAINs of two more ADOM-lists are identical; both denote the set of surrogates identifying instances of entities of type COURSE.

For

{ENROLLMENT_INSTANCE [CLASS¢ = CLASS¢]
{CLASS_INSTANCE [COURSE¢ = COURSE¢]
{$\sigma_{NUMBER = '810'}$
{COURSE_PROPERTIES [FIELD¢ = AREA¢]
($\sigma_{NAME = 'CMPT'}$(AREA_NAME))}}}}

we obtain:

[(N) (ENROLLMENT_INSTANCE.ENROLLMENT¢) (¢) (1)]
[(N) (ENROLLMENT_INSTANCE.STUDENT¢) (¢) (1)]
[(C) (ENROLLMENT_INSTANCE.CLASS¢) (CLASS¢) (1)]
[(C) (CLASS_INSTANCE.CLASS¢) (CLASS¢) (1)]
[(N) (CLASS_INSTANCE.INSTRUCTOR¢) (¢) (1)]
[(C) (CLASS_INSTANCE.COURSE¢) (COURSE¢) (1)]
[(N) (CLASS_INSTANCE.SEMESTER¢) (¢) (1)]
[(N) (CLASS_INSTANCE.FINAL) (day_num) (1)]
[(C) (COURSE_PROPERTIES.COURSE¢) (COURSE¢) (1)]
[(C) (COURSE_PROPERTIES.FIELD¢) (AREA¢) (1)]

[(C) (COURSE_PROPERTIES.NUMBER) ({'810'}) (1)]
[(N) (COURSE_PROPERTIES.UNITS) (unit_num) (1)]
[(C) (AREA_NAME.AREA¢) (AREA¢) (1)]
[(C) (AREA_NAME.NAME) ({'CMPT'}) (1)]
E(1) = {2,1}.

Before performing the step corresponding to the next join operation, we have to evaluate the relation to be joined. We start again at the innermost level and in a new environment, E(2):

For

(ENROLLMENT_GRADE)

we obtain:

[(N) (ENROLLMENT_GRADE.ENROLLMENT¢) (¢ (2)]
[(N) (ENROLLMENT_GRADE.GRADE) (GRADE) (2)] *
E(2) = {3}.

Another exception rule has been activated here. It is rule number 3, specified in the table of EXCEPTION_RULES in the knowledge-base and is valid in the current environment E(2).

For

{$\sigma_{\text{GRADE} = \text{'A'}}$(ENROLLMENT_GRADE}}

we determine the set corresponding to ENROLLMENT_GRADE.GRADE, which is the entire set of domain values of GRADE. We then make sure that 'A' is a member of the set GRADE. We obtain:

[(N) (ENROLLMENT_GRADE.ENROLLMENT¢) (¢) (2)]
[(C) (ENROLLMENT_GRADE.GRADE) ({'A'}) (2)]
E(2) = {3}.

We now perform the operation corresponding to the join: For

```
{      {ENROLLMENT_INSTANCE [CLASS¢ = CLASS¢]
       {CLASS_INSTANCE [COURSE¢ = COURSE¢]
       {σ_NUMBER = '810'
              {COURSE_PROPERTIES [FIELD¢ = AREA¢]
                     (σ_NAME = 'CMPT'(AREA_NAME))}}}}
       [ENROLLMENT¢ = ENROLLMENT¢]
       {σ_GRADE = 'A'(ENROLLMENT_GRADE}}
```

we obtain:

[(C) (ENROLLMENT_INSTANCE.ENROLLMENT¢) (ENROLLMENT¢) (1)]
[(N) (ENROLLMENT_INSTANCE.STUDENT¢) (¢) (1)]
[(C) (ENROLLMENT_INSTANCE.CLASS¢) (CLASS¢) (1)]
[(C) (CLASS_INSTANCE.CLASS¢) (CLASS¢) (1)]
[(N) (CLASS_INSTANCE.INSTRUCTOR¢) (¢) (1)]
[(C) (CLASS_INSTANCE.COURSE¢) (COURSE¢) (1)]
[(N) (CLASS_INSTANCE.SEMESTER¢) (¢) (1)]

[(N) (CLASS_INSTANCE.FINAL) (day_num) (1)]
[(C) (COURSE_PROPERTIES.COURSE¢) (COURSE¢) (1)]
[(C) (COURSE_PROPERTIES.FIELD¢) (AREA¢) (1)]
[(C) (COURSE_PROPERTIES.NUMBER) ({'810'}) (1)]
[(N) (COURSE_PROPERTIES.UNITS) (unit_num) (1)]
[(C) (AREA_NAME.AREA¢) (AREA¢) (1)]
[(C) (AREA_NAME.NAME) ({'CMPT'}) (1)]
[(C) (ENROLLMENT_GRADE.ENROLLMENT¢) (ENROLLMENT¢) (2)]
[(C) (ENROLLMENT_GRADE.GRADE) ({'A'}) (2)]
E(1) = E(2) = {2,1,3}.

The last join operation has caused the two environments E(1) and E(2) to become identical: they both include their current union.

For
{UNDER_MINOR [UNDERGRAD¢ = STUDENT¢]
{ {ENROLLMENT_INSTANCE [CLASS¢ = CLASS¢]
{CLASS_INSTANCE [COURSE¢ = COURSE¢]
{$\sigma_{\text{NUMBER = '810'}}$
{COURSE_PROPERTIES [FIELD¢ = AREA¢]
($\sigma_{\text{NAME = 'CMPT'}}$(AREA_NAME))}}}}
[ENROLLMENT¢ = ENROLLMENT¢]
{$\sigma_{\text{GRADE = 'A'}}$(ENROLLMENT_GRADE}}}

we obtain:

[(N) (UNDER_MINOR.UNDER_MVP2¢) (¢) (1)] *
[(C) (UNDER_MINOR.UNDER_GRAD¢) (∅) (1)]
[(N) (UNDER_MINOR.MINOR) (FIELD) (1)]
[(C) (ENROLLMENT_INSTANCE.ENROLLMENT¢) (ENROLLMENT¢) (1)]
[(C) (ENROLLMENT_INSTANCE.STUDENT¢) (∅) (1)]
[(C) (ENROLLMENT_INSTANCE.CLASS¢) (CLASS¢) (1)]
[(C) (CLASS_INSTANCE.CLASS¢) (CLASS¢) (1)]
[(N) (CLASS_INSTANCE.INSTRUCTOR¢) (¢) (1)]
[(C) (CLASS_INSTANCE.COURSE¢) (COURSE¢) (1)]
[(N) (CLASS_INSTANCE.SEMESTER¢) (¢) (1)]
[(N) (CLASS_INSTANCE.FINAL) (day_num) (1)]
[(C) (COURSE_PROPERTIES.COURSE¢) (COURSE¢) (1)]
[(C) (COURSE_PROPERTIES.FIELD¢) (AREA¢) (1)]
[(C) (COURSE_PROPERTIES.NUMBER) ({'810'}) (1)]
[(N) (COURSE_PROPERTIES.UNITS) (unit_num) (1)]
[(C) (AREA_NAME.AREA¢) (AREA¢) (1)]
[(C) (AREA_NAME.NAME) ({'CMPT'}) (1)]
[(C) (ENROLLMENT_GRADE.ENROLLMENT¢) (ENROLLMENT¢) (2)]
[(C) (ENROLLMENT_GRADE.GRADE) ({'A'}) (2)]
E(1) = E(2) = {2,1,3,7}.

The operation corresponding to the last join operation, together with exception rule 2, has produced an empty set, which in turn corresponds to an Ξ-type null event. We abort the processing of the entire formula representing the query. No database access was required. For the response generation we supply the comments associated with all the rules that were involved in the derivation of the empty set. In this case there is just one such rule and one comment: *You specified some higher level course(s); undergraduate students cannot be enrolled in such courses.*

In this chapter we have outlined an algorithm to process user queries, stated in relational calculus, in such a way that null events will be detected with a minimized amount of database access. The amount of database access is a good parameter to evaluate the cost of a query, since even in sophisticated natural language-database systems, most of the time from query entry to response presentation is spend for database accesses (see [Kaplan 79]).

Chapter 6
A four-valued logic to deal with two types of null events

We introduce a formal method for handling the two different types of null events introduced in Chapter 4, namely the Ω-type null event caused by some missing piece of information (*value at present unknown*) and the Ξ-type null event caused by some type mismatch (*property inapplicable*). The title of this chapter indicates that we use the term 'logic' to refer to this method. As we will show in Section 6.4, this term could be considered inadequate, nevertheless 'logic' is used because of its widespread use in this context. Our approach extends Codd's notion of a three valued logic to deal with Ω-type null events [Codd 79] to a fourth truth value to deal with Ξ-type null events. In fact, the present Chapter is a digression on Codd's proposals of a three-valued logic and a four-valued logic [Codd 87]. The four-valued logic we introduce does not constitute a relational reconstruction of the algorithms of Chapter 5 used to detect Ξ-type null events.

The relevant aspects of the standard two-valued logic and of two different three-valued logics will be presented first. Those aspects are then gradually extended to a four-valued logic, which allows a combined treatment of both Ω-type and Ξ-type null events.

As we have seen in Chapter 5, relational calculus is based on first order predicate calculus and thus a kind of first order logic is used during query processing and answer generation (see, for example, [Ullman 82] [16] or [Reiter 84]). Under the closed world assumption, a two-valued logic is appropriate for a relational database without null values. The corresponding truth tables for negation, conjunction, and disjunction are illustrated in Table 6-1

[16] **Chapter 5: The Relational Model**

NOT(T)	=	F
NOT(F)	=	T

AND	F	T
F	F	F
T	F	T

OR	F	T
F	F	T
T	T	T

Table 6-1: Truth tables for two-valued negation, conjunction and disjunction

Thus, for conjunctions the truth value F dominates over the truth value T, whereas for disjunctions the truth value T dominates over the truth value F. Basically the closed world assumption, together with the assumption that no null values exist in the database, guarantees that the *law of excluded middle* (an object must either have a predicate or its negation) holds. Under these assumptions, such queries as

- *Are the students Mary and John both PhD students ?* or
- *Is Mary or John a PhD student ?*

can always be answered properly using the two-valued logic, since the PROGRAM attribute for each student represented in the database, contains some nonnull value. Problems arise, however, in more conventional cases that allow the occurrence of null values.

6.1. The *value at present unknown* type of null event

In [Codd 79], Codd suggests the use of a three-valued logic in order to deal with the *value at present unknown* type of null values and gives the corresponding truth tables for conjunction, disjunction, and denial. He uses the symbol 'ω' to denote two distinct kinds of unknown information, namely the null events and the null values for which we introduced the two distinct terms Ω and ω in Chapter 4.

Codd ([Codd 79]) [17] justifies his twofold use of 'ω' by stating:

"We use the same symbol 'ω' to denote the unknown truth value, because truth values can be stored in databases and we want the treatment of all unknown or null values to be uniform."

A proper distinction between null values and null events is important for rigorous approaches to the problem with null events in general. We adapt the notation introduced in Chapter 4:

1. 'ω' substitutes a missing and therefore currently unknown value in some attribute field in the database.
2. 'Ω' denotes the unknown truth value of a statement about the current state of the database.

The truth tables shown in Table 6-2 are the ones given in [Codd 79], except for the symbol 'ω' which

[17] In later publications ([Codd 86], [Codd 87]) Codd makes the distinction clearer by using the logical truth value 'MAYBE', and a 'mark' to record the fact that a database value is missing.

NOT (*T*)	=	*F*
NOT (*F*)	=	*T*
NOT (Ω)	=	Ω

AND	*F*	Ω	*T*
F	*F*	*F*	*F*
Ω	*F*	Ω	Ω
T	*F*	Ω	*T*

OR	*F*	Ω	*T*
F	*F*	Ω	*T*
Ω	Ω	Ω	*T*
T	*T*	*T*	*T*

Table 6-2: Truth tables for three-valued negation, conjunction and disjunction, as given by Lukasiewicz and Codd

is now replaced by 'Ω'. These truth tables are also identical with the ones given by Lukasiewicz (except for the notation) when he first introduced his three-valued logic in 1920 in order to deal with propositions such as "*I shall be in Warsaw at noon on 21 December of next year*". According to Lukasiewicz such propositions are, at the moment they are considered, neither true nor false and must possess a third value, different from falsity and truth (see [Rescher 69]). This truth value corresponds to our *unknown* truth value Ω, used to signal a state of partial ignorance (the truth value is *at present unknown*), not a state in which neither *T*rue, nor *F*alse are applicable.

As the truth tables in Table 6-2 show, the truth value Ω dominates over the truth value *T* in conjunctions, and dominates over the truth value *F* in disjunctions. The obvious reasoning behind this is the fact that

- In the case of a conjunction the truth value *F* can be established as soon as one component has truth value *F*, whereas the truth value *T* can only be established when all components have truth value *T*.
- In case of a disjunction the truth value *T* can be established as soon as one component has truth value *T*, whereas the truth value *F* can only be established when all components have truth value *F*.

Consider, for example, the STUDENT_PROPERTIES relation of our database (see Appendix B.26) and assume that two tuples have the values '87000-0001' and '87000-0002' in their respective NUMBER attributes and the values 'F' and 'ω' in their respective SEX attributes. We obtain the following question - answer pairs: [18]

- *Are the two students with numbers 87000-0001 and 87000-0002 both female ?* — Ω
- *Is one of the two students with numbers 87000-0001 and 87000-0002 female ?* — *T*

In case the first of the above two tuples contains the value 'M' instead of 'F' in its SEX attribute, and the rest remains unchanged, we obtain:

- *Are the two students with numbers 87000-0001 and 87000-0002 both female ?* — *F*
- *Is one of the two students with numbers 87000-0001 and 87000-0002 female ?* — Ω

Reconsidering our primary objective to eliminate wasteful operations and null events, we can see that Ω-type

[18] The question - answer pairs given as examples in this chapter do not include the complete answers a knowledge-based system should provide, but a simplified form only.

null events cannot be predicted by inspection of the catalog of the database or other preprocessing the knowledge-based subsystem might perform. A search through the extensional data is necessary before the ω-type null value can be found, and the ω-type null value has to be found in the database before the *unknown* truth value 'Ω' can be determined.

6.1.1. Some problems with the third truth-value

Some problems arise when a decision has to be made as to whether or not to include elements in an answer list when the attribute used as a selection criteria for a tuple contains the ω-type null value. Two examples of a corresponding query are

- *List all female students* and
- *List all male students.*

The question is whether those students whose SEX in not known at the time of query processing should be included in either one or both of the answers to the above queries. The most appropriate solution seems to include them in both cases, together with an explanatory comment.

However, such comments might seem superfluous or even annoying in cases such as

- *List all students which are female or male* and
- *List all students which are female and male.*

The first of these two queries should preferably result in a list of *all* students, the second in an empty list together with the additional comment that in the micro-world represented by this database all students are bound to be either female or male, but cannot be both at the same time. This latter case would constitute one, in which dispensable operations could be eliminated even though no null event was bound to occur.

The above case is not trivial in the database world, even though it seems obvious to the human user. The cause of this phenomenon is the fact, that in the given three-valued logic $(P \lor \neg P)$ need not be *T*rue and $(P \land \neg P)$ need not be *F*alse, namely in case $P = \Omega$.

The above examples are chosen for illustrative purposes only and might seem somewhat contrived. More realistic examples would involve attributes that accept a larger, yet limited number of distinct values instead of only two, like Female and Male. For example, the course PRE_REQuisites of any COURSE offered in a university is a proper subset of all COURSEs offered.

Here the ω-type null value acts as a special additional value and similar counterintuitive results have to be expected if the given three-valued logic is to be applied directly in some specific cases. In general, if

V_τ is the value of a specific attribute of tuple τ, and n is the number of possible distinct values for that attribute, then

$\{ \ [V_\tau = V_1] \ \vee \ [V_\tau = V_2] \ \vee \ \ldots \ \vee \ [V_\tau = V_n] \ \}$ need not be *T*rue, and

$\{ \ [V_\tau = V_a] \ \wedge \ [V_\tau = V_b] \ \}$, with $a \neq b$, need not be *F*alse,

in case V_τ = 'ω'; in this case both of the above propositions have the truth value 'Ω'.

A feasible solution to this problem of generating appropriate responses to all composite queries, where the components are connected via the logical AND or OR operator, requires query preprocessing, which detects precisely the two cases specified above and acts accordingly.

Notice, that even in a database without null values, the database system might not be able to infer that the truth values of the above propositions must be *T*rue. To do so the system needs to know the complete set of values that make up the domain of the respective attribute. In databases designed according to the RM/T* model, the required information is available via the extended catalog, unless it is not specifiable in principle (if the domain of the respective attribute is not enumerable).

