Learning the Structure and Content of an Electronic Medical Record System

by

Jiri Schindler

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Author .... [illegible] ....
Department of Electrical Engineering and Computer Science
May 27, 1997

Certified by .... [illegible] ....
Peter Szolovits
Professor of Computer Science
Thesis Supervisor

Accepted by .... [illegible] ....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Learning the Structure and Content of an Electronic Medical Record System

by

Jiri Schindler

Submitted to the Department of Electrical Engineering and Computer Science
on May 27, 1997, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This report presents a methodology for learning the structure and the content of Electronic Medical Record Systems (EMRS). The described method first learns the structure of the data repository and then tries to identify five elements of medical records - patient demographics, problems, allergies, medications, and visit notes. The identification of the various elements is done by (a) a set of simple heuristics for distinguishing patient demographics and (b) a method of identifying medical terms using UMLS Knowledge Source. It also describes an implementation of the methodology and presents results for data from the EMRS of Children's Hospital in Boston.

Thesis Supervisor: Peter Szolovits
Title: Professor of Computer Science

Acknowledgments

The path from initial, and often incongruous, ideas to the final version of my master's thesis was a long one and included many sidetracks and blind alleys. In the following paragraphs I would like to thank some of the people who helped me stay on the right path and carry through till the very end.

My thesis advisor, Professor Peter Szolovits, has helped me with the choice of the topic and has supplied thoughful and invaluable advice throughout the whole journey. He has been my guide since my sophomore year and served as a role model during my initiation into the field of Medical Computing.

Latanya Sweeney, a graduate student at the Laboratory for Computer Science at MIT, has provided many insights and helpful suggestions throughout the entire process of designing and implementing the thesis. She has also provided several template files without which it would be impossible to run my program.

Eric Jordan and Milos Hauskrecht, both graduate students at the Clinical Decision Making Group at MIT, have on numerous occasions patiently listened to my ideas in progress and helped me to refine them in order to make them fit into the scope of my thesis.

Finally, my family and friends Katrina Van Dellen and Indranath Neogy provided me with moral support at difficult times. Their kind words and understanding made it possible to finish this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Medical records play an important role in providing health care in today's medical practice. Since the introduction of medical records in 19th centrury, when a medical record consisted of few notes about the patient, the medical record has grown up into a detailed description of many aspects of the patient's life ranging from information about patient's social background, through past medical history to a detailed record of laboratory tests and measurements. Today's medical record is not only a detailed record of patient's health, but also a financial record for billing as well as a legal document.

Because of the sheer volume of information that is kept in medical records the use of computers for storing medical records is attractive. Using electronic medical record provides many advantages over traditional paper records such as high avaiability, simultaneous access to the same data, fast retrieval of information, and the possibility of conducting reviews and studies from multiple records.

Despite the attractiveness of storing medical records elecronically, the Electronic Medical Record Systems (EMRS) have been catching up very slowly both with the demand for comprehensive systems and the technology available for building state-of-the-art systems.

1.1 Types of Medical Records

There exist different organizations of medical records. The traditional view of medical record is the patient-oriented medical record in which all data concerning one patient is put together. Using database systems terminology, the access key is the information abouth the patient such as his name, birthday, social security number etc.

The patient-oriented medical record can have several formats. One possibility is chronologically record all events, lab measurements, and other information and keep them in that order within the patient's record. Another possible organization is so called problem-oriented medical record which lists all active and resolved problems of the patient at the front of the medical record. Each item on the problem list then refers to the appropriate documents that are related to that problem.

A computerized medical record is primarily organized in such a way to reflect the organization of the data adopted in the health institution. It has, however, the advantage of creating alternative views of the medical record and accessing the information stored in the record in different ways.

1.2 Existing EMRSs

The development of electronic medical records started in 1960's and the first such system was COSTAR [2] developed at the Laboratory for Computer Science at the Massachusetts General Hospital. The system was written in MUMPS [1] which was specifically designed for COSTAR. The organization of the data in the data repositories is hierarchical. There have been several revisions of the COSTAR since 1969 and several EMRSs developed since then are based on COSTAR such as the EMRS at the Beth Israel Hospital in Boston, MA.

There also exist EMRSs that were developed at different hospitals from the ground up and, like COSTAR, they use a special database tools developed solely for the needs of the EMRS. These are for example HELP system [23], STOR [32], or TMR [25].

Finally there are EMRSs that use commercial database systems on whose top they are built. These systems include the Children's Hospital in Boston EMRS which uses ORACLE RDBMS or the Deaconness hospital that uses Sybase.

1.2.1 Size of EMRSs

Current Electronic Medical Record Systems (EMRS) store considerable amount of information about a patient. In addition to the information relevant to the medical history and current health status of a patient, the systems store information about insurance, billing records for each procedure done in the institution, and increasingly more detailed information about the personnel of the hospital directly or indirectly involved in providing care. As a result of this trend, the size of current EMRS data repositories, in relational database terms, is on the order of hundreds of relational tables each having tens of different fields with the number of records in range of a thousand to a million [11]. Consequently, learning the semantics of each field of such a large database without any a priori knowledge of the data in the database is very difficult, if not impossible.

Despite the overwhelming size of the data repositories, certain fields in the database can be, however, recognized as the key elements of the patient's medical record (such as problems, allergies, and medications) and presented as a partial medical record of a patient. Even though the few identified key elements are not sufficient for creating a complete medical history of a patient, they are invaluable at emergency rooms, where, for example, retrieving patient's allergies to certain medications can prevent complications later and in some situations even death of the patient.

1.2.2 Information Model

The organization of a medical record such as the problem-oriented medical record is, at least to a certain degree, reflected in the information model of the data repository. If the underlying system is a relational database, the information model is called a relational model and it captures the structure and relations among the tables that constitute the database. However even systems that are not based on a relational database system (e.g. [2, 25]) have a notion of structure or hierarchy that enables access to the data in an organized fashion.

The information model is a roadmap to the data repository. Not only does it enable to determine how to access certain information but it also shows all the relations and dependencies of the data stored in the repository. If the roadmap is complete then any information that is available in the repository can be easily obtained no matter how complex the path to the appropriate data is. Even though there are formal methods for designing an optimal information model [6, 17], it is difficult to use them in practice for building complex systems such as an EMRS.

There does not exist any evidence in publicly available literature that EMRSs have formal information models. And even if formal information models exist they are not disclosed as they are considered a proprietary information.

Another characteristics of current EMRSs is an incremental development and upgrades that spread across many years. Instead of rebuilding the system from the ground up when substantial improvements of the sysem are done, the existing systems are extended and new features added to it. As a result the data repository information model is modified several times. However the modifications of the information model are often inconsistent with the original design goals. Consequently, the existing information model is either outdated, because it has not been updated as the system was expanded, or is no longer valid because the organization of the data in the data repository has been completely changed.

For these reasons the information model is not precisely defined and instead only a set of programs that retrieve a specific information from the data repository exists. These programs however only satisfy one function of the information model: how to retrieve information from the repository. They do not provide an overall picture about the complete structure of the data repository. Some EMRSs, for example, consist of hundreds of different programs [2] where each program can access and retrieve particular information, however any attempt to circumvent these programs to obtain data arranged in a different way is an onerous task that requires a programmer who not only can program in the query language of the system, but is also closely acquainted with the hierarchy of the data repository.

The possibility of automatically retrieving an information model from the existing data repository is therefore attractive to the data repository administrators who can learn about the redundancies or inefficiencies in the structure of the data repository and correct it. Finally, the retrieved information model allows access to the data in the data repository without any a priori knowledge of the database structure, which is beneficial to users and programmers of the data repository who have not been involved with the system from it conception and who can then navigate through the data repository in order to retrieve the necessary data.

1.3 Aims of the Research

The primary goal of this thesis is to come up with a method for learning the structure of an EMRS in order to retrieve some data that is available in the data repository whose organization is not known. Current EMRSs are complex and store enormous amount of information. Designing a method to learn the complete structure of an arbitrarily complex system is very hard and is a subject to active research [21]. The proposed method therefore only learns enough information about the structure of the EMRS to be able to locate and retrieve a subset of the information available in the data repository. The subset consists of the five elements of medical records defined by the Boston Collaborative Group for creating a comprehensive medical record based on information retrieved from several health institutions. The elements are:

Patient Demographics - information about a patient such as name, social security number, address etc.

Problems - a collection of medical problems with which the patient was diagnozed.