We have elaborated on some peculiarities arising from the fact that the law of excluded middle does not hold in the three valued logic introduced, and based on the three connectives conjunction, disjunction, and negation. These are the only connectives that Codd defines in his papers [Codd 79], [Codd 86], [Codd 87]; he gives no definition for implication.

6.1.2. Implication and incomplete information

An appropriate definition of implication for a multi-valued logic is not a trivial problem. In the standard two valued logic, negation reverses the truth value. Here the substitution of $(A \rightarrow B)$ by $(\neg A \vee B)$ causes no problems: If $(A \rightarrow B)$ holds, all combinations for the values of A and B are permissible, except the value pair ($\langle A, B \rangle = \langle$ *T*rue, *F*alse $\rangle$) and we obtain the truth table illustrated in Table 6-3.

A B	*F*	*T*
F	*T*	*T*
T	*F*	*T*

Table 6-3: Truth table for two-valued implication $A \rightarrow B$

Table 6-3 is identical with the truth table for $(\neg A \vee B)$. This interpretation of implication is usually referred to as '*material implication*'.

In the three-valued logic negation does not necessarily reverse the truth value since the law of excluded middle does not hold ($\neg\,\Omega = \Omega$). If we use the analogue of material implication in the three valued logic as specified by Codd, we end up with the left table of the two truth tables illustrated in Table 6-4.

A B	*F*	Ω	*T*
F	*T*	*T*	*T*
Ω	Ω	Ω	*T*
T	*F*	Ω	*T*

A B	*F*	Ω	*T*
F	*T*	*T*	*T*
Ω	Ω	*T*	*T*
T	*F*	Ω	*T*

Table 6-4: Truth tables for three-valued implication $A \rightarrow B$ as given by Kleene (left) and by Lukasiewicz (right)

Now the reflexive law ($P \rightarrow P$) need not hold any more, namely if ($P = \Omega$). This seems counter-intuitive: no matter what truth value proposition P may have, we would expect ($P \rightarrow P$) to *always* be true.

In order to ensure that ($P \rightarrow P$) is tautologous, Lukasiewicz changed the corresponding entry to *T*rue in his system, as shown in the right truth table for implication in Table 6-4. In fact, Lukasiewicz chose negation and this definition of implication as the primitives of this system (see [Rescher 69]), and derived the other two connectives OR and AND:

- $A \vee B$ is given by $(A \rightarrow B) \rightarrow B$, and
- $A \wedge B$ is given by $\neg(\neg A \vee \neg B)$.

Lukasiewicz' solution is not problem-free either: we obtain the counter-intuitive result (($P \rightarrow \neg P$) = *T*rue) if ($P = \Omega$).

Kleene argued (see [Rescher 69]) that the truth value of ($A \vee B$) is defined only if either both A and B are known to be false (in which case the resulting truth value is *F*alse), or either one of A and B is known to be true (in which case the resulting truth value is *T*rue). In the latter case nothing is said about the other variable. Consequently he did not adopt the change made by Lukasiewicz and used the left truth table in Table 6-4 to define implication for his 'strong' connectives (see [Rescher 69]).

For the purpose of information retrieval from a database, Kleene's version is more appropriate. Since all deductions are based on propositions whose truth values are either *T*rue or *F*alse, all the implication ($A \rightarrow B$) tells us is that the truth value *T*rue propagates from proposition A to proposition B. Nothing is said about the propagation of truth-value *F*alse. If the truth-value *F*alse was also supposed to propagate from proposition A to proposition B, we would write A = B. Thus in Table 6-4 for implication we obtain the truth value

- *True* whenever A = *F* (since then B could be anything) and whenever B = *T* (since then A could be anything),
- *False* whenever ((A = *T*) ∧ (B = *F*)) (since in this case the truth value *T* has not propagated from A to B).

If we base our interpretation of Table 6-4 on the more intuitive notion of entailment, we find the truth value

- *True* whenever the current state of the database is consistent with the proposition 'A entails B',
- *False* whenever the current state of the database is inconsistent with the proposition 'A entails B',
- Ω whenever the current state of the database does not allow either one of the above conclusions.

Notice that the extensional database does not contain enough information to allow us to conclude that an entailment holds among the data. Even the inspection of the complete set of states which the extensional data in the database can adopt would not allow the conclusion of such rules. Given a pair of values for A and B, one can conclude whether or not (¬ A ∨ B) holds *for this pair.* Given a complete set of value pairs that A and B can possibly adopt, one can conclude whether or not (¬ A ∨ B) *always* holds. But such a set includes no information about whether or not B is *True* only due to A, that is, whether or not A is *relevant* for B. Therefore it is impossible to derive any rule governing the interrelationship between the entities described by the data without observing this interrelationship in the first place. Such a rule must be explicitely specified in the knowledge-base, or via integrity constraints.

Directly related to this problem is the problem of interpreting ambiguous queries. Consider the query

- *Does every graduate student have a supervisor ?*

which can be interpreted in two different ways:

1. *Does every graduate student currently represented in the database have a supervisor, who is also represented ?*
2. *Does in the micro-world represented by the database being a graduate student entail having a supervisor in general ?*

The first interpretation aims at the extensional aspect of the micro-world being represented; it corresponds to the logic form (¬ A ∨ B). This interpretation is probably correct, if it is entered by some department administrator wishing to know whether he has to assing a supervisor to some new grad student. Based on this interpretation and the values of the extensional data, a '*Yes*' answer is possible.

The second interpretation aims at the intensional aspect of the micro-world being represented; it corresponds to the logic form (A |— B) (that is A entails B). This is probably the correct interpretation if the query is entered by a new student at the university. Based on this interpretation and no more than the values of the extensional data, a '*Yes*' answer to the query is impossible.

There exist a number of other proposals for a definition of implication in many-valued logics. An extensive overview is given in [Rescher 69]. In the Section 6.2 we introduce another type of null event and an associated fourth truth value; we also present the corresponding truth tables for negation, conjunction, disjunction and our version of implication.

6.2. The *property inapplicable* type of null event

We adapt the notation introduced in Chapter 4 and distinguish between the database value 'ξ' and the truth value 'Ξ' as follows:

1. ξ substitutes an inapplicable value in some attribute field in the database.
2. Ξ denotes a fourth possible truth value of a statement about the current state of the database. It is distinct from truth, falsity, and ignorance, and represents the truth value of a proposition that is malformed with respect to the intensional content of the database.

We obtain a the truth value Ξ whenever the elements of a query cannot be matched with the structure of the database. In general this happens without the existence of a corresponding ξ-type null value among the extensional data, unless some misconceptions are actually represented in the database. As we have shown in Chapter 5, Ξ type null events are predictable by careful inspection of the database catalog and with the use of some additional knowledge. However, Lukasiewicz's three-valued logic is not applicable to this kind of null events, as we show in this section.

Our database differentiates between students and instructors (academics) by using distinct entities, The attribute PROG is present for some students (namely GRAD students), but not for instructors. With only one third truth value (Ω), we would end up with question — answer pairs such as

- *Is instructor Galilei or instructor Kepler a PhD student ?* — Ω

This statement makes sense if we change the interpretation of the truth value Ω to that of the truth value Ξ. However we could get the same answer for a different question too:

- *Is instructor Galilei or student Mary a PhD student ?* — Ω

Now the interpretation of the answer is not clear. Furthermore, in case of the question — answer pair

- *Are instructor Galilei and student Mary PhD students ?* — *F*

Mary must be a Master's or Special student and cannot be a PhD student, but by generating the above answer Lukasiewicz's three-valued logic would hide the underlying misconception in the query. In order to deal with Ξ-type null events, we need to introduce a different kind of logic.

We disregard the Ω-type null events for a moment, and concentrate on the Ξ-type null events. An appropriate set of connectives to deal with Ξ-type null events could be based on the three-valued logic presented by D. A. Bochvar in 1939. Here the third truth value stands for something like '*paradoxical*' or '*meaningless*' (see [Rescher 69]), which is analogous to the *property inapplicable* interpretation of null events. This truth value, which we denote with 'Ξ', is assigned to any compound which has at least one component with that value. Thus, the presence of the third truth value among any connectives *infects* the entire formula with meaninglessness and the user will be informed about the existence of any detectable misconception, no matter where in his query such a misconception is manifested. The corresponding truth tables for negation, conjunction, and disjunction are illustrated in Table 6-5.

NOT(*T*)	= *F*
NOT(*F*)	= *T*
NOT(Ξ)	= Ξ

AND	Ξ	*F*	*T*
Ξ	Ξ	Ξ	Ξ
F	Ξ	*F*	*F*
T	Ξ	*F*	*T*

OR	*F*	*T*	Ξ
F	*F*	*T*	Ξ
T	*T*	*T*	Ξ
Ξ	Ξ	Ξ	Ξ

Table 6-5: Truth tables for three-valued negation, conjunction and disjunction, as given by Bochvar

For implication we use the interpretation introduced in Section 6.1.2 and obtain for (A → B):
- *T*rue whenever (A = *F*alse) or (B = *T*rue)
- *F*alse whenever ((A = *T*rue) ∧ (B = *F*alse))
- Ω in all other cases

The last of these three cases requires further explanation. Assume the implication (A → B) is given. If A's value is Ξ (*inapplicable*), B's value might, but need not, be so too. We cannot assume that the implication (A → B) always interrelates two equally in/applicable concepts, that is, the implication need not be inapplicable just because the antecedent or the consequent is inapplicable. Consider the following example: Assume PREGNANT(X) → ON_LEAVE(X). Now, if MALE(X), then PREGNANT(X) = Ξ, but ON_LEAVE(X) ≠ Ξ.

In this case there is not enough information to confirm or refute the implication (PREGNANT(X) → ON_LEAVE(X)), or to establish its inapplicability. Thus, we need to make use of the truth value Ω and obtain the truth table for implication illustrated in Table 6-6.

A B	*F*	Ξ	*T*
F	*T*	*T*	*T*
Ξ	Ω	Ω	*T*
T	*F*	Ω	*T*

Table 6-6: Truth table for implication A → B in the presence of misconceptions

6.3. The four-valued logic

We now combine the two distinct three-valued logics presented and obtain a four-valued logic which is suitable to deal with both Ω-type and Ξ-type null events. The corresponding truth tables for negation, conjunction, and disjunction are illustrated in Table 6-7.

NOT (*T*)	=	*F*
NOT (*F*)	=	*T*
NOT (Ω)	=	Ω
NOT (Ξ)	=	Ξ

AND	Ξ	*F*	Ω	*T*
Ξ	Ξ	Ξ	Ξ	Ξ
F	Ξ	*F*	*F*	*F*
Ω	Ξ	*F*	Ω	Ω
T	Ξ	*F*	Ω	*T*

OR	*F*	Ω	*T*	Ξ
F	*F*	Ω	*T*	Ξ
Ω	Ω	Ω	*T*	Ξ
T	*T*	*T*	*T*	Ξ
Ξ	Ξ	Ξ	Ξ	Ξ

Table 6-7: Truth tables for four-valued negation, conjunction and disjunction

Again the truth value Ξ dominates over all the other truth values in both, conjunctions and disjunctions. Contrast this with Codd's proposal of a four-valued logic [Codd 87], which hides misconceptions from the user: In Codd's logic the truth value Ξ does not infect the whole disjunction and we could obtain such question - answer pairs as, for example,

- *Is Mr. Brown or Mrs. White pregnant ? — Yes,*

which is impossible in our four-valued logic, since it reveals all misconceptions that can be detected.

For implication we obtain the truth table illustrated in Table 6-8:

A B	*F*	Ω	Ξ	*T*
F	*T*	*T*	*T*	*T*
Ω	Ω	Ω	Ω	*T*
Ξ	Ω	Ω	Ω	*T*
T	*F*	Ω	Ω	*T*

Table 6-8: Truth table for four-valued implication A → B

6.3.1. Expressive completeness

The logic proposed is not expressively complete (see [Jeffrey 81] for a definition of expressive completeness). For the two-valued logic the three truth-functional connectives conjunction, disjunction, and negation form an expressively complete set: for each of the 2^{2^n} different ways of assigning *T*ruth and *F*alsity to each of the 2^n different truth functions of a statement consisting of n variables, there exists a statement compounded out of those variables by means of conjunction, disjunction and negation, that has the same truth values. For example, let n = 2 and call the two variables A and B. Then we obtain the $2^{2^2} = 16$ different truth functions illustrated in Figure 6-1.

A	B	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
F	*F*	*F*	*T*	*F*	*T*	*F*	*T*	*F*	*T*	*F*	*T*	*F*	*T*	*F*	*T*	*F*	*T*
F	*T*	*F*	*F*	*T*	*T*	*F*	*F*	*T*	*T*	*F*	*F*	*T*	*T*	*F*	*F*	*T*	*T*
T	*F*	*F*	*F*	*F*	*F*	*T*	*T*	*T*	*T*	*F*	*F*	*F*	*F*	*T*	*T*	*T*	*T*
T	*T*	*F*	*F*	*F*	*F*	*F*	*F*	*F*	*F*	*T*	*T*	*T*	*T*	*T*	*T*	*T*	*T*

Figure 6-1: The 16 truth functions for statements with two two-valued components.

The following list shows how these functions can be obtained by means of conjunction, disjunction and negation.

$$\begin{aligned}
f_0 &= F \\
f_1 &= \neg(A \vee B) \\
f_2 &= \neg A \wedge B \\
f_3 &= \neg A \\
f_4 &= A \wedge \neg B \\
f_5 &= \neg B \\
f_6 &= (A \wedge \neg B) \vee (\neg A \wedge B) \\
f_7 &= \neg(A \wedge B) \\
f_8 &= A \wedge B \\
f_9 &= (A \wedge B) \vee (\neg A \wedge \neg B) \\
f_{10} &= B \\
f_{11} &= \neg A \vee B \\
f_{12} &= A \\
f_{13} &= A \vee \neg B \\
f_{14} &= A \vee B \\
f_{15} &= T
\end{aligned}$$

In the case of a three-valued logic, we obtain 3^{3^n} truth functions for n variables. The three connectives do not form an expressively complete set. For example, the assignment of the truth value Ω to any compound statement that does not contain at least one variable with truth-value Ω, is not representable by means of the three truth functional connectives.

However, for the purpose of data retrieval from a database, the logic is not required to be expressively complete: for example, functions that take *T*rue and *F*alse as the only truth values of their arguments and return an Ξ value are meaningless. They should neither be directly expressible, nor indirectly via some combination of given connectives. The same applies to truth functions that produce Ω under equivalent circumstances.

6.4. Final remarks

We have presented a formal way of handling the two types of null events introduced in Chapter 4, namely the Ω-type null event (*value at present unknown*), and the Ξ-type null event (*property inapplicable*).

The 'truth values' Ω (*value at present unknown*) and Ξ (*property inapplicable*) could also be seen as 'epistemic values'. The four-valued logic would then be seen as a four truth-valued semantics. In fact, as mentioned in the introduction, the term 'logic' has been used quite loosely in this Chapter, not to classify the concepts introduced as logics, but rather to follow the tradition of other papers published in this area, for example, [Belnap 75], [Vassiliou 79], [Date 86a] [19], [Codd 87]. What has been introduced as three-valued and four-valued 'logic' in those papers is far from being a complete logic system. No rules of derivation are given, and the fact that these 'logics' are not expressively complete is not even mentioned.

Our four-valued logic is also not expressively complete but we have shown that the specific purpose it has been designed for, does not require expressive completeness. We have given a set of rules of inference and an appropriate definition of implication, which are of crucial importance as soon as the system is to make use of intensional data such as functional dependencies and additional rules in order to derive values that are missing among the extensional data, whenever possible.

Our four-valued logic is based on pragmatic and methodological considerations, with a specific application in mind. It provides a formal way to investigate on composite queries where each component may result in a known, unknown or inapplicable value. The generation of appropriate responses to all kinds of queries can be stated in a formal and shorter way than it would be possible using algorithms or procedures. The notation is clear, exact and therefore easily verifiable.

[19] Null Values in Database Management, page 313-334

Chapter 7
Conclusions

7.1. Summary of the work done

We have presented a practical approach to eliminate wasteful operations in natural language access to relational database systems, using a knowledge-based subsystem.

The presentation of the approach is based on the extended relational model RM/T, which we have further extended to the RM/T* model. All relevant aspects of the extended RM/T model and all further extensions of the RM/T* model are presented in Chapters 2 and 3. Extensions of the RM/T* model concerning the database structure mainly consist of:

- A naming convention for characteristic entity types: the names of all characteristic entity types end in '_MVP' or '_MVP*n*', where *n* is an integer.
- A restriction in the use of characteristic entity types: In RM/T*, only kernel entity types and associative entity types are allowed to have characteristic entity types characterizing them.
- The notion of mutually exclusive subtypes.

In Chapter 2 we have also explicitly stated a number of integrity constraints which apply to the model but were not directly pointed out in [Codd 79] and [Date 83]. We have also clarified the distinction between characteristic entities and the designative entities of the extended RM/T model. This distinction seems to have been overlooked so far (see, for example, [Date 86b]); it is applicable to RM/T* as well. In Appendix A we have presented a diagramming technique for all versions of the RM/T model. This diagraming technique has proven to be helpful in the design of the database.

In Chapter 3 we have presented the catalog of the extended RM/T model and the additional extensions of the RM/T* model. The extensions are partially the counterparts of the extensions made in the database structure, and partially extensions which are independent of the changes made in the database structure. Altogether the extensions concerning the catalog of the RM/T* model include:

- An additional QUALIFIER attribute in the CATLG_DOMAINS relation used to further qualify the data types of the different domains.
- An explicit representation of the domain values of boolean data types and enumeration data types using a distinct relation for each type.

- Three additional relations: ATTRIBUTE_DOMAINS, ELEMENTS, and INTERVALS, which allow the representation of sub-domains for property attributes, that is, the domain of one property attribute can be a subdomain of the domain of another property attribute. This concept is analogous to the hierarchical aspect of entity types with their subtypes and supertypes.
- An additional EXC attribute in the CATLG_ATTRIBUTES relation, whose purpose is to facilitate the detection of the null events caused by misconceptions on the part of the user, which manifest in the query. The values in the EXC attribute field refer to specifically applicable constraints specified in the table EXCEPTION_RULES of the knowledge-base.
- Two additional attributes in the SUBTP_GRAPH relation: SPANNING_SUPERTYPE and MUTUALLY_EXCLUSIVE which specify whether or not a category spans the supertype and whether or not the subtypes are mutually exclusive.

In Chapter 4 we have clarified the distinction between null values and null events, and based on that distinction, shown the fundamental distinction between null events of type '*value at present unknown*' (Ω-type null events) and null events of type '*property inapplicable*' (Ξ-type null events). We have then shown that a misconception which manifests in the user query reflects a misconception on the part of the user, whereas a misconception which is actually represented in the database reflects a misconception on the part of the database designer. In a properly designed database, null values of type '*property inapplicable*' should not be represented. In fact, as we have shown, the notion of a null value of type '*property inapplicable*' [Atzeni and Parker 82], [Reiter 84], [Codd 86], [Date 86b], [Codd 87], is misleading by itself. We have then presented a classification of null events which is mainly based on minimizing the amount of database access required to detect the null event.

In Chapter 5 we outlined a practical implementation to detect Ξ-type null events which is based on the results of the previous Chapters. The method relies on the extensions made in the RM/T* model, although it is in principle adaptable to other semantic models as well. The goal in this method is to minimize, for each attribute occurring in a query, the set of possibly matching attribute values in the database. The minimization is subject to the constraint that the resulting set includes all domain values which could possibly be present in the tuples matching the constraints implied by the query. An Ξ-type null value is detected, whenever the resulting set for any of the attributes involved in the query is empty. The comment part of the rule which was involved in producing the empty set is then used for an appropriate answer generation.

Finally, in Chapter 6 we presented a digression on Codd's notion of a 'three valued logic' to deal with Ω-type null events [Codd 79] and an alternative to his notion of a four-valued logic to deal with Ω-type and Ξ-type null events [Codd 87]. Since the term 'logic' is widely used in the context of null values to designate an extension of the two standard truth values, we refer to our method as 'four valued logic'. However, we are aware of the fact that the term is actually a misnomer: the respective methods are far from being 'logics'.

7.2. General advantages of the RM/T* model

An important aspect of the RM/T* model is the fact that the domains of the different attributes are specified in a precise manner. The favorable consequences of this aspect are not limited to the detection of Ξ-type null events. As was pointed out in Chapter 6, such queries as

- Q.: *List all graduate students who are either in a special program, or in a master's program, in a Phd. program.*

Can be simplified if the knowledge that the three values specified represent the complete list of all possible domain values, is available.

In addition to the impact on query evaluation, RM/T* could also offer advantages to the query interpretation. For example, the different values of the NAME attribute of the relation DEPARTMENT_PROPERTIES are specified as a complete list. This list could be made accessible the parser and the semantic interpreter in the same way as the lexicon (see Figure 1-2). Thus the semantic interpreter of an RM/T* database could properly handle such queries as

- Q.: *Who taught CMPT 567 last semester ?* and
- Q.: *Who taught BOND 007 last semester ?*

The first query would be processed 'as expected' and result in a null value in case the *CMPT* department does not offer a course with number *567*. For the processing of the second query, the semantic interpreter would make use of the information that *BOND* is not a legal value for the DEPARTMENT_PROPERTIES.NAME attribute and either reject the query (with an appropriate explanation, and free the knowledge-based subsystem from doing the same task), or interpret *BOND 007* as a possible instance of a student name and generate the corresponding formal database query.

7.3. The problem of focus

An investigation on whether or not the problems associated with a lack of knowledge about the user and about the current focus might be reduced if the database system gives some explanations on what it is currently doing while the searches through the extensional data are going on might be quite interesting. Consider again the example of Section 6.1.2, page 75:

- *Does every grad student have a supervisor ?*

A system explaining its activities might respond with the following sequence of comments to the user:

- I'm looking for all the grad students...
- Done.
- I'm checking whether any of them does not have a supervisor...

- Done.
- ANSWER: *Yes, every grad student has a supervisor.*

A user might then interact as soon as he detects that the system interprets his query in some unintended way. For example, he might interrupt the evaluation process after reading the first comment and reformulate his query:

- *Is there a rule stating that every grad student has to have a supervisor ?*

7.4. Pseudo-indexed null values as a future extension

One important result of the research presented in this thesis is the disclosure of the concept of a 'null value of type *property inapplicable*' (see, for example, [ANSI 75], [Codd 79], [Vassiliou 79], [Atzeni and Parker 82], [Reiter 84], [Codd 86], [Date 86b], [Codd 87]) as a misconception. This disclosure has considerable impact on earlier approaches to the problems with null events of type *property inapplicable* (for example, [Vassiliou 79], [Codd 86], [Codd 87]) which rely on this misconception.

In this final section we present some guidelines for future extensions concerning the representation of null values of type *value at present unknown.* As was mentioned in Chapter 4, there is some dispute over the proper representation of those null values, in particular over the question whether the null values should be indexed or not. The approach outlined here can be seen as a pseudo-indexed representation of null values: a null value would be indexed in precisely thoses cases, when the actual value of the respective attribute is unknown, but some further information about the value is present. In this case we replace the term 'null value' with the term 'special value'. The null value would not be indexed if no further information is available.

To represent a null value in some attribute field of a database, it is necessary to find a bit configuration that is different from all bit configurations that represent nonnull values in that field so that the null value cannot be confused with any nonnull value. In general such a bit configuration might not exist. It is then necessary to introduce a *hidden field,* in order to distinguish the null values from all other values in the attribute field. For example, in [Date 86b] Date states that [20]

> "In DB2, a column that can accept null values is physically represented in the stored database by two columns, the data column itself and a hidden indicator column, one byte wide, that is stored as a prefix to the actual data column. An indicator column value of all ones indicates that the corresponding data column value is to be ignored (that is, taken as null); an indicator column of all zeroes indicates that the corresponding data column value is to be taken as genuine (that is, nonnull)."

[20] page 120. DB2 is an IBM relational database product.

And in [Codd 86], Codd argues that

"in the context of computer-supported database management, it is unacceptable to reserve any specific character string value to denote the fact that a db-value is missing".

In case the hidden field marks an entry as null value, the data field of the attribute is wasted in the standard approach. Any bit configuration appearing in that part is not interpreted according to the declaration of the attribute type, but simply ignored. We propose to make extended use of both the hidden field and the data field. Part of our proposal is to use a minimum width for the physical storage of the data column such that each data column can accommodate for a surrogate value, and to add a hidden field to each attribute in the database which is allowed to contain null values or special values.

Theoretically the hidden field need be no more than a single bit wide, but for pragmatic reasons (mainly hardware of the system) it is more convenient to let it occupy an entire byte or word [21]. The hidden field will be used to mark not only null values, but also other special cases that could all be seen as *special values* with respect to the declaration of the attribute field: values whose data fields are to be interpreted in a different way (specified by the hidden field) than the declaration of the respective attribute suggests. Except for the case of the 'normal' null values, the data field will be used to store a key value of some additional catalog relation whenever the hidden field marks some special value.

We use a one-byte hidden field as prefix to the data field

H_7	H_6	H_5	H_4	H_3	H_2	H_1	H_0

Instead of using the bit combinations '00000000' and '11111111' in the hidden field to distinguish between a genuine data value and a null value, the database uses only bit H_7 for the same purpose and ignores bits H_6 through H_0. Thus the database will interpret the contents data field as genuine value whenever H_7 = '0' and as null value otherwise.

Whenever H_7 = '1', the knowledge-base will interpret the data field as special value. The specific type of special value is specified by bits H_6 - H_0. For example, if bits $<H_7, H_6>$ = '11', the data field is interpreted as the 'normal' ω-type null value; if bits $<H_7 - H_4>$ = '1000', the data field is interpreted as a surrogate value, referring to one or more special relations in the catalog. Bits H_3 - H_0 specify those special

[21] here 'word' refers to a computer word, that is, the number of bits addressable by the cpu at a time, or the number of bits stored as one unit in memory.

relations. Thus the extensions are hidden from the database and only visible to the knowledge-based subsystem.

7.4.1. Relations for comparisons and set in/exclusions

We add four distinct relations to the catalog to represent extensional and intensional comparisons among values of equal type. Whenever the hidden field contains the bit combination '1000' in bits H_7 - H_4, the data field will contain a system generated surrogate. Bits H_3 - H_0 will then be used to specify one or more of the additional catalog relations, which contain the respective surrogate in one of their key attributes (all four additional relations have composite key attributes). All constraints specified via these additional relations hold simultaneously.

7.4.1.1. Extensional comparisons

We add the two relations SPECIAL_E_AND and SPECIAL_E_OR to the catalog in order to represent the extensional constraints that are met by specific special values in the database, relative to specified constant values.[22]

- SPECIAL_E_AND (SPECIAL_E_AND¢, IS, VALUE)
- SPECIAL_E_OR (SPECIAL_E_OR¢, IS, VALUE)

The format of both relations is identical. SPECIAL_E_AND represents the constraints that must hold simultaneously, SPECIAL_E_OR represents the set of constraints of which at least one must hold. Bit H_0 of the hidden field indicates, whether or not the data field contains a surrogate value referring to the SPECIAL_E_AND relation; bit H_1 indicates, whether or not the data field contains a surrogate value referring to the SPECIAL_E_AND relation. We use the SPECIAL_E_OR relation to explain how the information is represented in both relations. The analogue holds for the SPECIAL_E_AND relation.

The SPECIAL_E_OR relation has two key attributes, SPECIAL_E_OR¢ and VALUE. As the suffix '¢' indicates, the domain of SPECIAL_E_OR¢ consists of system generated surrogates. The domain of the VALUE attribute field is not directly specified. In some way it might be seen as a superset of all the declared domains of the database. The values in the VALUE attribute field of all tuples that contain the same value in their SPECIAL_E_OR¢ attribute field, belong to precisely one of the declared domains of the database: the 'primary' domain of the attribute field which contains the respective value of the SPECIAL_E_OR¢ attribute. Thus the

[22] Adapting the terminology from programming languages, we could call this a 'specification by value'.

specification of the domain of the VALUE attribute field of the knowledge-base relation SPECIAL_E_OR is *inherited* from the database for each individual entry, and so is the correct interpretation of the data stored in the VALUE field. The same bit pattern stored in different locations in the VALUE field might have different meanings, depending on what interpretation of the data has been inherited in the particular case. The specification of the inherited interpretation need not be stored in the knowledge-base, since the access path to the SPECIAL_E_OR relation always starts somewhere in the database and implicitly carries the required information. A direct access to the SPECIAL_E_OR relation would result in meaningless data and is made impossible. The SPECIAL_E_OR relation is invisible to the user. The attribute name VALUE does not have the suffix '¢', since the corresponding values could refer to entities as well as to properties.

The domain of the IS attribute consists of six different symbols representing the six comparators '≤', '<', '=', '≥', '>', and '≠'. Semantically the special value in the SPECIAL_E_OR¢ field of a tuple is connected to the constant value in the VALUE field via the comparator in the IS field.

An example will clarify this concept. Suppose we want to represent the fact that

- *The status of instructor Newton is one of the three: instructor, assistant professor, or associate professor.*

Notice, that the specified list is complete. If *Newton* could as well have some other status, the information provided by giving just that subset is equal to none. Further suppose the surrogate identifying *Newton* is 'newton01'. The ACADEMIC_PROPERTY relation then contains the following tuple (call it ***aca1*** for further reference):

ACADEMIC_PROPERTY

ACADEMIC¢	STATUS	
...	...	
newton01	82 surrg01	***aca1***
...	...	

The special value 'surrg01' in the data field of the STATUS attribute is a system generated surrogate; the prefix '82' represents the bit pattern in the hidden field, coded in hexadecimal ('82_{hex}' = '10000010_{binary}'). This bit pattern in the hidden field indicates that the data field contains a surrogate value referring to the SPECIAL_E_OR relation of the catalog. Just as it was the case with the 'normal' null values, the value in the hidden field prohibits an interpretation of the data field according to the domain declaration of the attribute.

Now the SPECIAL_E_OR relation in the extended catalog *could* contain the three tuples:

SPECIAL_E_OR

SPECIAL_E_OR¢	IS	VALUE
...	...	...
surrg01	=	PROF
surrg01	=	ASSO
surrg01	=	ASSI
...	...	...

Notice that the above table renders the 'interpreted' data of the VALUE field. The information stored here is recognizable only due to the specific access path to the SPECIAL_E_OR relation. The same bit patterns could have a totally different interpretation, were they accessed via a different relation (or just a different attribute of the ACADEMIC_PROPERTY relation). Here the domain of the STATUS attribute has propagated to the VALUE attributes of all tuples with the specific SPECIAL_E_OR¢ value surrg01.

As long as it is not known, which one of the three stati corresponds to the instructor *Newton*, all possible values are represented in the SPECIAL_E_OR relation; as soon as one value is known to be true, the knowledge-based subsystem enters that value in the STATUS field for the respective academic in the ACADEMIC_PROPERTY relation and deletes all the remaining corresponding entries in the SPECIAL_E_OR relation.

The above representation in the SPECIAL_E_OR relation is not the most efficient one. Since the domain of the ACADEMIC_PROPERTY.STATUS attribute is declared as one to which the '>' predicate is applicable ('ORDERING' is 'YES'; see Appendix C.7 and Appendix C.1), and the three alternative values are adjacent to each other in the given ordering (see Appendix C.2), we can represent the same information with only two tuples (call them ***sp1*** and ***sp2*** for further reference) in the SPECIAL_E_AND relation:

SPECIAL_E_AND

SPECIAL_E_AND¢	IS	VALUE	
...	...	...	
surrg01	≤	ASSO	***sp1***
surrg01	≥	INST	***sp2***
...	...	...	

The switch to the SPECIAL_E_AND relation was required, since the constraints represented by the two tuples must hold simultaneously and not alternatively as was the case in the SPECIAL_E_OR relation. The only difference in the ACADEMIC_PROPERTY relation is the fact that among the set of the four bits H_3 - H_0 of the hidden field, used to represent the respective references, bit H_0 is set instead of bit H_1 to indicate that the surrogate in the data field refers to the SPECIAL_E_AND relation instead of to the SPECIAL_E_OR relation. Thus we have the bit pattern '10000001_{binary}' or '81_{hex}' as prefix.

7.4.1.2. Intensional comparisons

We add two more relations to the catalog in order to handle incomplete information via special values: SPECIAL_I_AND and SPECIAL_I_OR.