Allergies - a list of current allergies, reactions to medications etc.

Notes - free-form text that is a written summary of patient's condition upon admission, transfer or discharge.

Medications - a list of drugs a patient is currently using.

The purpose of this work is not to come up with a general methodology for learning concepts from data using unsupervised learning, but rather to identify specific information in the data repository and to enable retrieval of such information from the data repository without knowing the structure of the data repository.

1.4 Guide to This Report

This document is divided into several chapters. Chaper 2 gives an overview of the methodology used for identifying the different elements of medical records. Chapter 3 describes a program that implements the ideas presented in chapter 2. Chapter 4 presents the results obtained by the program running against a subset of data from one EMRS. Finally, chapters 5 and 7 discuss the results and their implications for future research.

Chapter 2

Approach and Methods

2.1 Scope

The underlying data repository systems of various EMRSs used different formats for storing the data. Some of them use commercial database systems [11], while othera use repository systems built from the ground up solely for the purposes of the EMRS. Regardless of the type of the data repository in which the electronic record is stored, the data can be accessed in a systematic way. To provide such access to various data repositories, the Structured Query Language (SQL) was designed [5].

SQL is best suited for accessing relational databases where data is stored in tables. The tables are linked, or related, by common data called keys. However, it is not necessary for the data repository to be a relational database. The SQL language is used merely for convenience and power. Wecan thus operate on the following assumptions:

1. The data repository can accept queries in Structured Query Language (SQL).

2. The program will have read access to all the data in the data repository without any restrictions.

Furtermore, some EMRSs that use a nonrelational database system as its data repository [2, 13] provide an interface that allows to issue queries in SQL [12] and therefore enable to view the data repository as a relational database.

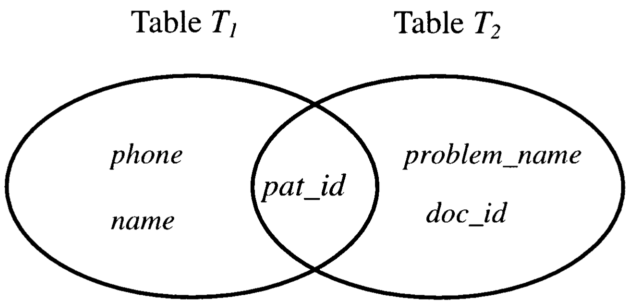
Table T_1

pat_id	*name*	*phone*
111	J. Smith	9996541234
123	A. Johnson	1231231234
⋮		

Table T_2

pat_id	*doc_id*	*problem_name*
123	1231231234	Malaria
999	0034234032	Diabetes
⋮		

(a)



(b)

Figure 2-1: Example of two tables related via a common field *pat_id*.

2.2 Learning the Information Model

A relational data repository, or a non-relational data repository that accepts queries in SQL, is organized in one or more tables, where each table contains several fields with different data. The tables are linked together via common columns/fields, called keys which are contained in both tables. To retrieve data from both tables, the tables are "joined" together through the key and the appropriate information is retrieved from both tables simultaneously.

The starting point for learning the information model identification of all relevant tables that constitute the EMRS and the retrieval of the column identifiers/names. The next step is the identification of the fields that give the relations among the tables. The fields are simply intersections of the field sets of each table, where a field set contains all field names for a given table.

If we consider table T_1 with fields *pat_id, name*, and *phone* and table T_2 with fields *pat_id,doc_id,* and *problem_name* in figure 2-1, then the intersection $T_1 \cap T_2 = pat\_id$ which means that the table T_1 is related to T_2 via the field *pat_id.*

Once all the relations among the tables are determined by identifying the keys of all the tables of the database, we can construct an information model consisting of all tables and relations among them.

The information model of the database can be represented by a Venn's diagram as depicted in figure 2-1 (b). Each set represents a table and the elements of the set are the fields of the table. The intersections of the sets are the fields which provide the relation between the two tables represented by the sets. Using our example, the intersection of T_1 and T_2 is the field *pat_id.*

Identifying Keys by Similar Data

To identify the keys of the database we can systematically go through all tables and its fields and compare the data in the field with every other field of the remaining tables of the database. A field whose data are subset of some other field in any of the remaining tables is theoretically a key for one of the tables.

Using this approach, however, some fields can be incorrectly identified as keys, even though in reality they are not. Suppose there are two numerical fields in two different tables where one field contains unique identifiers of visit notes, represented as 10-digit numbers, and the other field contains telephone numbers of patients written for compactness as 10-digit numbers with no delimiters. If some of the telephone numbers were identical to the visit note identifiers, they would be incorrectly marked as a relation between the two tables even though these two fields contain very different information as illustrated in figure 2-1 by the fields *phone* of table T_1 and *doc_id* of table T_2. We should not therefore solely rely on this technique to identify the keys.

```
foreach table $T$ in database do
    foreach field $F$ in $T$ do
        data = $retrieve\_records(T[F])$
        if $is\_demographics$(data) then
            $add\_demographics$(demogr,$T[F]$,type,score)
        if $is\_visit$(data) then
            $add\_visit$(visits,$T[F]$,score)
        if $is\_problem$(data) then
            $add\_problem$(problems,$T[F]$,score)
        if $is\_medication$(data) then
            $add\_medication$(medications,$T[F]$,score)
        if $is\_allergy$(data) then
            $add\_allergy$(allergies,$T[F]$,score)
$choose\_demographics$(demogr)
$choose\_visits$(visits)
$choose\_allergies$(allergies)
$choose\_medications$(medications)
$choose\_problems$(problems)
```

Figure 2-2: Basic algorithm for identifying data elements of the EMRS.

Identifying Keys by Field Names

To avoid incorrectly identifying some of the fields as keys, we can choose another method for locating the fields that make up the relations between two tables such as a comparison of the field name, type, and length. If the name and the field type (i.e. string or number) match then the fields may constitute a relation. This is demonstrated by the field *pat_id* in figure 2-1.

This method however does not identify all cases. Suppose the field *pat_id* in T_2 were labeled as *pid*. This field then would not be identified as a key of T_2. Nevertheless, it is a useful technique for identifying some of the keys of the tables of EMRS. Since the relations between them are designed by humans who tend to use the same mnemonic identifiers for the same data.

Combination of Both Techniques

Neither of the two techniques described above works well by itself. Combining both of them will, however, identify all true relations since both the field type and the data contained in those fields will match. Combination of those two techniques may still not identify all keys in the database, but the identified fields are guaranteed to be the keys of the appropriate tables.

2.3 Identifying Elements of Medical Records

The second main objective of this research is to identify the elements of the medical record which include the patient demographics, allergies, problems, medications, and visit notes as stated in section 1.3. We can systematically go through all fields in each table constituting the EMRS and by analyzing the data in that field we can identify it as the appropriate element of the medical record. The basic algorithm for identifying the elements is given in figure 2-2.

The algorithm goes through all fields of all tables and attempts to identify the data as one of the data elements of the medical record. This identification results in a confidence score for each medical record element. If the score for the particular element of the medical record is above a certain threshold, the field and its confidence score is added to the list of potential candidates for the respective data element.

Once all potential fields are identified, the procedure *choose* goes through the list of candidates and select one field that best resembles the respective element. The heuristics for each data element differs depending on the element being identified as described bellow in more detail.

2.3.1 Demographics

Patient demographics contains information about a patient such as first and last name, address, phone number etc. The information can be either stored in one field that contains all names, such as first name, middle name, and last name, or can be broken in separate field. Likewise a home address can be in one field or broken into fields such as street, city, state, and zip code.

To identify patient's name we can extract from the database certain number of distinct records and compare those against a set of common American first and last names. We can count the number of matches relative to the number of the records and report the confidence score.

Similarly, we can identify the address field. Addresses contain well-identifiable parts such as the words street and avenue or their acronyms. Identifying the town field, if it is a separate one from the address field, can be done by partial matches to some most common names of towns. These common names include the strings -ville (Greenville), -field (Springfield), -town (Yorktown) etc. Again, we can compute the confidence score as the number of matches divided by the number of records examined.

Finally, to identify numerical elements such as zip code or social security number (SSN) we can match the records against a pattern. In case of zip codes, the records are most likely five-digit numbers or nine-digit numbers optionally separated by a dash. For SSN we can look for a 9-digit number. In order to increase sensitivity of the zip code match, we can exploit some additional characteristics. For example most of the patients live in one region and thereofre their zip codes have several digits that are the same.