- SPECIAL_I_AND (<u>SPECIAL_I_AND¢</u>, IS, <u>REF_RELNAME</u>, <u>REF_KEY¢</u>, <u>REF_ATTR</u>, <u>XX</u>, <u>OFFSET</u>)
- SPECIAL_I_OR (<u>SPECIAL_I_OR¢</u>, IS, <u>REF_RELNAME</u>, <u>REF_KEY¢</u>, <u>REF_ATTR</u>, <u>XX</u>, <u>OFFSET</u>)

SPECIAL_I_AND and SPECIAL_I_OR are used to represent the intensional constraints that are met by special values in the database, that is, constraints not relative to specified constant values, but relative to other values stored in the database, and referred to via these two relations[23]. Bit H_2 in the hidden field indicates whether or not the surrogate in the data field refers to the SPECIAL_I_AND relation; bit H_3 in the hidden field indicates whether or not the surrogate in the data field refers to the SPECIAL_I_OR relation.

In the relations SPECIAL_E_AND and SPECIAL_E_OR introduced in the last section, we need only one attribute (VALUE) to represent the value, the special value is to be compared to. Here we need five attributes (REF_RELNAME, REF_KEY, REF_ATTR, XX, and OFFSET) to represent the value the special value is to be compared to. REF_RELNAME specifies the relation in which that value is to be found, REF_KEY specifies the key value of the respective tuple, and REF_ATTR specifies the attribute within that tuple. The two attributes XX and OFFSET are used to specify a constant offset. The domain of attribute OFFSET is *inherited* in exactly the same way as it is the case with the attribute VALUE in the previous two relations. The domain of attribute XX consists of the three symbols '+', '-', and '*na*'. The first two specify the sign of the offset, if an offset value is given; the third specifies that an offset is not given or not applicable (for unordered domains). Otherwise the use of these two relations SPECIAL_I_AND and SPECIAL_I_OR is very similar to the one of the two relations SPECIAL_E_AND and SPECIAL_E_OR described in the last section.

An example should clarify the concept. First let us add the fact that

- *Kepler has the status of assistant professor*

Thus we add another tuple (call it ***aca2*** for further reference) to the ACADEMIC_PROPERTY relation (assume that 'kepler02' is the surrogate identifying *Kepler*):

[23] Adapting the terminology from programming languages, we could call this 'specification by reference'.

ACADEMIC_PROPERTY

ACADEMIC¢	STATUS	
...	...	
newton01	81 surrg01	*aca1*
kepler02	81 surrg02	*aca2*
...	...	

Now suppose we want to add some more information about *Newton*'s status:

• *Newton's status is at least as high as Kepler's*

The respective tuple (***aca1***) is already present in the ACADEMIC_PROPERTY relation. So far the special value surrg01 only refers to the SPECIAL_E_AND relation since the value of the hidden field is '10000001_{binary}'. We additionally set bit H_2 in the hidden field of the STATUS attribute and thus specify that there are additional constraints represented in the SPECIAL_I_AND relation. Thus the value of the hidden field becomes '10000101_{binary}' or '85_{hex}'. The SPECIAL_I_AND relation now contains the following tuple (call it ***sp3*** for further reference):

SPECIAL_I_AND

SPECIAL_I_AND¢	IS	REF_RELNAME	REF_KEY¢	REF_ATTR	XX	OFFSET
...	...	.....	...	...	...	...
surrg01	≥	ACADEMIC_PROPERTY	kepler02	STATUS	*na*	-----
...	...	.....	...	...	...	...

The values of the attributes IS, REF_RELNAME, and REF_ATTR specify that the respective special value is greater than some value (be it special too, or not) in the STATUS attribute of the ACADEMIC_PROPERTY relation. The value 'kepler02' of the REF_KEY attribute specifies the respective tuple.

The value '*na*' in the XX field specifies, that no offset is given (although it would be applicable here, since the domain STATUS is ordered). We could also have specified an offset of zero units, but this would result in a wasteful addition or subtraction operation whenever the value is retrieved.

It should be noted that the four relations introduced to handle incomplete information via special values, can relate a specific special value to any number of other values, but all those values have to be of the same matching type. Thus it is possible to represent such propositions as

• *Mary has a better grade than John* and

• *The departments for Physics and Business belong to different faculties*

but it is not possible to directly represent such propositions as

• *Newton is an assistant professor, or CMPT 100 has 9 units.*

Here two properties of different type are related to each other in a single statement. Higher level concepts are needed to be represented such propositions.

7.4.1.3. Deducing values

As soon as a set of values previously believed to be possible is known to be false, the respective tuples can be deleted from the corresponding SPECIAL[24] relations. If these operations leave no more than a single precise value for the respective group of entries, the system can deduce that this must be the only possible value and copy that value to the original attribute field. The single remaining tuple in the SPECIAL relation can then be deleted too.

In our above example this works as follows: The tuples ***sp1*** and ***sp2*** tell us that

- *Newton*'s status is INST, ASSI, or ASSO.

After adding the tuple ***sp3*** to the SPECIAL_I_AND relation we also know that

- *Newton's status is at least as high as Kepler's.*

An attempt to retrieve *Kepler*'s status successfully returns the value 'ASSI' from tuple ***in2*** Thus we know that in the current state

- *Newton*'s status is at least ASSI.

This excludes 'INST' from the alternatives specified via tuples ***sp1*** and ***sp2***. We are left with the more concise information

- *Newton*'s status is ASSI or ASSO.

We could now update the VALUE attribute in tuple ***sp2*** in the SPECIAL_E_AND relation accordingly, but we cannot delete the tuple in the SPECIAL_I_AND relation. Doing so would mean that we misinterpret the intensional information

- *Newton*'s status is at least as high as *Kepler*'s.

with the extensional information

- *Newton*'s status is at least ASSI.

Even an update of the VALUE attribute in tuple ***sp2*** should not be performed, since a degradation of *Kepler*'s status cannot be ruled out unless the database contains the respective constraints.

In general the deduction procedures should not be invoked by update operations (delete, insert, and change), but by retrieval operations only. If only extensional information was used to obtain the value, then that value can be represented in the respective tuple(s).

[24] SPECIAL stands for any of the four relations introduced to handle special values.

Although we believe that the outlined suggestion for future research is a promising alternative to represent both null values and partial information, we realize that the proposal is far from being a solution. Much work remains to be done in this area as well as related areas.

Appendix A
A diagramming technique for RM/T databases

We propose a diagramming technique for RM/T databases. The corresponding diagrams do not show all implementational details. For example, instead of showing the individual P-relations that are used to represent the different properties of an entity type, only the property attributes themselves are represented. However, we believe that it is a useful part of the overall documentation, as well as a valuable guide during the actual database design. Based on this technique we present in Figure A-8 a diagram of the RM/T* database designed in this thesis.

An ***entity type*** is represented by a wide rectangular box labeled with the name of the E-attribute primary key of the corresponding E-relation. Note that all E-attribute names end with the character '¢'. Figure A-1 illustrates the representation of the entity type STUDENT.

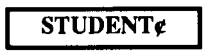


Figure A-1: Representation of the entity type STUDENT.

The set of property attributes for a particular entity type is represented by a contiguous string of boxes attached to the right of the rectangular box representing the entity type. The ***property references***, that is, the references from the E-attribute primary keys of those P-relations, which actually contain the property attributes, to the E-relation for the corresponding entity type are thus implicitly described by one contiguous string of boxes.

Each box represents one property attribute of a P-relation for the corresponding entity type, and is labeled with the respective attribute name, except for those property attributes representing associative and characteristic references (see the next two paragraphs). Figure A-2 illustrates the representation of the entity type STUDENT, together with its properties NAME, NUMBER and SEX.

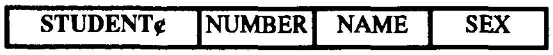


Figure A-2: Representation of the entity type STUDENT together with its properties NAME, NUMBER, and SEX.

An ***association reference***, that is, a reference from an E-attribute of a P-relation for an associative entity type to the E-relation for a participant in that association, is represented by an arrow pointing to the representation of the respective entity type participating in the association, and originating from a square with a circle labeled 'A'. Thus the square represents a property attribute which happens to be an E-attribute referring to some other entity type. The name of this E-attribute is not represented, instead an arrow points to the representation of the entity type referred to by the E-attribute. Like all boxes representing property attributes, the square is placed at the right of the wide rectangular box representing the corresponding associative entity type. Figure A-3 illustrates the representation of the entity type TEXT, together with its associative references to the entity types COURSE and BOOK.

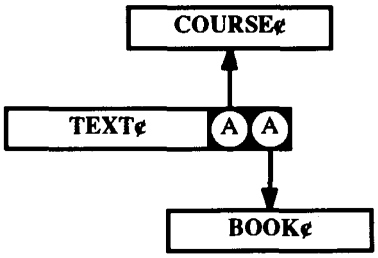


Figure A-3: Representation of the associative entity type TEXT together with its associative references to the entity types COURSE and BOOK.

A ***characteristic reference***, that is, a reference from an E-attribute of a P-relation for a characteristic entity type to the E-relation for the immediately superior entity type being characterized, is represented by an arrow pointing to the representation of the respective entity type being characterized, and originating from a square with a circle labeled 'C'. Like all boxes representing property attributes, the square is placed at the right of the wide rectangular box representing the corresponding characteristic entity type. Figure A-4 illustrates the representation of the kernel entity type BOOK, together with its single-valued property TITLE and its multi-valued property AUTHOR. The multi-valued property AUTHOR is represented via the characteristic entity type BOOK_MVP.

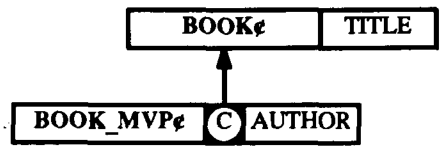


Figure A-4: Representation of the kernel entity type BOOK together with its single-valued property TITLE and its multi-valued property AUTHOR. The multi-valued property AUTHOR is represented via the characteristic entity type BOOK_MVP.

A ***designation reference***, that is, a reference from an E-attribute of a P-relation for a designative entity type to the E-relation for the immediately superior entity type being designated, is represented by an arrow pointing to the representation of the respective entity type being designated, and originating from a narrow rectangular box with half circles at its left and right ends. It is labeled with the name of the E-attribute property representing the designation. Figure A-5 illustrates the representation of the designative kernel entity type GRAD, together with its designative reference SUPRV to the entity type ACADEMIC.

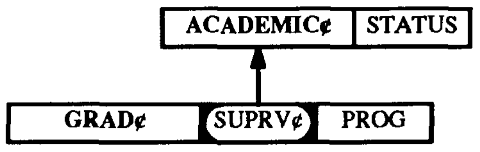


Figure A-5: Representation of the designative kernel entity type GRAD, together with its designative reference SUPRV to the entity type ACADEMIC.

Kernel-, associative- and characteristic entities can all be designative in addition. Thus,

- an associative entity type is one whose representation has at least two squares with circles labeled 'A' attached to its right,
- a characteristic entity type is one whose representation has a square with a circle labeled 'C' attached to its right, and
- a kernel entity type is one whose representation has neither a square with a circle labeled 'A' nor a square with a circle labeled 'C' attached to its right.

The dashed lines connecting the representations of entity types represent the ***hierarchical aspect***: they connect the superior entity with the immediately subordinate entities. The attached label specifies the category of the specialization / generalization. Figure A-6 illustrates the representation of the kernel entity type STUDENT, together with its subordinate entity types GRAD and UNDER_GRAD. The category of the specialization / generalization is named STATUS.

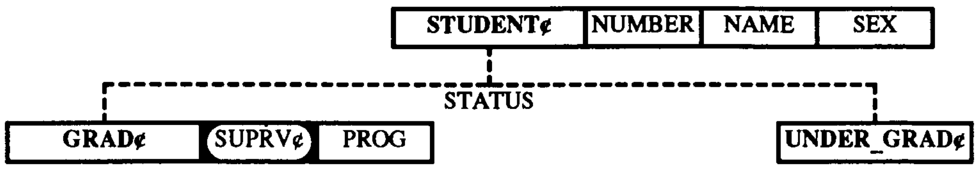


Figure A-6: Representation of the kernel entity type STUDENT, together with its subordinate entity types GRAD and UNDER_GRAD. The category of the specialization / generalization is named STATUS.

As mentioned above, the name of a property attribute representing an associative or a characteristic reference to some other entity type is not represented in this diagram. In general, such a name is identical to the name of the corresponding E-attribute. There are, however, exceptions to this rule of thumb. For example, two distinct attribute names, FOR_COURSE¢ and IS_COURSE¢, are required for the two properties of the entity type PRE_REQ. All non-square rectangular boxes are labeled with the names of the attributes they represent.

Attributes that do not accept null values (for example, all E-attribute primary keys) are enclosed in solid boxes. All attributes represented by dashed boxes, do accept null values. Figure A-7 illustrates the representation of the kernel entity type DEPARTMENT, together with its property types CHAIR¢, NAME and FACLTY. The primary key DEPARTMENT¢ and the user key NAME do not accept null values. The designative reference CHAIR¢ and the 'normal' property FACLTY do accept null values.

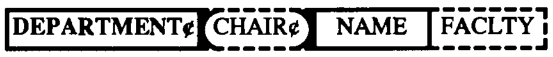


Figure A-7: Representation of the kernel entity type DEPARTMENT, together with its property types. The properties CHAIR¢ and FACLTY accept null values.

Based one this diagramming technique, Figure A-8 shows the diagram of the complete RM/T database designed in this thesis.

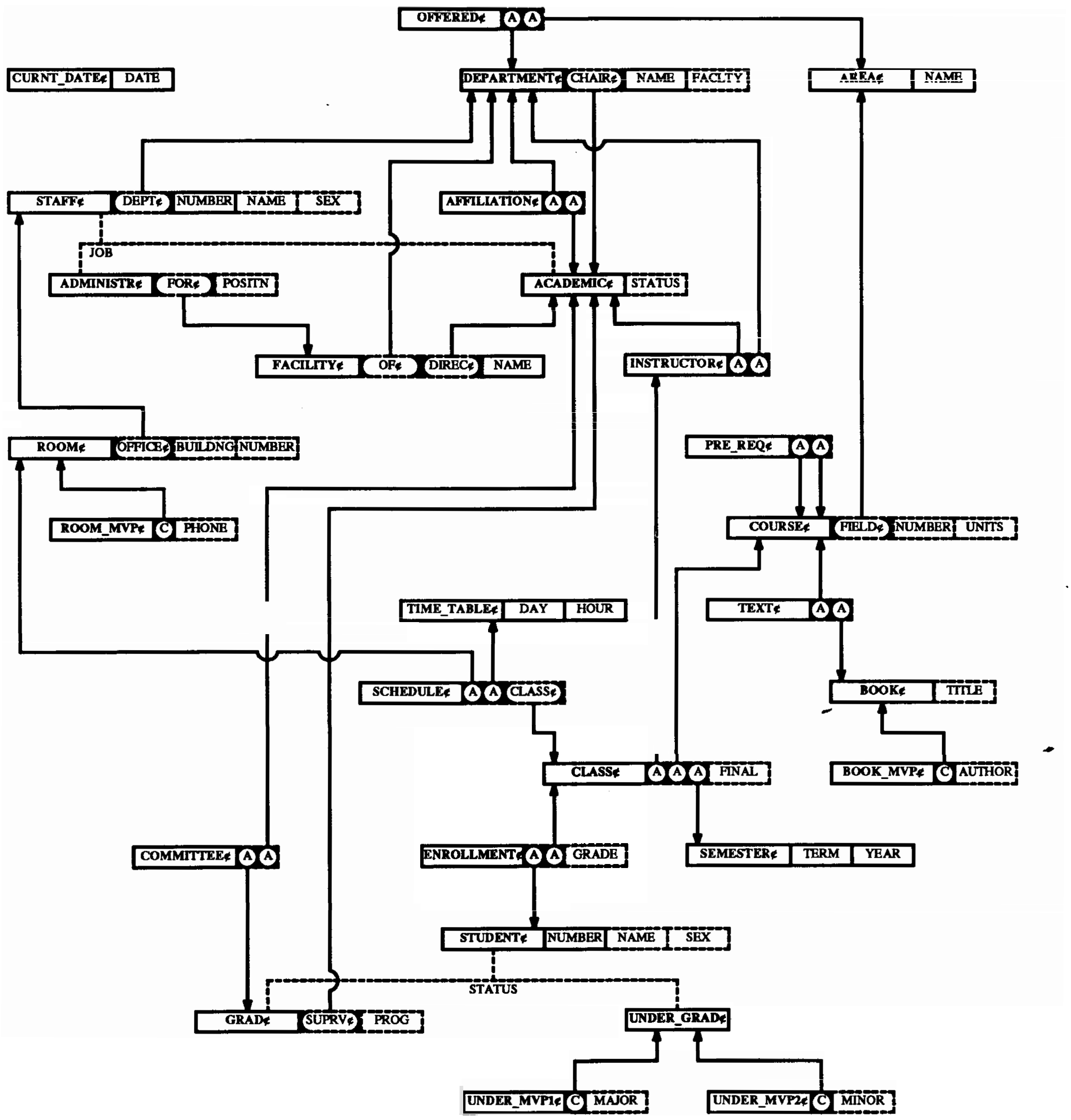
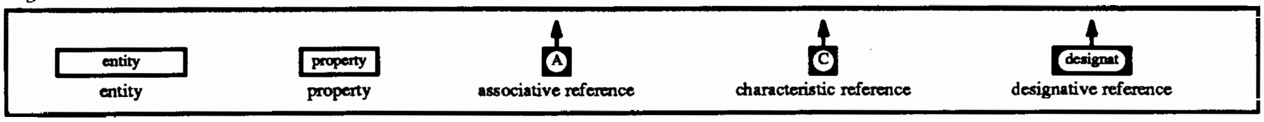


Figure A-8: Diagram of the logical structure of the database.

Legend:



Appendix B
The relations of the database

E-relations

B.1. E-relations for kernel entities

ACADEMIC kernel;
subentity of STAFF per category JOB

```
_______________ _
| ACADEMIC¢     |
|===============|
|               |
```

ADMINISTR kernel;
subentity of STAFF per category JOB
designating (FACILITY via FOR¢)

```
_______________ _
| ADMINISTR¢    |
|===============|
|               |
```

AREA inner kernel;

```
_______________ _
| AREA¢         |
|===============|
|               |
BOOK               inner kernel;

 ______________ _
| BOOK¢        |
|==============|
|              |


COURSE             inner kernel;
                   designating   (  AREA via FIELD¢                )

 ______________ _
| COURSE¢      |
|==============|
|              |


CURNT_DATE         inner kernel

 ______________ _
| CURNT_DATE¢  |
|==============|
|              |


DEPARTMENT         inner kernel
                   designating   (  ACADEMIC via CHAIR¢            )

 ______________ _
| DEPARTMENT¢  |
|==============|
|              |


FACILITY           inner kernel
                   designating   (  DEPARTMENT via OF¢             )
                   designating   (  ACADEMIC via DIREC¢            )

 ______________ _
| FACILITY¢    |
|==============|
|              |


GRAD               kernel
                   subentity of     STUDENT   per category   STATUS
                   designating   (  ACADEMIC via SUPRV¢            )

 ______________ _
| GRAD¢        |
|==============|
|              |
```

ROOM inner kernel
designating (STAFF via OFFICE¢)

```
 ______________ _
| ROOM¢        |
|==============|
|              |
```

SEMESTER inner kernel

```
 ______________ _
| SEMESTER¢    |
|==============|
|              |
```

STAFF inner kernel
designating (DEPARTMENT via DEPT¢)

```
 ______________ _
| STAFF¢       |
|==============|
|              |
```

STUDENT inner kernel

```
 ______________ _
| STUDENT¢     |
|==============|
|              |
```

TIME_TABLE inner kernel

```
 ______________ _
| TIME_TABLE¢  |
|==============|
|              |
```

UNDER_GRAD kernel
subentity of STUDENT per category STATUS

```
 ______________ _
| UNDER_GRAD¢  |
|==============|
|              |
```

B.2. E-relations for associative entities

```
AFFILIATION          associating (  ACADEMIC via ACADEMIC¢,
                                    DEPARTMENT via DEPARTMENT¢     )

 _______________
| AFFILIATION¢  |
|===============|
|               |
```

```
CLASS                associating (  COURSE via COURSE¢,
                                    INSTRUCTOR via INSTRUCTOR¢,
                                    SEMESTER via SEMESTER¢         )

 _______________
| CLASS¢        |
|===============|
|               |
```

```
COMMITTEE            associating (  ACADEMIC via ACADEMIC¢,
                                    GRAD via GRAD¢                 )

 _______________
| COMMITTEE¢    |
|===============|
|               |
```

```
ENROLLMENT           associating (  CLASS via CLASS¢,
                                    STUDENT via STUDENT¢           )

 _______________
| ENROLLMENT¢   |
|===============|
|               |
```

```
INSTRUCTOR           associating (  ACADEMIC via ACADEMIC¢,
                                    DEPARTMENT via DEPARTMENT¢     )

 _______________
| INSTRUCTOR¢   |
|===============|
|               |
OFFERED             associating (  AREA via AREA¢,
                                   DEPARTMENT via DEPARTMENT¢      )
________________ _
| OFFERED¢       |
|===============|
|               |


PRE_REQ             associating (  COURSE via FOR_COURSE¢,
                                   COURSE via IS_COURSE¢           )
________________ _
| PRE_REQ¢       |
|===============|
|               |

SCHEDULE            associating (  ROOM via ROOM¢,
                                   TIME_TABLE via TIME_TABLE¢      );
                    designating (  CLASS via CLASS¢                )
________________ _
| SCHEDULE¢      |
|===============|
|               |


TEXT                associating (  BOOK via BOOK¢,
                                   COURSE via COURSE¢              )
________________ _
| TEXT¢          |
|===============|
|               |
```

B.3. E-relations for characteristic entities

BOOK_MVP	characterizing with multi-valued property	BOOK AUTHOR

```
 ______________ _
| BOOK_MVP¢    |
|==============|
|              |
```

ROOM_MVP	characterizing with multi-valued property	ROOM PHONE

```
 ______________ _
| ROOM_MVP¢    |
|==============|
|              |
```

UNDER_MVP1	characterizing with multi-valued property	UNDER_GRAD MAJOR

```
 ______________ _
| UNDER_MVP1¢  |
|==============|
|              |
```

UNDER_MVP2	characterizing with multi-valued property	UNDER_GRAD MINOR

```
 ______________ _
| UNDER_MVP2¢  |
|==============|
|              |
```

P-relations

B.4. P-relation for ACADEMIC

ACADEMIC_PROPERTY

ACADEMIC¢	STATUS

ACADEMIC is a kernel entity, subordinate to STAFF, representing academic staff members. The primary key of the relation ACADEMIC_PROPERTY is E-attribute ACADEMIC¢, the corresponding domain is the set of surrogate values currently existing in the E-relation ACADEMIC. The attribute STATUS represents a single valued property: the status of the academic staff member (for example 'ASSI'). The domain of STATUS is 'STATUS', the respective value type is represented as a complete list in Appendix C.2.

B.5. P-relation for ADMINISTR

ADMINISTR_PROPERTIES

ADMINISTR¢	FOR¢	POSITN

ADMINISTR is a kernel entity, subordinate to STAFF, representing administrative staff members. The primary key of the relation ADMINISTR_PROPERTIES is E-attribute ADMINISTR¢, the corresponding domain is the set of surrogate values currently existing in the E-relation ADMINISTR. The attribute FOR¢ represents a designative reference to the kernel entity type FACILITY; it indicates to which facility within the department the ADMINISTR is assigned. The domain of E-attribute FOR¢ is the set of surrogate values currently existing in the E-relation FACULTY, unioned with the null value E-null. The attribute POSITN represents a single valued property: the position of the administrative staff member within the department (for example 'SECRETARY'). The domain of POSITN is 'POSITION', the respective value type is represented as a complete list in Appendix C.2.

B.6. P-relation for AFFILIATION

AFFILIATION_INSTANCE

AFFILIATION¢	ACADEMIC¢	DEPARTMENT¢

AFFILIATION is an associative entity type. The primary key of relation AFFILIATION_INSTANCE is E-attribute AFFILIATION¢, the corresponding domain is the set of surrogate values currently existing in the E-relation AFFILIATION. Each tuple of the AFFILIATION_INSTANCE relation associates an ACADEMIC and a DEPARTMENT, whose respective surrogates are contained in the E-attributes ACADEMIC¢ and DEPARTMENT¢. The domain of E-attribute ACADEMIC¢ is the set of surrogate values currently existing in the E-relation ACADEMIC, unioned with the null value E-null. The domain of E-attribute DEPARTMENT¢ is the set of surrogate values currently existing in the E-relation DEPARTMENT, unioned with the null value E-null. The associative entity type AFFILIATION is used to represent the fact that an academic staff member of on department can be affiliated with several other departments.

B.7. P-relation for AREA

AREA_PROPERTY

AREA¢	NAME

AREA is a kernel entity, representing the different areas of study offered in the university. The primary key of the relation AREA_NAME is E-attribute AREA¢, the corresponding domain is the set of surrogate values currently existing in the E-relation AREA. The attribute NAME represents a single valued property: the name of the area (for example 'CMPT'). The domain of NAME is 'FIELD', the respective value type is represented as a complete list in Appendix C.2.

B.8. P-relation for BOOK

BOOK_PROPERTY

BOOK¢	TITLE

BOOK is a kernel entity, representing the different books used for the different courses offered in the university. The primary key of the relation BOOK_PROPERTY is E-attribute BOOK¢, the corresponding domain is the set of surrogate values currently existing in the E-relation BOOK. The attribute TITLE represents a single valued property: the title of the book (for example 'The Knowledge Frontier'). The domain of TITLE is 'NAME', the respective value type is an array of 60 characters representing one full title of a book.

B.9. P-relation for BOOK_MVP

BOOK_AUTHOR

BOOK_MVP¢	BOOK¢	AUTHOR

BOOK_AUTHOR is a characteristic entity, characterizing BOOK. The primary key of relation BOOK_AUTHOR is the E-attribute BOOK_MVP¢, the corresponding domain is the set of surrogate values currently existing in the E-relation BOOK_MVP. The E-attribute BOOK¢ represents the characteristic reference to the BOOK being characterized, the corresponding domain is the set of surrogate values currently existing in the E-relation BOOK. The property attribute AUTHOR represents the name of one author (for example 'Nick Cercone'). The domain of the attribute AUTHOR is 'NAME', the respective value type is an array of 60 characters representing the full name of one author.

B.10. P-relations for CLASS

CLASS_INSTANCE

CLASS¢	COURSE¢	INSTRUCTOR¢	SEMESTER¢

CLASS_PROPERTY

CLASS¢	FINAL

CLASS is an associative entity type. The primary key attribute of the relation CLASS_INSTANCE is E-attribute CLASS¢, the corresponding domain is the set of surrogate values currently existing in the E-relation CLASS. Each tuple associates a COURSE and an INSTRUCTOR, and a SEMESTER. The respective surrogate values are contained in the E-attributes COURSE¢, INSTRUCTOR¢ and SEMESTER¢. The domain of E-attribute COURSE¢ is the set of surrogate values currently existing in the E-relation COURSE, unioned with the null value E-null. The domain of E-attribute INSTRUCTOR¢ is the set of surrogate values currently existing in the E-relation INSTRUCTOR, unioned with the null value E-null. The domain of E-attribute SEMERSTER¢ is the set of surrogate values currently existing in the E-relation SEMERSTER, unioned with the null value E-null.

The primary key of the relation CLASS_PROPERTY is E-attribute CLASS¢, the corresponding domain is the set of surrogate values currently existing in the E-relation CLASS. The attribute FINAL represents a property of a class. The FINAL is the date of the final exam of the respective class. This date is specified as the day of the year; the domain of FINAL is 'day_num', the respective values are of type '1..365'.

B.11. P-relation for COMMITTEE

COMMITTEE_INSTANCE

COMMITTEE¢	GRAD¢	ACADEMIC¢

COMMITTEE is an associative entity type. The primary key of relation COMMITTEE_INSTANCE is E-attribute COMMITTEE¢, the corresponding domain is the set of surrogate values currently existing in the E-relation COMMITTEE. Each tuple of the COMMITTEE_INSTANCE relation associates an ACADEMIC and a

GRAD, whose respective surrogates are contained in the E-attributes ACADEMIC¢ and GRAD¢. The domain of E-attribute ACADEMIC¢ is the set of surrogate values currently existing in the E-relation ACADEMIC, unioned with the null value E-null. The domain of E-attribute GRAD¢ is the set of surrogate values currently existing in the E-relation GRAD, unioned with the null value E-null. The associative entity type COMMITTEE is used to represent that every graduate student can have several academic staff members in his supervisory committee, and every academic staff member can be part of several supervisory committees.

B.12. P-relation for COURSE

COURSE_PROPERTIES

COURSE¢	FIELD¢	NUMBER	UNITS

COURSE is an inner kernel entity, designating entities of type AREA. The primary key of the relation COURSE_PROPERTIES is E-attribute COURSE¢, the corresponding domain is the set of surrogate values currently existing in the E-relation COURSE. The E-attribute FIELD¢ refers to the E-relation of the AREA being designated. The domain of E-attribute FIELD¢ is the set of surrogate values currently existing in the E-relation AREA, unioned with the null value E-null. The two properties NUMBER and UNITS represent the course number and the number of units of the respective course. The domain of NUMBER is 'cours_num', the respective values are of type '0..999'. The domain of UNITS is 'unit_num', the respective values are of type '0..9'.

B.13. P-relation for CURNT_DATE

TODAY

CURNT_DATE¢	DATE

CURNT_DATE is an inner kernel entity. TODAY is a *dummy* relation. Its primary key is E-attribute CURNT_DATE¢, the corresponding value is a system generated surrogate. At any point in time TODAY contains exactly one tuple with one property value for the entity CURNT_DATE: the value of the current date represented in the DATE attribute. The domain of DATE is 'day_num', the respective type is '1..365'; the date is specified as the day of the year. Over the time, the surrogate key value of the single tuple in TODAY remains unchanged, only the value of its DATE attribute is updated daily.

B.14. P-relation for DEPARTMENT

DEPARTMENT_INSTANCE

DEPARTMENT¢	CHAIR¢	NAME	FACLTY

DEPARTMENT is an inner kernel entity, designating entities of type ACADEMIC. The primary key of the relation DEPARTMENT_PROPERTIES is E-attribute DEPARTMENT¢, the corresponding domain is the set of surrogate values currently existing in the E-relation DEPARTMENT. Each tuple contains three properties of a department: a reference to its chairperson, its name (for example, 'CMPT') and the name of the faculty it belongs to (for example, 'Applied Sciences'). The E-attribute CHAIR¢ refers to the E-relation of the ACADEMIC being designated. The domain of E-attribute CHAIR¢ is the set of surrogate values currently existing in the E-relation ACADEMIC, unioned with the null value E-null. The other two property values are stored in the two attributes fields NAME and FACLTY. The attribute NAME forms the user key of relation DEPARTMENT_PROPERTIES. The domain of NAME is 'FIELD', the respective value type is specified as a complete list in Appendix C.2, page 119. The domain of FACLTY is 'FACULTY', the respective value type is specified as a complete list in Appendix C.2, page 119.

B.15. P-relation for ENROLLMENT

ENROLLMENT_INSTANCE

ENROLLMENT¢	CLASS¢	STUDENT¢

ENROLLMENT_GRADE

ENROLLMENT¢	GRADE

ENROLLMENT is an associative entity. The primary key of the relation ENROLLMENT_INSTANCE is E-attribute ENROLLMENT¢, the corresponding domain is the set of surrogate values currently existing in the

E-relation ENROLLMENT. Each tuple of the ENROLLMENT_INSTANCE relation associates a STUDENT and a CLASS, whose respective surrogates are contained in the E-attributes STUDENT¢ and CLASS¢. The domain of E-attribute STUDENT¢ is the set of surrogate values currently existing in the E-relation STUDENT, unioned with the null value E-null. The domain of E-attribute CLASS¢ is the set of surrogate values currently existing in the E-relation CLASS, unioned with the null value E-null. The associative entity type ENROLLMENT is used to represent the fact that a student can be enrolled in several classes, and in each class there can be a number of students.

The primary key of the relation ENROLLMENT_GRADE is E-attribute ENROLLMENT¢, the corresponding domain is the set of surrogate values currently existing in the E-relation ENROLLMENT. The attribute GRADE represents a single valued property: the grade the respective student, enrolled in the respective class, got (for example 'B'). The domain of GRADE is 'GRADE', the respective value type is represented as a complete list in Appendix C.2.

The P-relation ENROLLMENT_INSTANCE does not include the attribute GRADE, since in the micro-world represented, the attribute GRADE can be *inapplicable* to some existing enrollment (associating a particular student and a particular class), and should thus not be represented toghether with those attributes.

B.16. P-relation for FACILITY

FACILITY_INSTANCE

FACILITY¢	OF¢	DIREC¢	NAME

FACILITY is an inner kernel entity, designating entities of type DEPARTMENT and of type ACADEMIC. The primary key of the relation FACILITY_PROPERTIES is E-attribute FACILITY¢, the corresponding domain is the set of surrogate values currently existing in the E-relation FACILITY. E-attribute OF¢ identifies the department to which the facility belongs; it refers to the E-relation of the DEPARTMENT being designated. The domain of E-attribute OF¢ is the set of surrogate values currently existing in the E-relation DEPARTMENT, unioned with the null value E-null. E-attribute DIREC¢ identifies the director of the facility; it refers to the E-relation of the ACADEMIC being designated. The domain of E-attribute DIREC¢ is the set of surrogate values currently existing in the E-relation ACADEMIC, unioned with the null value E-null. The property NAME represent the name of the facility. The domain of NAME is 'FACILITY', the respective value type is represented as a complete list in Appendix C.2.