2.3.2 Visit Notes

Visit notes are usually one-page-long free text documents that are a part of the medical record. Even though the information contained in the notes is not very structured and therefore hard to analyze, the notes provide an important insight to patient's medical history. Some early electronic medical records were centered around notes and the data repository just provided a convenient storage and retrieval of these notes without much additional information such as laboratory measurements or medications [7].

There are several possibilities how visit notes can be stored. The text can be contained in the database itself, in which case we can just look for fields that contain free text, or the database can contain just a reference to the individual notes that are stored somewhere else. This reference can be in a form of directory path and a filename for example.

Even though visit notes are just a free text, they have fairly rigid format. They contain, in addition to the note body, date, salutation, the name of the patient, and signature of a doctor as described in [27]. Recognizing these parts helps distinguish them from other possible elements of medical record that are also kept as free text such as medical history or history of present illness [24].

2.3.3 Allergies, Medications, and Problems

One of the aims of a computerized medical record is to store the medical history of the patient in a highly structured way, so that the data can be used for research or analysis later [7]. Keeping allergies, medications or problems in separate fields, as oppose to free text form, enables to structure the medical record to provide the desired functionality. If the fields that contain these elements of the medical record are present in the EMRS they can be recognized using the Unified Medical Language System (UMLS) [14] from the National Library of Medicine. Section 2.3.4 describes the system in more detail.

We can extract the records of the field we want to identify from the database and look up each record individually in the UMLS to identify it as a medication, allergy or a problem. To compute the confidence score we can relate the number of successfully identified terms to the total number of records used. For example, the record "James", which could be patient's name would not be found in UMLS since it is not a medical term, whereas "hypertension" would be identified in UMLS as high blood pressure which is a disease.

Associated Attributes

Structured electronic medical records with medications, problems, and allergies stored in separate fields usually contain in addition to those elements associate information or attributes. The attributes of one EMRS element, however, differ across several EMRS. One hospital might include a pharmacy information as a part of the EMRS, in which case many attributes are associated with medications in the EMRS, whereas some other hospitals have a separate pharmacy information system that may or may not be linked with the EMRS.

Because of the difference in quantity and quality of the attributes across different EMRSs, it is difficult to come up with a general procedure that would identify all the attributes in different EMRSs. Instead we can only look for some attributes that are perhaps common to some EMRSs. The common attributes are listed bellow.

Medications

Drugs with which a patient is or has been treated, constitute patient's medication list. In addition to the name of the medication an EMRS may store several associated attributes with the field such as the date of administration of the drug, duration of the treatment, dosage, price etc. The level of detail, however, varies greatly among various EMRSs and it is therefore difficult to identify all the attribute fields that are associated with the medications. For example [11] contains a table which stores medications with 20 separate attribute fields.

Despite the varying level of detail in associated attributes, we can attempt to identify some of the attributes such as the date of administration and dosage. Dosage information captures the unit size of the drug (e.g. milligrams, milliliters), type of administration (infusion, tablets etc.), or combination of both. We can thus look for these patters in the fields.

Allergies

Allergies usually denote medications to which a patient is allergic Although some allergies are temporary, most of them are persistent. Therefore some medical records do not store any other information besides the name of the medication to which the patient is allergic. Identifying this associate attribute, or more precisely the failure to identify any, thus helps distinguish allergies from medications.

Problems

Problems in medical records are often organized into patient's problem list which systematically presents all past and present problems of the patient. The ordering can be either chronological or the list can be divided into current and resolved problems.

Because of the different ways in which the problem list can be presented, there are stored one or several associated attributes along with the name of the problem such as status (resolved, current), date of discovery of the problem or date of resolution of the problem. As with medications, the amount of associated information greatly varies among different EMRSs and it is therefore hard to identify other attributes beyond the ones mentioned above.

2.3.4 UMLS

As the Unified Medical Language System plays an important role in identifying the elements EMRS the following paragraphs give a brief description of the system with the emphasis on the parts relevant to identifying problems and medications.

The UMLS Knowledge Source was created originally to unify medical terminology and to provide translations between recognized standardized medical vocabularies such a SNOMED, MeSH, and ICD-9 [14]. Since then it was extended from the original metathesaurus, which includes 30 different standard medical vocabularies (1996 edition), by a semantic network tool and the SPECIALIST lexicon. The semantic network tool identifies semantic relations between medical terms and SPECIALIST provides the lexical information needed for the SPECIALIST natural language processing system intended a s a general English lexicon that includes many biomedical terms [18].

Metathesaurus is a dictionary that combines the functions of a thesaurus as well as of a translation dictionary. It provides definitions of the terms included in the standardized vocabularies, translations among the terms in various dictionaries, synonyms, and additional information which is used by the other parts of the UMLS such as preferred terms, related concepts etc.

The 1996 edition of the semantic network recognizes 135 semantic types and 51 different relationships. The semantic types are organized into several groups and each group is structured into a multiple-level hierarchy. For example the drug Aspirin has the following hierarchy: Entity—Physical Object—Substance—Chemical—Chemical Viewed Structurally— Organic Chemical.

Using the information provided by the metathesaurus and the semantic network we can identify problems and medications fields of the EMRS. The records of the field under examination can be looked up in metathesaurus to obtain the semantic types. For example Aspirin and Influenza which is a medication and a problem respectively, the UMLS semantic network would return two distinct semantic categories each with its own hierarchy. Aspirin is a Substance whereas Influenza is a Phenomenon or Process with a complete hierarchy of Phenomenon or Process—Natural Phenomenon or Process—Pathologic Function—Disease or Syndrome. Because the two terms have different semantic types each having a different root we can determine the type of the field.

2.4 Difficulties and Alternative Approaches

The method for identifying various elements of the EMRS as described in section 2.3 assumes that the elements in the data repository are stored in a recognizable format, so that if the the field containing the appropriate element such as problems or medications is present in the data repository, it will be correctly identified as such. However, because there are many different EMRSs, it may be difficult to recognize the basic elements of the medical record as they may be stored in some cryptic or coded way. Furtermore the format doe storing the information may be diametrically different from the assumed organization of the database into related tables with different fields.

Another difficulty is the use of specialized vocabularies for storing medical information. For example [11] and [23] use specialized vocabularies for recording problems, while [25] uses codes for describing problems, laboratory measurements, and other attributes. However, despite these difficulties, it is possible to locate the elements of the medical record. If there exists a table that provides expansions of the codes into UMLS recognizable format, the method will identify the appropriate field in the database and, thorough the knowledge of the data repository information model, the information is related to the codes.

It is important to realize that the method outlined in section 2.3 does *not* guarantee to correct identification of the elements in the data repository even though the elements may be present. It only makes best effort in identifying the field(s) that correspond to the appropriate element, but it may identify the wrong field or not find one at all.

2.4.1 Alternatives to Data Repository Organization

While many data repositories of EMRSs are either directly organized into tables or can be viewed as such (for example [11, 2, 23]), there exist EMRSs whose data repositories are organized differently.

The STOR information system [32], for example, uses text files for storing the information while the Regenstrief Medical Records system [16] is essentially one table with three columns being the primary key, an attribute, and value. The attribute column contains some predefined values such as Problem, Laboratory Value, or Medication and the value column is the appropriate value of that attribute.

While it is difficult to view the text file-based data repository as a relational database, the former type of the EMRS based on the three-column table can be transposed into a structure suitable for the approach outlined in section 2.3. We can obtain a subset of the data by using only one attribute with all appropriate values. The data can be then accessed such that the rows become fields and the data, and the records of that field are the values stored in the third column of the original table. By performing this transformation, we can examine the data by the methods described in section 2.3 and identify the elements common to other EMRS. In the SQL terminology, we will be performing SELECTs on the rows of the original database as oppose to the columns which are used in conventional tables.

Chapter 3

Implementation

To test the ideas presented in chapter 2, a prototype program for learning the structure and content of an electronic medical record was implemented in OraPERL running under UNIX operating system. In its present form, the program can access an ORACLE database, learn the structure of the database, identify the five elements of medical record together with some additional attributes, and generate a file with SQL commands in a canonical form that allow to retrieve the five elements with its associated attributes from the data repository.

3.1 Choice of Programming Language

After considering several possible programming languages in which the program could be written, PERL version 5 was chosen as the best candidate for its ease of use, flexible build-in data structures, and powerful text manipulating capabilities. Among the programming languages that were considered was C, Java, and Objective C.

Of all languages mentioned above, C offers the best runtime performance. There also exists ProC library which provides connectivity to an ORACLE database. The disadvantage of C is that many higher level data structures such as hash table, lists etc. would have to be written from scratch. Also C does not offer much flexibility in text manipulating.

Java is attractive for its portability, availability of built-in classes for string manipulations, networking, and connectivity interface to several databases using JDBC [20].

Finally Objective C was considered because it combines both the speed of C and the connectivity to several databases similar to Java's JDBC. The connectivity to databases is provided via Enterprise Object Framework (EOF) which allows an object-oriented view of the database as well as a powerful and easy-to-use API to different RDBMSs. Unfortunately EOF was not available to the author in time to implement the program.

PERL

The Practical Extraction and Report Language (PERL) [31] is a programming language that offers a rich set of functions for easy text manipulation and advanced UNIX programming such as process control and networking. PERL programs run as shell programs, however they are first compiled into an intermediate form which is then interpreted as each statement is run. PERL is also attractive because of automatic garbage collection and availability of some powerful data structures.

Apart from scalar variables, which are either numbers or strings, PERL has lists and associative arrays. Arrays are completely interchangeable with lists and in fact arrays are implemented as lists. Associative arrays are hash tables with keys being a scalar variable and values any other PERL data structure including another associative array. Because associative arrays are hash tables, the lookup time for any value is constant. They are therefore a suitable data structure for storing intermediate values that are retrieved from the EMRS data repository. Associative arrays are used frequently in the program and some of the more important data structures built on top of them are described in the appropriate sections bellow.

There exist several extensions in PERL, called packages, that provide additional functionality. One of the packages is OraPERL which allows access to ORACLE databases. The OraPERL package is built on top of PERL's DB module using ORACLE SQLNet for accessing ORACLE server.

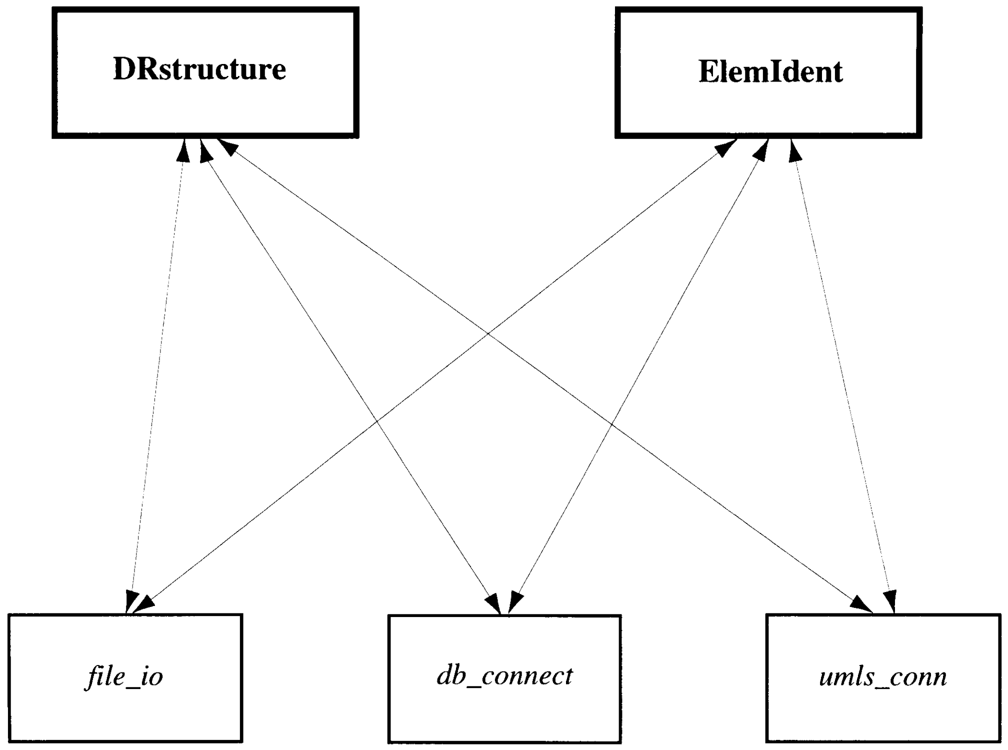


Figure 3-1: Modules of the program for learning the structure and content of EMRS.

3.2 Program Overview

The implemented program is divided into several functional and logical modules as depicted in figure 3-1. The two main logical modules are **DRstructure**, which learns the structure of the EMRS data repository, and **ElemIdent** which takes the information about the structure acquired by the **DRstructure** module and identifies the different data elements of the medical record.

Both logical modules use the **file_io**, **db_connect**, and **umls_conn** modules that abstract the details of file management and the connectivity to the data repository and the UMLS Knowledge Source server. Specifically, even though the module **db_connect** currently only communicates with ORACLE databases, this fact is irrelevant to the **DRstructure** and **ElemIdent** modules as the **db_connect** module provides this abstraction for them.

The **file_io** module contains functions that are used for reading and writing all files that are generated or used by the program. The **db_connect** module provides connectivity to the data repostory, and the **umls_conn** module implements the interface to the UMLS Knowledge Source server.

The communication between the logical modules and the functional modules is done through global data structures. The two logical modules communicate through files that are created by the **DRstructure** and read by the **ElemIdent** module. The files include information about the attributes of the fields, such as names, data types, and lengths, table relations, and keys that constitute the relation.

The modules depicted in figure 3-1 correspond to the source files of the program. There are also two additional files that do not correspond to any modules – `config.pl` and `utils.pl`. The former file contains general variables that are used as parameters to the program such as how many records are retrieved from the data repository for identification. The latter file contains additional functions such as breath-first search and various conversion functions. A complete listing of the source code appears in appendix A.

3.3 Learning the Data Repository Structure

The learning of the data repository structure is implemented in the **DRstructure** module and currently only supports connectivity to an ORACLE database as described in section 3.1. Even though the implementation of **db_connect** uses some features specific to ORACLE, the **DRstructure** module itself does not require any additional features that are not supported by standard SQL.

The functionality of the module **DRstructure** can be summarized into the following steps:

- All table names that constitute the data repository are retrieved via the procedure `GetTableNames` of the **db_connect** module and stored in a global array `table_names`.

- For each field of every table in the `table_names` list the name, datatype, and length of the field is identified. The acquired information is stored in the associative array `all_tables`.
- Using the information stored in `table_names` and `all_tables`, the procedure `FindRelations` identifies the fields that make the relations among tables. The procedure identifies relations based both on the same field names as well as data content and stores the information in the associative array `relations`.

3.3.1 Table Names

The procedure `GetTableNames` retrieves the names of tables via the select statement

SELECT table_name FROM all_tables WHERE owner=*DRowner*

which returns from the table *all_tables* all the names of the tables that make up the the data repository of the EMRS. The table *all_tables* is an administrative table which contains the names of all tables that are stored in the RDBMS. The selector WHERE is necessary to limit the tables to the ones belonging to the data repository only as the RDBMS can contain databases other than the EMRS. The table names are stored in the array `table_names` and written into a file `table_names.txt` for later use.

3.3.2 Field Descriptions

Once all the names of the tables constituting the EMRS are retrieved, the program identifies all the fields for every table. This is accomplished in the procedure `GetDescriptions` of the **db_connect** module by issuing the SQL command

SELECT column_name,data_type,data_length FROM all_tab_columns
WHERE table_name=*tname*

for every table in the `table_names` array. The SELECT command retrieves from the table *all_tab_columns* all records containing the appropriate attributes of the fields. The table *all_tab_columns* is another administrative table of the ORACLE that contains the field names, together with other attributes, of all tables in the RDBMS.

The retrieved information is stored in the associative array `all_tables` which is a hash of a hash of an array, where the array stores the retrieved attributes. The first hashing is done on the table name and the second hashing is done on the field name. This double hashing allows easy retrieval of all fields for a given table as well as fast, constant time, lookup of the attributes for a given table and field. In addition to the data type and field length attributes, the position of the field in the table is stored in the array. The first field of a given table has position one, the second one two etc. The information stored in `all_tables` is also copied into the file `all_tables.txt` for later use in the **ElemIdent** module.