B.17. P-relation for GRAD

GRAD_INSTANCE

GRAD¢	SUPRV¢	PROG

GRAD is a kernel entity, subordinate to STUDENT, and designating entities of type ACADEMIC. GRAD stands for 'graduate student'. The primary key of the relation GRAD_PROPERTY is E-attribute GRAD¢, the corresponding domain is the set of surrogate values currently existing in the E-relation GRAD (which in turn is a subset of the surrogate values currently existing in the E-relation STUDENT). E-attribute SUPRV¢ identifies the supervisor of the graduate student; it refers to the E-relation of the ACADEMIC being designated. The domain of E-attribute SUPRV¢ is the set of surrogate values currently existing in the E-relation ACADEMIC, unioned with the null value E-null. The attribute PROGRAM represents a single valued property: the program the graduate student is registered in (for example 'PhD'). The domain of PROGRAM is 'PROGRAM', the respective value type is represented as a complete list in Appendix C.2.

B.18. P-relation for INSTRUCTOR

INSTRUCTOR_INSTANCE

INSTRUCTOR¢	ACADEMIC¢	DEPARTMENT¢

INSTRUCTOR is an associative entity. The primary key of the relation INSTRUCTOR_INSTANCE is E-attribute INSTRUCTOR¢, the corresponding domain is the set of surrogate values currently existing in the E-relation INSTRUCTOR. Each tuple of the INSTRUCTOR_INSTANCE relation associates an ACADEMIC and a DEPARTMENT, whose respective surrogates are contained in the E-attributes ACADEMIC¢ and DEPARTMENT¢. The domain of E-attribute ACADEMIC¢ is the set of surrogate values currently existing in the E-relation ACADEMIC, unioned with the null value E-null. The domain of E-attribute DEPARTMENT¢ is the set of surrogate values currently existing in the E-relation DEPARTMENT, unioned with the null value E-null. The associative entity type INSTRUCTOR is used to represent the fact that a particular academic can be appointed as instructor by several departments and each department can appoint a number of academics.

B.19. P-relation for OFFERED

OFFERED_INSTANCE

OFFERED¢	AREA¢	DEPARTMENT¢

OFFERED is an associative entity. The primary key of the relation OFFERED_INSTANCE is E-attribute OFFERED¢, the corresponding domain is the set of surrogate values currently existing in the E-relation OFFERED. Each tuple of the OFFERED_INSTANCE relation associates a DEPARTMENT and a AREA, whose respective surrogates are contained in the E-attributes DEPARTMENT¢ and AREA¢. The domain of E-attribute DEPARTMENT¢ is the set of surrogate values currently existing in the E-relation DEPARTMENT, unioned with the null value E-null. The domain of E-attribute AREA¢ is the set of surrogate values currently existing in the E-relation AREA, unioned with the null value E-null. The associative entity type OFFERED is used to represent the fact that a department can offer courses in several areas (for example, CMPT can offer CMPT, COGS, MACM,...), and in each area can be offered by several departments (for example, COGS can be offered by CMPT, LING, PHIL, and PSYCH).

B.20. P-relation for PRE_REQ

PRE_REQ_INSTANCE

PRE_REQ¢	FOR_COURSE¢	IS_COURSE¢

PRE_REQ is an associative entity. The primary key of the relation PRE_REQ_INSTANCE is E-attribute PRE_REQ¢, the corresponding domain is the set of surrogate values currently existing in the E-relation PRE_REQ. Each tuple of the PRE_REQ_INSTANCE relation associates two entities of type COURSE, whose respective surrogates are contained in the E-attributes FOR_COURSE¢ and IS_COURSE¢. The domain for both E-attributes FOR_COURSE¢ and IS_COURSE is the set of surrogate values currently existing in the E-relation COURSE, unioned with the null value E-null. The associative entity type PRE_REQ is used to represent the fact that a particular course can have several other courses as its pre-requisites, and each course can be a pre-requisite for a number of other courses. Notice, that two distinct names are required for the two distinct associative references, even though both refer to the same entity type.

B.21. P-relation for ROOM

ROOM_PROPERTIES

ROOM¢	BUILDING	NUMBER

ROOM_OFFICE

ROOM¢	OFFICE¢

ROOM is an inner kernel entity, designating entities of type STAFF. Two property relations are used to represent the three properties of a room: ROOM_OFFICE and ROOM_PROPERTIES. The primary key for both property relations is E-attribute ROOM¢, the corresponding domain is the set of surrogate values currently existing in the E-relation ROOM. The E-attribute OFFICE¢ identifies the staff member, for which the room is an office; it refers to the E-relation of the STAFF being designated. The domain of E-attribute OFFICE¢ is the set of surrogate values currently existing in the E-relation STAFF, unioned with the null value E-null. The two properties BUILDING and NUMBER represent the building and the number of the respective room. The domain of BUILDING is 'BUILDING', the respective value type is specified as a complete list in Appendix C.2. The domain of NUMBER is 'room_num', the respective value is 'integer'. The two attributes BUILDING and NUMBER might seem to be a good candidate for a composite user key, however, we want to allow null values in either one of these two attributes. As a result relation ROOM_PROPERTIES has no user key. We use two distinct property relations to represent the three properties of a room, since only a fraction of all the rooms are used as offices. A single property relation representing all three properties would inevitably contain a lot of null values in it's OFFICE¢ attribute.

B.22. P-relation for ROOM_MVP

ROOM_PHONE

ROOM_MVP¢	ROOM¢	PHONE

ROOM_PHONE is a characteristic entity, characterizing ROOM. The primary key of relation

ROOM_PHONE is the E-attribute ROOM_MVP¢, the corresponding domain is the set of surrogate values currently existing in the E-relation ROOM_MVP. The E-attribute ROOM¢ represents the characteristic reference to the ROOM being characterized, the corresponding domain is the set of surrogate values currently existing in the E-relation ROOM. The property attribute PHONE represents the number of one phone (for example '2914302'). The domain of the attribute PHONE is 'NUMBER', the respective values are of type integer.

B.23. P-relation for SCHEDULE

SCHEDULE_INSTANCE

SCHEDULE¢	ROOM¢	TIME_TABLE¢	CLASS¢

SCHEDULE is an associative entity. The primary key of the relation SCHEDULE_INSTANCE is E-attribute SCHEDULE¢, the corresponding domain is the set of surrogate values currently existing in the E-relation SCHEDULE. Each tuple of the SCHEDULE_INSTANCE relation associates a TIME_TABLE and a ROOM, whose respective surrogates are contained in the E-attributes TIME_TABLE¢ and ROOM¢. The domain of E-attribute TIME_TABLE¢ is the set of surrogate values currently existing in the E-relation TIME_TABLE, unioned with the null value E-null. The domain of E-attribute ROOM¢ is the set of surrogate values currently existing in the E-relation ROOM, unioned with the null value E-null. The associative entity type SCHEDULE is used to represent the fact that a room can be occupied at several times of the day, and at each time of the day a number of rooms can be occupied. The E-attribute CLASS¢ identifies the class, for which the particular room is occupied at the particular time; it refers to the E-relation of the CLASS being designated. The domain of E-attribute CLASS¢ is the set of surrogate values currently existing in the E-relation CLASS, unioned with the null value E-null.

B.24. P-relation for SEMESTER

SEMESTER_PROPERTIES

SEMESTER¢	TERM	YEAR

SEMESTER is an inner kernel entity. The primary key of the relation SEMESTER_PROPERTIES is E-attribute SEMESTER¢, the corresponding domain is the set of surrogate values currently existing in the

E-relation SEMESTER. The attribute TERM represents a single valued property: the term of the respective semester. The domain of TERM is 'TERM', the respective value type consists of the three elements 'Spring', 'Summer', and 'Fall' (see Appendix C.2). The attribute YEAR represents a single valued property: the respective year. The domain of YEAR is 'year_num', the respective value type is 'integer'. The two attributes TERM and SEMESTER, together form the composite user key to relation SEMESTER_PROPERTIES.

B.25. P-relation for STAFF

STAFF_PROPERTIES

STAFF¢	DEPT¢	NUMBER	NAME	SEX

STAFF is an inner kernel entity, designating entities of type DEPARTMENT. The primary key of the relation STAFF_PROPERTIES is E-attribute STAFF¢, the corresponding domain is the set of surrogate values currently existing in the E-relation STAFF. The E-attribute DEPT¢ refers to the E-relation of the DEPARTMENT being designated. The domain of E-attribute DEPT¢ is the set of surrogate values currently existing in the E-relation DEPARTMENT, unioned with the null value E-null. The attribute NUMBER represents a single valued property: the instructor's id-number. This attribute forms the user key of relation STAFF_PROPERTIES. The domain of NUMBER is 'ins_num', the respective value type is 'integer'. The attribute NAME represents a single valued property: the instructor's name. The domain of NAME is 'pers_name', the respective value is an array of 60 characters representing one full name. The attribute SEX represents a single valued property: the instructor's sex. The domain of SEX is 'SEX', the respective value type contains two elements: 'Male' and 'Female' (see Appendix C.2).

B.26. P-relation for STUDENT

STUDENT_PROPERTIES

STUDENT¢	NUMBER	NAME	SEX

STUDENT is an inner kernel entity. The primary key of the relation STUDENT_PROPERTIES is E-attribute STUDENT¢, the corresponding domain is the set of surrogate values currently existing in the E-relation STUDENT. The attribute NUMBER represents a single valued property: the student's id-number. This attribute forms the user key of the relation STUDENT_PROPERTIES. The domain of NUMBER is 'stu_num', the respective value type is 'integer'. The attribute NAME represents a single valued property the student's name. The domain of NAME is 'pers_name', the respective value is an array of 60 characters representing one full name. The attribute SEX represents a single valued property: the student's sex. The domain of SEX is 'SEX', the respective value type contains two elements: 'Male' and 'Female' (see Appendix C.2).

B.27. P-relation for TEXT

TEXT_INSTANCE

```
 ____________ ______________ _______________ _
| TEXT¢      | COURSE¢      | BOOK¢          |
|============|--------------|----------------|
|            |              |                |
```

TEXT is an associative entity. The primary key of the relation TEXT_INSTANCE is E-attribute TEXT¢, the corresponding domain is the set of surrogate values currently existing in the E-relation TEXT. Each tuple of the TEXT_INSTANCE relation associates a COURSE and a BOOK, whose respective surrogates are contained in the E-attributes COURSE¢ and BOOK¢. The domain of E-attribute COURSE¢ is the set of surrogate values currently existing in the E-relation COURSE, unioned with the null value E-null. The domain of E-attribute BOOK¢ is the set of surrogate values currently existing in the E-relation BOOK, unioned with the null value E-null. The associative entity type TEXT is used to represent the fact that a book can be used as text-book in several courses, and each course can use a number of books as its text-books.

B.28. P-relation for TIME_TABLE

TIME_TABLE_PROPERTIES

```
 ______________ _______________ _______________ _
| TIME_TABLE¢  | DAY           | HOUR           |
|==============|---------------|----------------|
|              |               |                |
```

TIME_TABLE is an inner kernel entity. The primary key of the relation TIME_TABLE_PROPERTIES is E-attribute TIME_TABLE¢, the corresponding domain is the set of surrogate values currently existing in the E-relation TIME_TABLE. The attribute DAY represents a single valued property: the day of the week (for example 'Monday'). The domain of DAY is 'DAY_OF_WEEK', the respective value type is specified as a complete list in Appendix C.2. The attribute HOUR represents a single valued property the time of the day at which a lecture can begin (for example 15:30). The domain of HOUR is 'hour', the respective value type is given by two 'integers', separated by a ':'. The two attributes DAY and HOUR form the composite user key of the relation TIME_TABLE_PROPERTIES. This relation records all the possible times at which a class could be scheduled. Each semester the courses offered have to be scheduled anew and entered in the database. If a course is scheduled at some 'odd' time, then the user entering the update might notice this, because a separate addition of the 'odd' time will be required in the TIME_TABLE_PROPERTIES relation.

B.29. P-relation for UNDER_MVP1

UNDER_MAJOR

```
 ____________________________________________________________
| UNDER_MVP1¢   | UNDER_GRAD¢   | MAJOR          |
|===============|---------------|----------------|
|               |               |                |
```

UNDER_MVP1 is a characteristic entity, characterizing UNDER_GRAD. The primary key of relation UNDER_MAJOR is the E-attribute UNDER_MVP1¢, the corresponding domain is the set of surrogate values currently existing in the E-relation UNDER_MVP1. The E-attribute UNDER_GRAD¢ represents the characteristic reference to the UNDER_GRAD being characterized, the corresponding domain is the set of surrogate values currently existing in the E-relation UNDER_GRAD. The property attribute MAJOR represents the name of one major field of the undergraduate student (for example 'CMPT'). The domain of the attribute MAJOR is 'FIELD', the respective value type is specified as a complete list in Appendix C.2.

B.30. P-relation for UNDER_MVP2

UNDER_MINOR

```
 ____________________________________________________________
| UNDER_MVP2¢   | UNDER_GRAD¢   | MINOR          |
|===============|---------------|----------------|
|               |               |                |
```

UNDER_MVP2 is a characteristic entity, characterizing UNDER_GRAD. The primary key of relation UNDER_MINOR is the E-attribute UNDER_MVP2¢, the corresponding domain is the set of surrogate values currently existing in the E-relation UNDER_MVP2. The E-attribute UNDER_GRAD¢ represents the characteristic reference to the UNDER_GRAD being characterized, the corresponding domain is the set of surrogate values currently existing in the E-relation UNDER_GRAD. The property attribute MINOR represents the name of one minor field of the undergraduate student (for example 'MATH'). The domain of the attribute MINOR is 'FIELD', the respective value type is specified as a complete list in Appendix C.2.

Appendix C
The catalog for the database: relations and their contents

C.1. DOMAINS

CATLG_DOMAINS

DOMNAME	DATA_TYPE	QUALIFIER	ORDERING
¢	surrogate		NO
cours_num	integer	0 — 999	YES
day_num	integer	1 — 366	YES
hour	real	0.0 — 24.0	YES
name	string	60	NO
number	integer	1 — maxint	NO
pers_name	string	60	NO
p_num	integer	1 — 999999999	YES
room_num	integer	1 — 9999	NO
stu_num	integer	650000000 — 999999999	YES
unit_num	integer	0 — 12	YES
year_num	integer	65 — 99	YES
BUILDING	enumeration	BUILDING	NO
DAY	enumeration	DAY	YES
FACILITY	enumeration	FACILITY	NO
FACULTY	enumeration	FACULTY	NO
FIELD	enumeration	FIELD	NO
GRADE	enumeration	GRADE	YES
POSITION	enumeration	POSITION	NO
PROGRAM	enumeration	PROGRAM	YES
SEX	boolean	SEX	NO
STATUS	enumeration	STATUS	YES
TERM	enumeration	TERM	YES

C.2. Definition of the enumeration types

BUILDING

ELEMENT#	VALUE
1	AQ
2	ASB
3	B
4	C
5	CA
6	CAE
7	CC
8	DEC
9	FLTC
10	GYM
11	IMAGES_TH
12	K
13	LB
14	MMT
15	MPX
16	P
17	PDC
18	THTR

DAY

ELEMENT#	VALUE
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday
7	Sunday

FACILITY

ELEMENT#	VALUE
1	BAMFIELD_MARINE_STATION
2	CENTRE_FOR_ECONOMIC_RESEARCH
3	CENTRE_FOR_PEST_MANAGEMENT
4	CENTRE_FOR_SYSTEMS_SCIENCE
5	CHEMICAL_ECOLOGY_RESEARCH_GROUP
6	CRIMINOLOGY_RESEARCH_CENTRE
7	ENERGY_RESEARCH_INSITUTE
8	GERONTOLOGY_RESEARCH_CENTRE
9	HISTORICAL_RECORDS_INSTITUTE
10	INSTITUTE_FOR_BUSINESS_STUDIES
11	INSTITUTE_OF_FISHERIES_ANALYSIS
12	INSTITUTE_FOR_HUMAN_PERFORMANCE
13	INSTITUTE_FOR_THE_HUMANITIES
14	INSTITUTE_OF_INTERNATIONAL_DEVELOPMENT
15	INSTITUTE_FOR_QUARTERNARY_RESEARCH
16	INSTITUTE_FOR_STUDIES_IN_CRIMINAL_JUSTICE_POLICY
17	INSTRUCTIONAL_PSYCHOLOGY_RESEARCH_GROUP
18	LABORATORY_FOR_COMPUTER_AND_COMMUNICATIONS_RESEARCH
19	NORTHERN_CONFERENCE_RESOURCE_CENTRE
20	PSYCHOLOGY_AND_LAW_INSTITUTE
21	THEORETICAL_SCIENCE_INSTITUTE

FACULTY

ELEMENT#	VALUE
1	APPLIED_SCIENCES
2	ARTS
3	BUSINESS_ADMINISTRATION
4	EDUCATION
5	SCIENCE

FIELD

ELEMENT#	VALUE
1	ARC
2	ATHL
3	BICH
4	BISC
5	BUS
6	BUEC
7	CN_S
8	CHEM
9	CHIN
10	CMNS
11	CMPT
12	CRIM
13	ECON
14	EDUC
15	ENSC
16	ENGL
17	FPA
18	FREN
19	G_S_
20	GEOG
21	GERM
22	GERO
23	GRE
24	HIST
25	HUM
26	KIN
27	LAS
28	LING
29	MACM
30	MASC
31	MATH
32	MSSC
33	NUSC
34	PHIL
35	PHYS
36	POL
37	PSYC
38	RUSS
39	S_A_
40	SPAN
41	W_S_

GRADE

ELEMENT#	VALUE
1	D
2	C
3	B
4	A

POSITION

ELEMENT#	VALUE
1	RECEPTIONIST
2	SECRETARY
3	DEPT_ASSISTANT

PROGRAM

ELEMENT#	VALUE
1	Special
2	MSc
3	PhD

SEX

ELEMENT#	VALUE
1	Male
2	Female

SEX

ELEMENT#	VALUE
1	F
2	M

STATUS

ELEMENT#	VALUE
1	SESS
2	ASSI
3	ASSO
4	PROF

TERM

ELEMENT#	VALUE
1	Spring
2	Summer
3	Fall

C.3. ATTRIBUTE_DOMAINS

ATTRIBUTE_DOMAINS

RELNAME	ATTNAME	DOMNAME	CARDINALITY	EL_KEY	INTRV_KEY
ACADEMIC_PROPERTY	STATUS	STATUS	4	—	—
ADMINISTR_PROPERTIES	POSITN	POSITION	3	—	—
AREA_NAME	NAME	FIELD	41	—	—
BOOK_PROPERTY	TITLE	name	—	—	—
BOOK_AUTHOR	AUTHOR	pers_name	—	—	—
CLASS_PROPERTY	FINAL	day_num	366	—	—
COURSE_PROPERTIES	NUMBER	cours_num	1000	—	—
COURSE_PROPERTIES	UNITS	unit_num	13	—	—
DEPARTMENT_PROPERTIES	FACLTY	FACULTY	5	—	—
DEPARTMENT_PROPERTIES	NAME	FIELD	19	1	—
ENROLLMENT_GRADE	GRADE	GRADE	4	—	—
FACILITY_INSTANCE	NAME	FACILITY	21	—	—
GRAD_INSTANCE	PROG	PROGRAM	3	—	—
ROOM_PROPERTIES	BUILDING	BUILDING	18	—	—
ROOM_PROPERTIES	NUMBER	buil_num	10000	—	—
ROOM_PHONE	PHONE	number	—	—	—
SEMESTER_PROPERTIES	TERM	TERM	3	—	—
SEMESTER_PROPERTIES	YEAR	year_num	34	—	—
STAFF_PROPERTIES	NAME	pers_name	—	—	—
STAFF_PROPERTIES	NUMBER	p_num	999999999	—	—
STAFF_PROPERTIES	SEX	SEX	2	—	—
STUDENT_PROPERTIES	NAME	pers_name	—	—	—
STUDENT_PROPERTIES	NUMBER	stu_num	350000000	—	—
STUDENT_PROPERTIES	SEX	SEX	2	—	—
TIME_TABLE_PROPERTIES	DAY	DAY	7	—	—
TIME_TABLE_PROPERTIES	HOUR	hour	—	—	—
TODAY	DATE	day_num	366	—	—
UNDER_MAJOR	MAJOR	FIELD	41	—	—
UNDER_MINOR	MINOR	FIELD	41	—	—

C.4. ELEMENTS

ELEMENTS

EL_KEY	EL_NUM
1	1
1	4
1	8
1	10
1	11
1	12
1	13
1	15
1	16
1	20
1	24
1	26
1	28
1	31
1	34
1	35
1	36
1	37
1	39

C.5. INTERVALS

INTERVALS

KEY	MIN	MAX

C.6. RELATIONS

CATLG_RELATIONS

RELNAME	RELTYPE
ACADEMIC	EI
ACADEMIC_PROPERTY	P
ADMINISTR	EKD
ADMINISTR_PROPERTIES	P
AFFILIATION	EA
AFFILIATION_INSTANCE	P
AREA	EI
AREA_NAME	P
BOOK	EI
BOOK_PROPERTY	P
BOOK_MVP	EC
BOOK_AUTHOR	P
CLASS	EA
CLASS_INSTANCE	P
CLASS_PROPERTY	P
COMMITTEE	EA
COMMITTEE_INSTANCE	P
COURSE	EID
COURSE_PROPERTIES	P
CURNT_DATE	EI
DEPARTMENT	EID
DEPARTMENT_PROPERTIES	P
ENROLLMENT	EA
ENROLLMENT_GRADE	P
ENROLLMENT_INSTANCE	P
FACILITY	EID
FACILITY_INSTANCE	P
GRAD	EKD
GRAD_INSTANCE	P
INSTRUCTOR	EA
INSTRUCTOR_INSTANCE	P
OFFERED	EA
OFFERED_INSTANCE	P
PRE_REQ	EA
PRE_REQ_INSTANCE	P
ROOM	EID
ROOM_PROPERTIES	P
ROOM_OFFICE	P
ROOM_MVP	EC
ROOM_PHONE	P
SCHEDULE	EAD
SCHEDULE_INSTANCE	P
SEMESTER	EI
SEMESTER_PROPERTIES	P
STAFF	EID
STAFF_PROPERTIES	P
STUDENT	EI
STUDENT_PROPERTIES	P
TEXT	EA
TEXT_INSTANCE	P
TIME_TABLE	EI
TIME_TABLE_PROPERTIES	P

```
| TODAY                  | P          |
| UNDER_GRAD             | EK         |
| UNDER_MVP1             | EC         |
| UNDER_MAJOR            | P          |
| UNDER_MVP2             | EC         |
| UNDER_MINOR            | P          |
-------------------------------------------- -
```

C.7. ATTRIBUTES

CATLG_ATTRIBUTES

RELNAME	ATTNAME	DOMNAME	PKEY	UKEY	NULLS	EXC
ACADEMIC	ACADEMIC¢	¢	YES	NO	NO	0
ACADEMIC_PROPERTY	ACADEMIC¢	¢	YES	NO	NO	0
ACADEMIC_PROPERTY	STATUS	STATUS	NO	NO	YES	0
ADMINISTR	ADMINISTR¢	¢	YES	NO	NO	0
ADMINISTR_PROPERTIES	ADMINISTR¢	¢	YES	NO	NO	0
ADMINISTR_PROPERTIES	FOR¢	¢	NO	NO	YES	0
ADMINISTR_PROPERTIES	POSITN	POSITION	NO	NO	YES	0
AFFILIATION	AFFILIATION¢	¢	YES	NO	NO	0
AFFILIATION_INSTANCE	AFFILIATION¢	¢	YES	NO	NO	0
AFFILIATION_INSTANCE	ACADEMIC¢	¢	NO	NO	YES	0
AFFILIATION_INSTANCE	DEPARTMENT¢	¢	NO	NO	YES	0
AREA	AREA¢	¢	YES	NO	NO	0
AREA_NAME	AREA¢	¢	YES	NO	NO	0
AREA_NAME	NAME	FIELD	NO	NO	YES	0
BOOK	BOOK¢	¢	YES	NO	NO	0
BOOK_PROPERTY	BOOK¢	¢	YES	NO	NO	0
BOOK_PROPERTY	TITLE	name	NO	NO	YES	0
BOOK_MVP	BOOK_MVP¢	¢	YES	NO	NO	0
BOOK_AUTHOR	BOOK_MVP¢	¢	YES	NO	NO	0
BOOK_AUTHOR	BOOK¢	¢	NO	NO	NO	0
BOOK_AUTHOR	AUTHOR	pers_name	NO	NO	YES	0
CLASS	CLASS¢	¢	YES	NO	NO	0
CLASS_INSTANCE	CLASS¢	¢	YES	NO	NO	0
CLASS_INSTANCE	COURSE¢	¢	NO	NO	YES	0
CLASS_INSTANCE	INSTRUCTOR¢	¢	NO	NO	YES	0
CLASS_INSTANCE	SEMESTER¢	¢	NO	NO	YES	0
CLASS_PROPERTY	CLASS¢	¢	YES	NO	NO	0
CLASS_PROPERTY	FINAL	day_num	NO	NO	YES	1
COMMITTEE	COMMITTEE¢	¢	YES	NO	NO	0
COMMITTEE_INSTANCE	COMMITTEE¢	¢	YES	NO	NO	0
COMMITTEE_INSTANCE	ACADEMIC¢	¢	NO	NO	YES	0
COMMITTEE_INSTANCE	GRAD¢	¢	NO	NO	YES	0
COURSE	COURSE¢	¢	YES	NO	NO	0
COURSE_PROPERTIES	COURSE¢	¢	YES	NO	NO	0
COURSE_PROPERTIES	FIELD¢	¢	NO	NO	YES	0
COURSE_PROPERTIES	NUMBER	cours_num	NO	NO	YES	2
COURSE_PROPERTIES	UNITS	unit_num	NO	NO	YES	0
CURNT_DATE	CURNT_DATE¢	¢	YES	NO	NO	0
DEPARTMENT	DEPARTMENT¢	¢	YES	NO	NO	0
DEPARTMENT_PROPERTIES	DEPARTMENT¢	¢	YES	NO	NO	0
DEPARTMENT_PROPERTIES	CHAIR¢	¢	NO	NO	YES	0
DEPARTMENT_PROPERTIES	FACULTY	FACULTY	NO	NO	YES	0
DEPARTMENT_PROPERTIES	NAME	FIELD	NO	YES	NO	0
ENROLLMENT	ENROLLMENT¢	¢	YES	NO	NO	0
ENROLLMENT_GRADE	ENROLLMENT¢	¢	YES	NO	NO	0
ENROLLMENT_GRADE	GRADE	GRADE	NO	NO	YES	3
ENROLLMENT_INSTANCE	CLASS¢	¢	NO	NO	YES	0
ENROLLMENT_INSTANCE	ENROLLMENT¢	¢	YES	NO	NO	0
ENROLLMENT_INSTANCE	STUDENT¢	¢	NO	NO	YES	0
FACILITY	FACILITY¢	¢	YES	NO	NO	0
FACILITY_INSTANCE	FACILITY¢	¢	YES	NO	NO	0
FACILITY_INSTANCE	OF¢	¢	NO	NO	YES	0

```
| FACILITY_INSTANCE            | DIREC¢          | ¢          | NO  | NO  | YES |     0 |
| FACILITY_INSTANCE            | NAME            | FACILITY    NO  | YES | NO  |     0 |
| GRAD                         | GRAD¢           | ¢          | YES | NO  | NO  |  4    |
| GRAD_INSTANCE                | GRAD¢           | ¢          | YES | NO  | NO  |  4    |
| GRAD_INSTANCE                | SUPRV¢          | ¢          | NO  | NO  | YES |     0 |
| GRAD_INSTANCE                | PROG            | PROGRAM    | NO  | NO  | YES |     0 |
| INSTRUCTOR                   | INSTRUCTOR¢     | ¢          | YES | NO  | NO  |     0 |
| INSTRUCTOR_INSTANCE          | INSTRUCTOR¢     | ¢          | YES | NO  | NO  |     0 |
| INSTRUCTOR_INSTANCE          | ACADEMIC¢       | ¢          | NO  | NO  | YES |     0 |
| INSTRUCTOR_INSTANCE          | DEPARTMENT¢     | ¢          | NO  | NO  | YES |     0 |
| OFFERED                      | OFFERED¢        | ¢          | YES | NO  | NO  |     0 |
| OFFERED_INSTANCE             | OFFERED¢        | ¢          | YES | NO  | NO  |     0 |
| OFFERED_INSTANCE             | AREA¢           | ¢          | NO  | NO  | YES |     0 |
| OFFERED_INSTANCE             | DEPARTMENT¢     | ¢          | NO  | NO  | YES |     0 |
| PRE_REQ                      | PRE_REQ¢        | ¢          | YES | NO  | NO  |     0 |
| PRE_REQ_INSTANCE             | FOR_COURSE¢     | ¢          | NO  | NO  | YES |     0 |
| PRE_REQ_INSTANCE             | IS_COURSE¢      | ¢          | NO  | NO  | YES |     0 |
| PRE_REQ_INSTANCE             | PRE_REQ¢        | ¢          | YES | NO  | NO  |     0 |
| ROOM                         | ROOM¢           | ¢          | YES | NO  | NO  |     0 |
| ROOM_OFFICE                  | OFFICE¢         | ¢          | NO  | NO  | YES |  5    |
| ROOM_OFFICE                  | ROOM¢           | ¢          | YES | NO  | NO  |     0 |
| ROOM_PROPERTIES              | BUILDING        | BUILDING   | NO  | NO  | YES |     0 |
| ROOM_PROPERTIES              | NUMBER          | buil_num   | NO  | NO  | YES |     0 |
| ROOM_PROPERTIES              | ROOM¢           | ¢          | YES | NO  | NO  |     0 |
| ROOM_MVP                     | ROOM_MVP¢       | ¢          | YES | NO  | NO  |  5    |
| ROOM_PHONE                   | PHONE           | number     | NO  | NO  | YES |  5    |
| ROOM_PHONE                   | ROOM¢           | ¢          | NO  | NO  | NO  |     0 |
| ROOM_PHONE                   | ROOM_MVP¢       | ¢          | YES | NO  | NO  |     0 |
| SCHEDULE                     | SCHEDULE¢       | ¢          | YES | NO  | NO  |     0 |
| SCHEDULE_INSTANCE            | CLASS¢          | ¢          | NO  | NO  | YES |  6    |
| SCHEDULE_INSTANCE            | ROOM¢           | ¢          | NO  | NO  | YES |     0 |
| SCHEDULE_INSTANCE            | SCHEDULE¢       | ¢          | YES | NO  | NO  |     0 |
| SCHEDULE_INSTANCE            | TIME_TABLE¢     | ¢          | NO  | NO  | YES |     0 |
| SEMESTER                     | SEMESTER¢       | ¢          | YES | NO  | NO  |     0 |
| SEMESTER_PROPERTIES          | SEMESTER¢       | ¢          | YES | NO  | NO  |     0 |
| SEMESTER_PROPERTIES          | TERM            | TERM       | NO  | YES | NO  |     0 |
| SEMESTER_PROPERTIES          | YEAR            | year_num   | NO  | YES | NO  |     0 |
| STAFF                        | STAFF¢          | ¢          | YES | NO  | NO  |     0 |
| STAFF_PROPERTIES             | STAFF¢          | ¢          | YES | NO  | NO  |     0 |
| STAFF_PROPERTIES             | DEPT¢           | ¢          | NO  | NO  | YES |     0 |
| STAFF_PROPERTIES             | NAME            | pers_name  | NO  | NO  | YES |     0 |
| STAFF_PROPERTIES             | NUMBER          | p_num      | NO  | YES | NO  |     0 |
| STAFF_PROPERTIES             | SEX             | SEX        | NO  | NO  | YES |     0 |
| STUDENT                      | STUDENT¢        | ¢          | YES | NO  | NO  |     0 |
| STUDENT_PROPERTIES           | NAME            | pers_name  | NO  | NO  | YES |     0 |
| STUDENT_PROPERTIES           | NUMBER          | stu_num    | NO  | YES | NO  |     0 |
| STUDENT_PROPERTIES           | SEX             | SEX        | NO  | NO  | YES |     0 |
| STUDENT_PROPERTIES           | STUDENT¢        | ¢          | YES | NO  | NO  |     0 |
| TEXT                         | TEXT¢           | ¢          | YES | NO  | NO  |     0 |
| TEXT_INSTANCE                | BOOK¢           | ¢          | NO  | NO  | YES |     0 |
| TEXT_INSTANCE                | COURSE¢         | ¢          | NO  | NO  | YES |     0 |
| TEXT_INSTANCE                | TEXT¢           | ¢          | YES | NO  | NO  |     0 |
| TIME_TABLE                   | TIME_TABLE¢     | ¢          | YES | NO  | NO  |     0 |
| TIME_TABLE_PROPERTIES        | DAY             | DAY        | NO  | YES | NO  |     0 |
| TIME_TABLE_PROPERTIES        | HOUR            | hour       | NO  | YES | NO  |     0 |
| TIME_TABLE_PROPERTIES        | TIME_TABLE¢     | ¢          | YES | NO  | NO  |     0 |
| TODAY                        | CURNT_DATE¢     | ¢          | YES | NO  | NO  |     0 |
| TODAY                        | DATE            | day_num    | NO  | NO  | NO  |  1    |
| UNDER_GRAD                   | UNDER_GRAD¢     | ¢          | YES | NO  | NO  |  7    |
| UNDER_MVP1          | UNDER_MVP1¢   | ¢      | YES | NO  | NO  |  7  |
| UNDER_MAJOR         | MAJOR         | FIELD  | NO  | NO  | YES |    0|
| UNDER_MAJOR         | UNDER_GRAD¢   | ¢      | NO  | NO  | NO  |    0|
| UNDER_MAJOR         | UNDER_MVP1¢   | ¢      | YES | NO  | NO  |  7  |
| UNDER_MVP2          | UNDER_MVP2¢   | ¢      | YES | NO  | NO  |  7  |
| UNDER_MINOR         | MINOR         | FIELD  | NO  | NO  | YES |    0|
| UNDER_MINOR         | UNDER_GRAD¢   | ¢      | NO  | NO  | NO  |    0|
| UNDER_MINOR         | UNDER_MVP2¢   | ¢      | YES | NO  | NO  |  7  |
------------------------------------------------------------------------------
```