Even though the procedure `GetDescriptions` uses features specific to the ORACLE RDBMS to obtain the field attributes such as the type and length, this information could be obtained by another method that is universal to any database accessible vial SQL. For example one could retrieve several records for each field and identify the datatype by regular expression matching. With regular expressions we can easily recognize numerical, string and date fields which correspond to the NUMBER, CHARACTER, and DATE field types obtained from the ORACLE administrative table *all_tab_columns*. Moreover the length and nullable attributes that are obtained from *all_tab_columns* are not used in the program anyway. Using the *all_tab_columns* instead of the more general principle only gives more convenience and takes less time that having to identify the type of each field of every table by matching the retrieved records.

3.3.3 Finding Relations

The relations between tables are identified by the procedure `FindRelations` which implements the method described in section 2.2. The procedure goes through each field of every table and compares the fields names and types of all the fields of the remaining tables in the data repository. If they are identical then a match is recorded in the `db_keys` and `relations` data structures.

The comparison based on the data is done by the `compare_records_by_data` procedure. The procedure performs a select statement that returns all distinct values occurring in both fields under examination. If the count of the obtained records is higher than a threshold, currently set to the 40% of the records in the field with fewer records, the fields are identified as keys and the relation is recorded in the `relations` data structure.

The `relations` data structure is a hash of a hash of an array. The first hash table uses as a key the name of the first table under examination, the second hash table the name of the second table, and the array stores the following information:

- the name of the field in the first table
- the name of the field in the second table
- flag if the relation has been identified by the field name matching
- flag if the relation has been identified by the comparing the data
- confidence score of identification by data comparison (0-100)

If the relation has been identified by matching the field names of the two tables then both field names will have identical values, and the name relation flag will be set to one. All other values of the array will be zero.

The data structure `db_keys` is also a doubly hashed table where the first hashing is done on the field name and the second on the table name. Thus when a relation is identified, two entries are added to the `db_keys` array.

The `db_keys` data structure only reflects the relations based on identical field names and is used primarily for convenience to provide easy lookup of all table names for a given field name. It also allows to easily count the number of tables for which a specific field is a key.

Because both data structures are necessary for the **ElemIdent** module, the content of the data structures is written into the `table_relations.txt` file, which records all the relations from the `relations`, and into the `name_keys.txt` file, which stores the values from the `db_keys` array.

3.4 Identifying Elements of Medical Records

The fields that store the elements of the medical record described in section 1.3 are identified in the **ElemIdent** module. This module implements the algorithm described in figure 2-2. Even though the algorithm runs sequentially, it is possible to run some parts of it in parallel. Specifically, once the appropriate records for a given field are retrieved from the data repository, the identification of the field by the procedures from the algorithm in figure 2-2 *is_visit*, *is_demographics* etc. can be run in parallel.

The initial version of the program exploited this parallelism and implemented it using the only available mechanism for parallelization in PERL - forking [26]. The implementation of the program using *fork* was eventually dropped because of poor performance and the original serial algorithm was used. Spawning several new processes for each field of every table created a big overhead since upon issuing the *fork* system call all the global data structures are copied into the new process.

3.4.1 Overview

The module **ElemIdent** first loads all the information from the text files that were created by the module **DRstructure** into the appropriate data structures. The data structures in the module **ElemIdent** have identical names to the ones used in **DRstructure**. Once all the necessary text files are read and the information put into the data structures, the program tries to identify the fields that contain the elements of the medical record.

The program calls the procedure `TagFields` which goes through each field of every table and tries to identify the appropriate fields. The possible candidates for the demographics, problems, medications/allergies, and notes are stored in the `dem_relev`, `med_scores`, `problem_field`, and `notes_field` arrays respectively.

The procedure `TagFields` augments the array of the `all_tables` data structure with the following tags. The fifth array element denotes the type of demographics, the sixth array element is the confidence score for the given demographics type, the seventh array element is the confidence score for the problems, the eighth array element the score for medications and finally the ninth array element is the score for notes. Since the data retrieved from each field of every table is inspected as a possible candidate for all the five elements described in section 1.3, it is possible that multiple scores will be nonzero. The score can have values anywhere between 0 and 100 where 0 means that the field was not identified as the respective element of the medical record and 100 means that all records retrieved from the field have been positively identified. Any other value denotes the percentage of records that have been positively identified.

The most probable candidate for each element is then selected in the procedure `SelectCandidates` which goes thorough all the elements of the `all_tables` associative array and selects the candidates based on the scores.

Finally when the most probable candidates are selected, if any were identified at all, the results are written in a form of five SELECT statements that allow to retrieve the appropriate element from the database.

3.4.2 Demographics

The procedure `IdentifyDemograph`, called from `TagFields`, currently recognizes six different types of demographics information - first name, last name, address, city, and zip code. The zip code is identified by matching it to a pattern of five or nine digits, where the first five and last four digits can be separated by a dash.

The other five demographics types are recognized using patterns or template files which contain most common instances of the appropriate demographics type. For example there are two files that contain each approximately 1200 male and female first names and a file that contains 730 most common surnames. The template file for the states contains the names, abbreviations, and two-letter codes for the 50 US states. The template file for the city names contains 11 endings appearing in the names of many American towns such as *-ville* or *-ton.* The addresses are identified by a template where address contains the number, name, and an abbreviation or a full word denoting street, avenue etc.

Using the template files and the regular expressions for pattern matching the procedure `IdentifyDemograph` does the following for each field of every table:

1. Take successively each retrieved record from the given field and match it against every template file and regular expression to identify the zip, address, first name, and last name. If a match is found, increase the respective score by one.
2. After all records have been used, normalize each score. This is done by dividing the number of matched records, the score, by the total number of records inspected and then multiplying by 100 to obtain the percentage.
3. Select the highest score among the scores for zip, address, name etc. and tag the appropriate element in the `all_tables` with the type of the demographics and the confidence score.

Selecting the Best Candidate

When all the fields in `all_tables` have been tagged, the procedure `SelectCandidates` goes through each field of every table and chooses the table which has the most fields tagged. This is done by the procedure `dem_accn` which simply counts the number of identified demographics fields and stores the result in the associative array `dem_relev`.

Even though the table with the highest number of demographics has been selected and recorded in `dem_relev`, the table can contain multiple fields of each demographics type. For example there can be multiple addresses in one table. The addresses can be for instance an address of the patient, next of kin, or patient's previous address. This situtation can occur when the data repository is "flat", i.e. when there is only one table in the data repository and that table contains all the information of the EMRS.

To identify the address of the patient the procedure `resolve_addr` performs the following steps to distinguish the address of the patient from the other addresses:

1. Find the primary key of the database if such a key exists. The primary key is located by the procedure `resolve_primkey` which looks through the `db_keys` array to locate the field that is the key to the most tables. In a patient-oriented medical record, the primary key is likely to be some patient identifier.
2. Go through all the fields of the table identified in `dem_relev` that are tagged as demographics types and select a field for each demographics type that is the

least number of columns away from the key identified by the `resolve_primkey` procedure.

3. Store the results in the array `dem_proximity`.

The method outlined above does not guarantee that the right fields will be identified. Therefore it is possible to identify the wrong fields as the patient demographics.

3.4.3 Visit Notes

As currently implemented, the visit notes are identified only based on the length of the data stored in the data repository. The method therefore positively identifies the visit notes EMRS element only if the content of the notes is stored in the database. The alternative methods described in section 2.3.2 such as looking for pointers to files or recognition of the content of the text have not been implemented.

3.4.4 UMLS Knowledge Source Server

As described in chapter 2 the medications and problems are identified via the UMLS Knowledge Source server. Because the methodology for both elements is almost identical, the lookup of the terms in the UMLS server is implemented as one procedure that identifies both the medications and problems.

The lookup of the records for each field, implemented by the procedure `IdentifyPM`, happens in two steps. The first step takes the retrieved record and runs it against the UMLS metathesaurus. The second step takes the semantic type of the term obtained from the UMLS metathesaurus and retrieves the ancestor list for it using the semantic network. If the record is not a valid term that is defined in the metathesaurus, the second step is not performed.

Lookup in Metathesaurus

To lookup a term in the metathesaurus, the procedure `IdentifyPM` first gets rid of some extraneous information from the term so that it can be identified by the UMLS metathesaurus. Specifically, units and dosage information that may occur together with the name of the drug are removed. The query to the UMLS server is then issued and all the possible semantic types of the term being examined are retrieved. For example, if the term is Aspirin, the UMLS server returns two possible semantic types Pharmacologic Substance and Organic Chemical which are stored in the list `semtypes`.

Lookup in Semantic Network

Once a term has been found in the metathesaurus and the semantic type retrieved, the procedure `IdentifyPM` goes through each element of the `semtypes` array and looks up the ancestor tree. The ancestor tree is then evaluated to determine whether the term is a medication or a problem.

3.4.5 Medications and Allergies

The current implementation of the program assumes that allergies are just a list of medications and therefore the fields with the name of the drug that denote either a medication ar an allergy are identical. The two cases are distinguished later on when all the fields have been tagged.

To identify a term as a medication, the ancestor tree of any medication must have Substance as a root. However each term can have multiple semantic types as illustrated above by the Aspirin example. Therefore we want to select the semantic type that is closest to the medication in order to distinguish a medication from an ordinary chemical that could be a laboratory measurement.

To better differentiate between medications and other EMRS elements, the biggest weight is assigned to the semantic type closest to a medication. This assignment is reflected in the procedure `med_iden`. It assigns weight 5 if the ancestor tree of the term being examined contains the semantic type Pharmacologic Substance, which can be only a medication, whereas it assigns only weight 2 if the semantic type is Chemical Viewed Structurally.

After all the terms have been looked up in UMLS, the highest weight for each term is chosen and all weights are added together. The sum is divided by the maximal possible weight for the terms, multiplied by 100, and reported as a confidence score for the given field.

Selecting the Best Candidate

When all fields of all tables have been examined and the confidence scores determined, the procedure `resolve_medication`, called from `SelectCandidates`, selects three fields with the highest confidence score to determine which field corresponds to medications and which one to allergies. The names of the three fields are stored in `med_scores` list together with the confidence socres.

The procedure `resolve_medication` works as follows: It takes each element of the `med_scores` list in turn and tries to identify additional attributes of the medication field as described in section 2.3.3. Specifically, `resolve_medication` looks for a date field that denotes the day on which the drug was administered, and dosage information. Each identified attribute field is given a confidence score.

When all the attributes for all fields in the `med_scores` list are found, the field with most attributes and the highest confidence scores is chosen as a medication EMRS element. The field with the second highest confidence score is chosen as an allergy EMRS element.

3.4.6 Problems

The identification of the problems is almost identical to the identification of medications. The main difference is that problems span several groups of semantic types. A problem can be, for example, an Anatomical Abnormality, whose ancestor tree has the root Entity, a Phenomenon or Process, or an Injury or Poisoning with a common root Event. Thus the `prob_iden` procedure which identifies the semantic type based on the ancestor tree assigns the following weights: a Disease or Syndrome semantic type, which is a descendant of Phenomenon or Process has weight 5 because it is the closest semantic type for a disease, whereas Anatomical Abnormality as well as Pathologic Function has only weight 4. The assignment of the confidence score is done in the same way as for the medications.

Selecting the Best Candidate

The lookup of additional attributes for problems (see section 2.3.3) is currently not implemented. Therefore the filed with the highest confidence score is selected as the problemelement of the EMRS.

3.5 Reporting the Results

The results of the implemented program are recorded into text files that are human-readable, but provide enough structure so that they can be easily parsed for further processing by other applications.

Both the **DRstructure** and **ElemIdent** create several files which store the results of the intermediate steps. For example the table relations identified by the `FindRelations` procedure are stored in the `table_relations.txt` file and the results of the `TagFields` procedure are stored in the `all_field_tags.txt` file. The results of some other procedures are also stored in text files as described in section 3.3. A complete listing of all intermediate results files appears in appendix B.

There are two files that summarize the results of the program. The file `tag_results.txt` shows for each identified EMRS element the name of the table and the fields in which the information is stored. In addition to the names, the file also lists the relevancy scores for the identified fields and some additional statistics such as how many fields in the data repository were identified as a candidate for the given EMRS element and the names of the fields that are the runner ups for each EMRS element if such a field exist.

The second file that summarizes the results of the program is the file `select_statements.txt`. This file gives for each identified EMRS element a SELECT statement that, given some information about a patient such as patient's name, will retrieve the appropriate fields from the database together with all identified attributes of that EMRS element.

The SELECT statements in the `select_statements.txt` file assume that the data repository is organized as a patient-oriented medical record. Thus the SELECT starts at the table with patient demographics and retrieves the appropriate EMRS element by finding the fewest number of relations necessary to reach the table with the EMRS element The shortest path is determined by breath-first algorithm and implemented in the `bfs_search` procedure which is called from the `create_select_stmt` procedure.

Chapter 4

Results

In order to establish the validity of the program described in chapter 3 and to evaluate its performance, the program was tested on a subset of data available from The Children's Hospital (TCH) in Boston, MA. The TCH data contains records of 300 patients whose personal information was changed prior to copying the data from the EMRS to maintain confidentiality and privacy of the patients.

Since only one EMRS was availble these results should not be considered as a thorough test of the program. The dataset from the TCH also served as a "training set" and therefore inevitably some of the program design was influenced by the structure of this training set data.

4.1 The TCH Data

Electronic medical records contain patient's medical history which is considered to be confidential and thus the data is not publicly available. Even if the data can be disclosed with patient's consent, they are considered proprietary and therefore the institutions are not likely to release them. Furthermore, the structure of the electronic records is also proprietary and is therefore hard to obtain. Because of the reasons given above the author could only obtain data from one existing EMRS - The Children's Hospital in Boston, MA, and test the program on those data.

4.1.1 Structure of the TCH EMRS

The Children's Hospital EMRS uses an ORACLE RDBMS (relational database management system) as its data repository. The complete database consists of approximately 1000 tables which contain anywhere from a few to a hundred of fields each.

The TCH EMRS stores detailed patient demographics information, financial records, and information about the personnel of the hospital. It has tables with problems, medications, patient's medical history, laboratory measurements, and visit notes. TCH uses a specialized vocabulary for coding the problems in patient's problem list. This vocabulary contains over 8000 terms and is stored in the EMRS. The terms in the vocabulary are more specific than the once used by standardized vocabularies such as ICD-9 or SNOMED.

4.1.2 Available Data

The data used for testing and evaluation of the program were stored in another ORACLE Server away from the actual TCH EMRS. The copied database consists of only 12 tables. The names of the tables and the number of fields and records in each database is given in table 4.1. A complete listing of all tables with their field names, types, and lengths is given in appendix B. This listing has been obtained by the program and is stored in the `all_tables.txt` file.

The records about patient's name, address, date of birth, and SSN were modified to maintain petient's confidentiality. Also the patient provider information has been altered in the PARSNL_PUBLIC table which stores the names of the personnel involved in providing care. Furthermore, the values of many fields in the database have been blanked out. For example most of the fields in PAT_FIN_ACCT are empty as well as the data about next of kin which is stored in the PAT_DEMOGRAPH table.

Despite the fact that the available database is relatively sparse, that is that many of the records are blank, the records storing the five EMRS elements used by the program are available and therefore the database is usable for testing and evaluation of the program.

Table name	*Fields*	*Records*
CHILD_DOCS	3	1619
CLINICAL_DATA	7	7013
DOC_ATTRIBUTES	3	3394
DOC_DESCRIPTION	11	1128
DOC_STORE	2	1128
PAT_DEMOGRAPH	60	300
PAT_FIN_ACCT	133	7617
PAT_TEST_HISTV	40	73421
PERSNL_PUBLIC	20	615
PHARMACY_TABLE	24	3668
PPR	10	1417
PROBLEMS	7	375

Table 4.1: The description of TCH EMRS tables used for the program evaluation.

4.2 Results for the TCH Data

The program correctly identified all five elements of the medical record and the results are summarized in `tag_results.txt` file. The complete results of the program running against the TCH data are listed in appendix B. which gives the content of all files that are generated when the program is run.

`all_field_tags.txt` - lists for each field of every table the confidence score for each element of the medical record. The score 0 means that the field has not been recognized.

`all_tables.txt` - written as a result of **DRstructure** module. It contains the field name, type, length, and order in the table for each field of the database. This file is read by **ElemIdent** module.

`name_keys.txt` - lists for each field that has been recognized as a key all tables in which this field occurs. This file reflects the content of the `db_keys` data structure.

`select_statements.txt` - contains the select statements that allow a retrieval of the appropriate EMRS element for a given `<clause>`. Usually, a `<clause>` is a condition written in SQL that specifies the name, address or other information about a patient. This file is the main result of the program.