Graph relations

C.8. property graph

PROPT_GRAPH

P_RELNAME	E_RELNAME
ACADEMIC_PROPERTY	ACADEMIC
ADMINISTR_PROPERTIES	ADMINISTR
AFFILIATION_INSTANCE	AFFILIATION
AREA_NAME	AREA
BOOK_PROPERTY	BOOK
BOOK_AUTHOR	BOOK_MVP
CLASS_INSTANCE	CLASS
CLASS_PROPERTY	CLASS
COMMITTEE_INSTANCE	COMMITTEE
COURSE_PROPERTIES	COURSE
DEPARTMENT_PROPERTIES	DEPARTMENT
ENROLLMENT_GRADE	ENROLLMENT
ENROLLMENT_INSTANCE	ENROLLMENT
FACILITY_INSTANCE	FACILITY
GRAD_INSTANCE	GRAD
INSTRUCTOR_INSTANCE	INSTRUCTOR
OFFERED_INSTANCE	OFFERED
PRE_REQ_INSTANCE	PRE_REQ
ROOM_PROPERTIES	ROOM
ROOM_OFFICE	ROOM
ROOM_PHONE	ROOM_MVP
SCHEDULE_INSTANCE	SCHEDULE
SEMESTER_PROPERTIES	SEMESTER
STAFF_PROPERTIES	STAFF
STUDENT_PROPERTIES	STUDENT
TEXT_INSTANCE	TEXT
TIME_TABLE_PROPERTIES	TIME_TABLE
TODAY	CURNT_DATE
UNDER_MAJOR	UNDER_GRAD
UNDER_MINOR	UNDER_GRAD

C.9. association graph

ASSOC_GRAPH

ASSOCIATION_ E_RELNAME	ASSOCIATION_ P_ATTNAME	PARTICIPANT_ E_RELNAME
AFFILIATION	DEPARTMENT¢	DEPARTMENT
AFFILIATION	ACADEMIC¢	ACADEMIC
COMMITTEE	ACADEMIC¢	ACADEMIC
COMMITTEE	GRAD¢	GRAD
CLASS	COURSE¢	COURSE
CLASS	INSTRUCTOR¢	INSTRUCTOR
CLASS	SEMESTER¢	SEMESTER
ENROLLMENT	CLASS¢	CLASS
ENROLLMENT	STUDENT¢	STUDENT
INSTRUCTOR	ACADEMIC¢	ACADEMIC
INSTRUCTOR	DEPARTMENT¢	DEPARTMENT
OFFERED	AREA¢	AREA
OFFERED	DEPARTMENT¢	DEPARTMENT
PRE_REQ	FOR_COURSE¢	COURSE
PRE_REQ	IS_COURSE¢	COURSE
SCHEDULE	ROOM¢	ROOM
SCHEDULE	TIME_TABLE¢	TIME_TABLE
TEXT	COURSE¢	COURSE
TEXT	BOOK¢	BOOK

C.10. characteristic graph

CHARC_GRAPH

CHARACTERISTIC_ E_RELNAME	SUPERIOR_ E_RELNAME
BOOK_MVP	BOOK
ROOM_MVP	ROOM
UNDER_MVP1	UNDER_GRAD
UNDER_MVP2	UNDER_GRAD

C.11. designation graph

DESIG_GRAPH

DESIGNATIVE_ E_RELNAME	DESIGNATIVE_ P_ATTNAME	DESIGNATED_ E_RELNAME
ADMINISTR	FOR¢	FACILITY
COURSE	FIELD¢	AREA
DEPARTMENT	CHAIR¢	ACADEMIC
FACILITY	DIREC¢	ACADEMIC
FACILITY	OF¢	DEPARTMENT
GRAD	SUPRV¢	ACADEMIC
ROOM	OFFICE¢	STAFF
SCHEDULE	CLASS¢	CLASS
STAFF	DEPT¢	DEPARTMENT

C.12. subtype graph

SUBTP_GRAPH

SUBTYPE_ E_RELNAME	SUPERTYPE_ E_RELNAME	CATEGORY	SPANNING_ SUPERTYPE	MUTUALLY_ EXCLUSIVE
ACADEMIC	STAFF	JOB	NO	NO
ADMINISTR	STAFF	JOB	NO	NO
GRAD	STUDENT	STATUS	YES	YES
UNDER_GRAD	STUDENT	STATUS	YES	YES

Appendix D
The contents of the knowledge-base

D.1. Exception Rules

[(0) 0 0]

[(1) ((CLASS_INSTANCE.FINAL $\geq$ TODAY.DATE)

$\rightarrow$ (ENROLLMENT_GRADE.ENROLLMENT := $\emptyset$))

(You specified some class(es) for which the date of the final has not passed yet; grades are not available for such classes.)]

[(2) ((COURSE_PROPERTIES.NUMBER $\geq$ 800)

$\rightarrow$((UNDER_GRAD.UNDER_GRAD¢ := $\emptyset$)
$\wedge$ (UNDER_MAJOR.UNDER_GRAD¢ := $\emptyset$)
$\wedge$ (UNDER_MINOR.UNDER_GRAD¢ := $\emptyset$)))

(You specified some higher level course(s); undergraduate students cannot be enrolled in such courses.)]

[(3) (CLASS_INSTANCE.FINAL < TODAY.DATE)

(You specified some class(es) for which the grades are currently available; for such classes the final date must have passed.)]

[(4) ((UNDER_GRAD.UNDER_GRAD¢ := $\emptyset$)
$\wedge$ (UNDER_MAJOR.UNDER_GRAD¢ := $\emptyset$)
$\wedge$ (UNDER_MINOR.UNDER_GRAD¢ := $\emptyset$))

(You specified some graduate student(s); graduate students are distinct from undergraduate students; 'major' and 'minor' are inapplicable to them.)]

[(5) ((SCHEDULE_INSTANCE.CLASS¢ := $\emptyset$)
$\wedge$ (SCHEDULE_INSTANCE.ROOM¢ := $\emptyset$))

(You specified some office(s); offices cannot appear in schedules for classes.)]

[(6) ((ROOM_OFFICE.ROOM¢ := $\emptyset$)
$\wedge$ (ROOM_MVP.ROOM¢ := $\emptyset$))

(You specified some classroom(s); classrooms are not used as offices and have no phones.)]

[(7) $((\text{GRAD.GRAD¢} := \varnothing) \wedge (\text{GRAD\_INSTANCE.GRAD¢} := \varnothing))$

(*You specified some undergraduate student(s); undergraduate students are distinct from graduate students; they have neither supervisors nor committees, and 'program' is inapplicable to them.*)]

Appendix E
Mathematical terms to specify functional relationships

A function f: $D \rightarrow C$ is a right unique mapping of elements of a domain D to elements of a co-domain C. A mapping is right unique if it maps no element of the domain to more than one corresponding element in the co-domain, i.e.,

$$(\forall x \in D, \forall y \in C, \forall y' \in C)\ ((f(x) = y \wedge f(x) = y') \rightarrow (y \equiv y')).$$

Partial means there may be elements of the domain on which the function is not defined.

Total means the function is defined on all elements of the domain, i.e.,

$$(\forall x \in D)\ (\exists y \in C \mid f(x) = y)$$

Surjective means all elements of the co-domain correspond to some function values, i.e.,

$$(\forall y \in C)\ (\exists x \in D \mid f(x) = y)$$

Injective means all function values are left unique, i.e.,

$$(\forall x \in D, \forall x' \in D)\ ((f(x) \neq f(x')) \rightarrow x \neq x')$$

Bijective means both surjective and injective.

Figure E-1 illustrates these terms graphicaly.

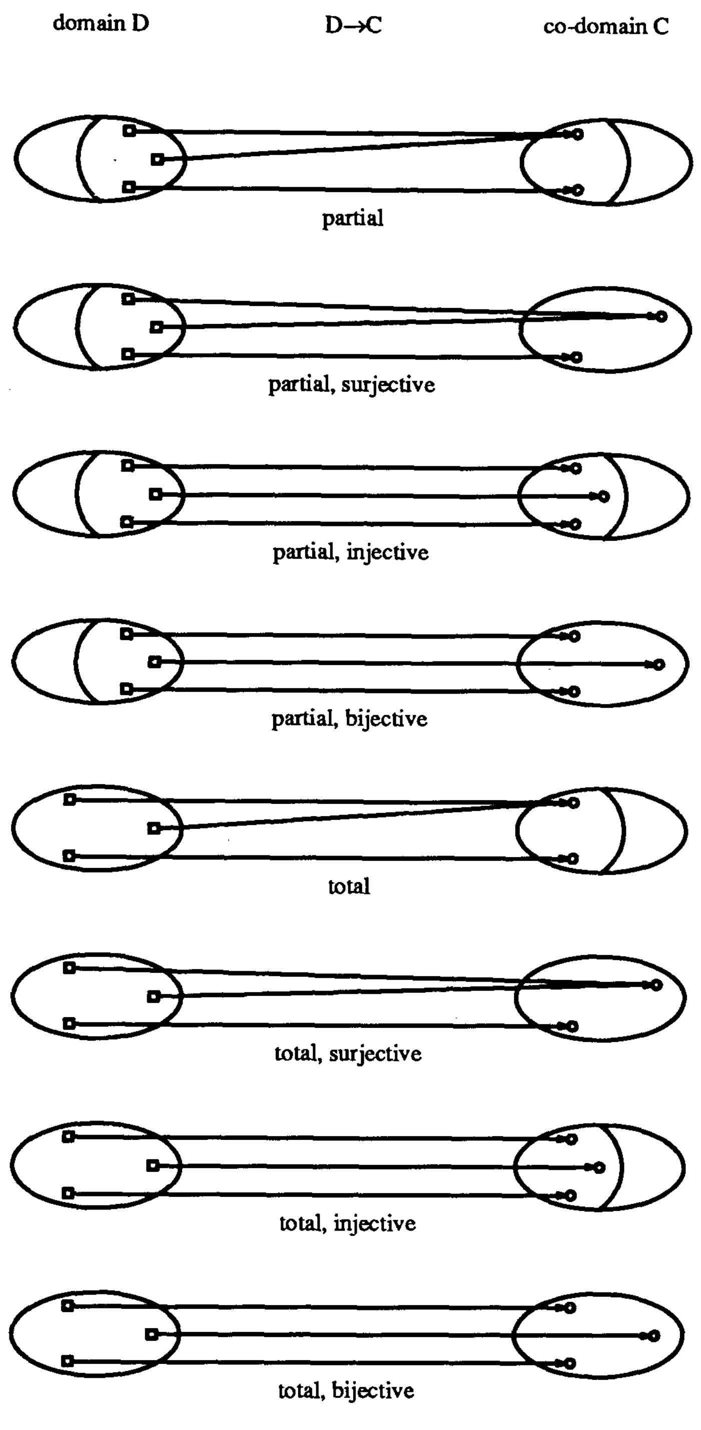


Figure E-1: Graphical illustration of the functional relationships

References

[ANSI 75] ANSI/X3/SPARC Study Group on Data Base Management Systems.
Interim Report 7:2.
American National Standards Committees: X3—Computers & Information Processing, CBEMA, 1828 L St NW (suite 1200), Washington DC 20036, 1975.

[Atzeni and Parker 82]
Atzeni, Paolo and Parker, D. Stott, Jr.
Assumptions in Relational Database Theory.
Association for Computing Machinery :pages 1-9, 1982.

[Belnap 75] Belnap, Nuel D. Jr.
A Useful Four-valued Logic.
Modern Uses of Multiple-Valued Logic.
D. Reitel Pub. Co., 1975.

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

[Cercone et al. 83]Cercone, Nick; Hadley, Robert; Strzalkowski, Tomek.
The Automated Academic Advisor: Introduction and Initial Assessment.
Technical Report TR83-11, LCCR, School of Computing Science, Simon Fraser University, Burnaby, British Columbia, 1983.

[Cercone et al. 84]Cercone, Nick; Hadley, Robert; Martin, Fred; McFetridge, Paul; Strzalkowski, Tomek.
Designing and Automating the Quality Assesment of a Knowledge-Based System: The Initial Automatic Academic Advisor Experience.
In *Proceedings of the Workshop on Principles of Knowledge-Based Systems*, pages 193-204. IEEE Computer Society, 1984.

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

[Codd 86] Codd, Edgar F.
Missing Information (Applicable and Inapplicable) in Relational Databases.
ACM SIGMOD RECORD 15(4):pages 53-78, 1986.

[Codd 87] Codd, Edgar F.
More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information).
ACM SIGMOD RECORD 16(1):pages 42-50, 1987.

[Date 83] Date, Christopher J.
An Introduction to Database Systems.
Addison-Wesley, 1983.

[Date 86a] Date, Christopher J.
Relational Databases: Selected Writings.
Addison-Wesley, 1986.

[Date 86b] Date, Christopher J.
An Introduction to Database Systems.
Addison-Wesley, 1986.

[Grosz 77] Grosz, Barbara J.
The Representation and Use of Focus in Dialogue Understanding.
Technical Report 151, SRI International, Menlo Park, California 94025, 1977.

[Hall 86] Hall, Gary W.
Querying Cyclic Databases in Natural Language.
Master's thesis, Simon Fraser University, October, 1986.

[Jeffrey 81] Jeffrey, Richard.
Formal Logic: Its Scope and Limits.
McGraw-Hill, 1981.

[Kao 86] Kao, Mimi A.
Turning Null Responses into Quality Responses.
Master's thesis, Simon Fraser University, November, 1986.

[Kaplan 79] Kaplan, Samuel J.
Cooperative Responses from a Portable Natural Language Data Base Query System.
PhD thesis, University of Pennsylvania, 1979.

[Martinich 85] Martinich, Aloysius P. (editor).
The Philosophy of Language.
Oxford University Press, 1985.

[McFetridge et al. 88]
McFetridge, Paul; Hall, Gary; Cercone, Nick; Luk, Wo-Shun .
System X: A Portable Natural Language Interface.
In *Proceedings of the CSCSI'88 Conference.* Canadian Society for Computational Studies of Intelligence, 1988.

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

[Rescher 69] Rescher, Nicholas.
Many-Valued Logic.
McGraw Hill , 1969.

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

[Vassiliou 79] Vassiliou, Yannis.
Null Values in Database Management: A Denotational Semantics Approach.
In *Proc. ACM SIGMOD 1979 International Conference on Management of Data*, pages 162-169. Boston, Mass., May, 1979.

[Winograd 83] Winograd, Terry.
Language as a Cognitive Process.
Addison-Wesley, 1983.

[Winograd and Flores 86]
Winograd, Terry; Flores, Fernando.
Understanding Computers and Cognition: A New Foundation for Design.
Ablex Publishing Corporation, Norwood, New Jersey, 1986.