`table_relations.txt` - lists all relations for each table. A relation is marked by the name of the other table, the name of the field in the first table constituting the relation, the name of the field in the second table, flags whether the relation has been identified based on name (NAME) or data content (DATA), and the score for the data relation.

`tag_results.txt` - lists the names of the fields and the tables that contain the elements of the electronic medical record as recognized by the program. The listing also contains some statistics such as confidence scores and the runner-ups for the respective elements.

4.2.1 Demographics

The patient demographics were located in the PAT_DEMOGRAPH table in which the first name, last name, address, city, and zip code were identified correctly. Even though the state field of the patient's address appears in the TCH database, it has not been identified by the program because the data in the STATE_CD field have been changed to abbreviations that do not correspond to the names of the 50 U.S. states.

The relevancy score for the PAT_DEMOGRAPH is 83 since the state demographics type was not recognized. The relevancy score is the percentage of identified demographics elements in the table. For example if for a given table only one field is recognized, then the score is 20. The relevancy score does not take into account zip codes, since there are potentially many numerical fields that can be identified as zip codes even though they are not.

The runner up for the patient demographics is the table PERSN_PUBLIC which stores information about the care providing personnel. The first name and last name were identified in this table. In total there were 33 fields identified as demographics types. The number of fields indentified as a given demographics type and the highest

Demographics Type	*Count*	*Score*
First Name	0	0
Last Name	0	0
Address	2	11
City	0	0
State	0	0
Zip Code	20	100

Table 4.2: The identified false positive demographics types.

confidence scores for all but the PAT_DEMOGRAPH and PERSN_PUBLIC tables are given in table 4.2. This table therefore lists all false positives.

4.2.2 Problems and Visit Notes

The problems were correctly identified as the field PROBLEM_NAME in the table PROBLEMS with the confidence score of 42. The only other field identified as problems is the ADMT_DIAG_CENTER field in the PAT_FIN_ACCT table. The confidence score for this field is 16.

The visit notes were identified as CONTENT field which is in the table DOC_STORE with score 1. This score is just a flag where 1 means that the field is free text. There were no other fields containing free text in the TCH database.

4.2.3 Medications and Allergies

There were 16 fields identified as potential candidates for medications and allergies. The field MEDICATION_NAME in the table PHARMACY_TABLE was identified as the medication element, while the field ALLERGY_TXT in the PAT_DEMOGRAPH table was identified as the allergy element. The confidence score for the MEDICATION_NAME is 60 and the confidence score for the ALLERGY_TXT field is 28.

Both fields were selected by the procedure `resolve_medication` as the only candidates for the medications and allergies, thus there were no false positives. Furthermore, some additional attributes for the medication element were identified. The date field type was correctly found to be the DATE_OF_SERVICE field, while the dosage attribute was incorrectly identified as the PATIENT_RECORD_NUMBER field.

4.2.4 SELECT Statements

The SELECT statements that retrieve the appropriate medical record element are put into the `select_statements.txt` file. The SELECT statements, as listed in the file, are not complete. The `<clause>` has to be replaced with a suitable SQL clause in order to create a valid SELECT statement that will retrieve the appropriate data. The `<clause>` must be a logical expression containing a field from the PAT_DEMOGRAPH table.

There is one noteworthy point about the SELECT statements. The statement for medications is not complete as no tables are given in the `from` clause. This is because the program did not find any fields that would relate this table to the PAT_DEMOGRAPH table which is assumed to be the starting point for any SELECTs.

4.3 Performance

The total running time for the **DRstructure** module was 28 seconds. This time includes connecting to the ORACLE database and running all parts of the program except for the procedure `compare_records_by_data` which finds relations based on similar data. During preliminary testing it took almost 3 days to find all relations based on the data. This poor performance is mainly due to two factors. The running time of the algorithm in **DRstructure** is $O(2^n)$ since it has to compare each field of every table with every other field. Furthermore, a single comparison took anywhere between 1 to 40 seconds as it performs a SELECT statement that joins two fields.

The running time for the **ElemIdent** was 23 minutes and 7 seconds and the upper bound estimate of this module is $O(n)$ where n is the number of fields of the database. The running time of 23:07 includes network connections to the UMLS server as well as connections to and retrieval from the ORACLE database.

Chapter 5

Discussion

The results presented in chapter 4 suggest that the implemented program performed well on the database from the Children's Hospital. It correctly identified all five elements of the medical record and discovered a flaw in the information model - there is no path from the patient PAT_DEMOGRAPH table to the PROBLEMS table. However before making any conclusions about the program performance, we should take a more detailed look to understand why the program performed the way it did.

5.1 Available Data

As mentioned in section 4.1, many records in the database are empty. It is therefore possible that if all records had meaningful data, the program would perform worse than it did. However, even though the used database was sparse, the program was able to clearly distinguish between the various elements of the medical record and identify them as such. Therefore we can infer that the program would perform reasonably well, even if the data were more complete.

The program might have difficulty choosing the right fields among the possible candidates for the given element of the medical record, but it would most likely identify correctly the possible candidates. This is demonstrated from the results by the fact that there is very little overlap, i.e. only five fields in the entire database were identified as two different medical record elements. However in each case the confidence scores were never higher than 16. For example in the case of the MEDICATION_NAME field in the PHARMACY_TABLE, the field was correctly identified as medications with the score of 60 and also as an address with the score 5.

Another case of identifying one field as two different elements is the field RELIG_CD in the PAT_DEMOGRAPH table, which was identified both as problems and medications. However each type had confidence score of only 5. The fourth case is the field PREV_SERV_CD in PAT_DEMOGRAPH table which was identified both as problems, with the score 2, and as medications, with the score of 5. Finally, the last case is the field ADMT_DIAG_DESCR, also in the PAT_DEMOGRAPH table, which was identified as zip code with the confidence score 5 and problems with confidence score 16. We can therefore say that the methods by which each element of the EMRS is identified provide enough distinction between the different elements.

5.2 Demographics

The program detected all demographics fields in the database which had nonempty records. However the STATE_CD and CITY_NAME fields in the PAT_DEMOGRAPH were tagged as address and last name respectively instead of as the state and the city demographics types. The reason for this inccorect identification of the demographics type is the fact that when the data were copied from the TCH EMRS, the city names were altered to the names of American presidents and the STATE_CD were replaced with meaningless data. Therefore the CITY_NAME field was identified as the last name demographics field. The program, however, chose the correct demographics type based on the content of the records.

Another interesting point about the demographics types is that large number of fields were identified as ZIP codes. The reason is that many unrelated numerical records are matched to the format of a zip code (five- or nine-digit number). This is not really significant, because the zip code demographics type is used only when the patient's address and city are selected from the possible candidates. Perhaps using more sophisticated techniques for determining zip codes such as the ones described in section 2.3.1 would give fewer false positives.

Even though the program selected the right fields in the correct table as patient's demographics, it would be interesting to evaluate if the method by which the fields are currently selected is valid for other EMRSs as well. The implementation of the program looks for the primary key of the table in which most demographics types were identified and then selects the identified demographics fields that are closest to the primary key field.

5.3 Problems

The problems in the PROBLEM_NAME field of the PROBLEMS table were identified with the confidence score of 42 which means that 21 out of 50 randomly selected records were positively identified as problems in the UMLS metathesaurus. Even though this score might seem to be low, it is high enough to distinguish the true positive from the false positives.

Many EMRSs use specialized vocabularies for recording the problems into the problem list. For example the TCH EMRS uses a specialized vocabulary therefore only 21 terms were identified as problems. As currently implemented in the program a problem is identified only if

1. the term is found in the UMLS metathesaurus and
2. the semantic type of the term falls into Disease or Syndrome, Anatomical Abnormality or Pathologic Function semantic category.

Because of these fairly strict rules, it is not surprising the score is only 42.

5.4 Medications

The medications element was identified with the confidence score of 60 and the allergy field received confidence score 28. The reason for the second score being so low is that the data in the available database were incomplete. There were only eight distinct records and out of those only three were meaningful names of drugs. The other five records contained either misspelled names of drugs, three-letter acronyms not recognized by UMLS, or the word "NONE", meaning that a particular particular does not have any allergies.

The program chose the field PATIENT_RECORD_NUMBER in PHARMACY_TABLE as the dosage field. The reason why the wrong field was selected is because the dosage information is already included in the MEDICATION_NAME field (for example `THIAMINE TAB 100MG UD`) and therefore there is no real dosage field in the PHARMACY_TABLE.

5.5 Performance

The total running time of the program is relatively short - 23 minutes and 35 seconds. The reason why the running time is so short compared to SCOUT [28] program, which ran for nine days on the same data set, is because SCOUT does not assume anything about the data in the database and learns concepts solely by analyzing the data. The implemented program, on the other hand, assumes that specific data types are present in the database and it only goes to the database to locate these data.

The reported running time of 23:35 does not include the running time of the procedure `compare_records_by_data`. As described in chapter 4, the running time of this method was alomost 3 days. The reason for this performance is inadequate design of the methodology. Currently, data from both fields are retireved and joined togoether which results in a SELECT operation that is very costly. To improve the running time we could instead sample certain number of records from one field and project those against the data in the second field rather than doing select and join on fields that can have each over 70000 records.

5.6 SELECT Statements Results

The main result of the program is the `select_statements.txt` file which contains SELECT statements that will locate the appropriate data in the database given some information about a patient for which the data should be located. This structure of the SELECT statement assumes that the electronic medical record is patient-oriented or at least that all information in the database is related to a patient and can be thus retrieved.

The SELECT file format for the results was chosen mainly to provide easy interface to a program that would use the information about the structure and content of the medical database. One of such programs is a program for unifying patient data from remote databases [30]. This program creates a complete medical record that is composed of data obtained from several EMRSs. In order to create a complete medical record, the program requires a site server that translates queries in HL7 protocol [9] into the queries understandable by the local data repository of the EMRS from which the data are obtained. Therefore the SELECT statements that are created by the implemented program are useful for creating the site servers.

Chapter 6

Related Work

6.1 Learning Concepts from Databases

The idea of learning concepts from a data repository appeared in a system called SCOUT [28] which used data from two medical databases. This system is concerned about learning concepts from a collection of data and does not require a specific query language. It only requires five basic operations on the data repository, but does not assume anything about the content.

The system described in this thesis, on the other hand, assumes that the data repository contains the specific information (see section 1.3) and looks for the occurrences of this information. Because the domain of the data is narrower than in the case of SCOUT, it uses more specialized methods to learn the structure such as semantic network of the UMLS and can learn more about the concepts in the data repository.

There exist literature describing various approaches to unsupervised learning of data from databases [8, 22]. The described techniques are applied to specific domains such as CAD databases or chemical databases. However there are no known methods that would extract information from medical databases.

The idea of obtaining a road map from databases appears, for example, in ORACLE's Database Designer product. The Reverse Engineer Wizard [19] connects to the database and "reverse engineers" a diagram showing the layout of the selected database. This product, however, works only on ORACLE RDBMS servers.

6.2 Structure of EMRSs

Even though comprehensive descriptions of the functionality of current EMRSs exist [13, 23, 16, 2, 11], the information about the structure or the information model of the data repository is not readily available and is only briefly mentioned in these publications. There are several reasons for the status quo: a formal information model was never developed for these systems, the existing model is outdated and therefore unusable, or the information is proprietary.

Levy and Beauchamp presented a poster [12] that describes the methods and working prototype of a system that maps the hierarchical MUMPS database to relational database tables. The approach they used, however, requires manual mapping between the two representations of the data repository.

Chapter 7

Conclusions

7.1 Lessons Learned

Working on the research, design, and implementation of this thesis has provided several valuable lessons. Some of the more important observations are summarized in the following paragraphs.

7.1.1 Usability of the Method

The methodologies presented in this thesis do not constitute any new or significant discovery in the area of computer science. The methods described in this thesis are specific to the EMRSs that contain certain information that is distinguishable from other data in the data repository. Furthermore, only a very small subset of data that is available from an EMRS data repository is recognized and accessed. However, based on the results obtained using the TCH database, the method works relatively well within its limited scope.

7.1.2 UMLS Knowledge Server

The UMLS Knowledge Server plays a central role in identifying some of the medical record elements. Even though the original purpose of the Knowledge Server was to provide dictionary services such as translations and semantic analysis, this thesis opens up a new area of knowledge discovery in which the UMLS Knowledge Server can be utilized. In author's opinion, the design and implementation of the UMLS extends well beyond its original goals and thus there will be more applications using the system outside of the area of medical vocabularies and translations in the future.

7.1.3 Programming in PERL

Implementation of the program proved that PERL is a suitable language for quick development of applications that use lot of text formatting and advanced UNIX programming. The power of the language, both in terms of the available data structures and the succinctness of the code, allowed to concentrate on the concepts of the program rather than spending time implementing higher-level data structures and elaborate text formatting routines.

The power of PERL, however, comes with a penalty. The circumvention of variable declarations before their use was a source of numerous frustrations which resulted simply from mistyping the variable names. Overall, however, the power and flexibility of PERL allowed rapid coding.

7.2 Future Work

The only results available at the time of writing this thesis were the ones obtained from the Children's hospital sample database. The next logical step is therefore to test the program against some other EMRSs. While it is impossible to test it against "live" EMRSs, it would be interesting to obtain some subset of data from other hospitals and run the program on it.

A natural extension is to use the results of the program for building a site server that takes queries in one format and translates them into the queries in the native format of the data repository as suggested in section 5.6. Using this approach an interesting application would be an automatic creation of a site server for a hospital. The integral part of this automatic generation of the site server would be the program presented in this thesis.

Finally, it would be worthwhile to implement the program again and "get it right this time". Specifically, rewriting the code in Objective C with EOF as stated in section 3.1 would result in faster execution of the program as well as usability of different data repositories. However this idea perhaps fits better the "wish" category.

7.3 Summary

The preceding chapters have demonstrated that the proposed approach to learning a small subset of information in a specialized database such as an EMRS data repository is possible. The presented results suggest that it is indeed possible to retrieve some information from a database without any prior knowledge of the structure and the format of the data.

The method relies on the fact that a small subset of specialized data is easily distinguishable from the other data in the database. However, readily available tools, namely the UMLS Knowledge Server, and a set of straightforward algorithms for distinguishing other elements of the medical record provided enough leverage to successfully identify the five basic elements of the medical records. The method may not be , however, applicable to domains other than databases with highly specific information such as EMRS.

Lastly, the conclusion of the work presented in this thesis is only valid for the data used in the experiment. Further testing against other EMRS data repositories is necessary before this conclusion can be generalized to other Electronic Medical Record Systems whose format and structure is different from the used TCH database.

Appendix A

Source Code Files

A.1 config.pl

```
#!/usr/local/bin/perl -w
# -*- Perl -*-
#
# $Id: $
#

use Oraperl;

#---------------------------------------------------------------------
# General settings
#---------------------------------------------------------------------

#logs

# switch for generating output
$GENERATE_LOGS = 1;

# switch for writing logs to a file rather than to STDOUT
$GENERATE_TO_FILES = 1;

#---------------------------------------------------------------------
# Log files/directories
#---------------------------------------------------------------------

#subdirectory for logs
$LOGS_DIR = "logs/";

$activity_log = $LOGS_DIR."activity.log";

$table_relations_file = $LOGS_DIR."table_relations.txt";
$table_names_file = $LOGS_DIR."table_names.txt";
$all_tables_file = $LOGS_DIR."all_tables.txt";
$tags_file = $LOGS_DIR."all_field_tags.txt";
$tag_result_file = $LOGS_DIR."tag_results.txt";
$db_keys_file = $LOGS_DIR."name_keys.txt";

$select_file = $LOGS_DIR."select_statements.txt";

#directory with templeate (master) files such as address templates, names etc.
$MASTERS_DIR = "masters/";
```

```
#-----------------------------------------------------------------------
# For column identifications
#-----------------------------------------------------------------------

#number of unique records to select from the database
$NUMBER_OF_RECORDS = 50;

#list of tags for database columns, given
@DEMOGRAPH_TYPES = ("ZIP","FNAME","LNAME","ADDR","CITY","STATE","",);

#how many identified fields for medications to report minus one
$MEDS = 1;

# Hack, to ignore "bad free" messages.  See DBD README.
$SIG{__WARN__} = sub { warn $_[0] unless $_[0] =~ /^Duplicate free/ };


#-----------------------------------------------------------------------
# UMLS Semantic Network indentifiers
#-----------------------------------------------------------------------

$DISEASE = "Disease or Syndrome";
$PATHOLOGIC = "Pathologic Function";
$ABNORMALITY = "Anatomical Abnormality";

$PHARMACOLOGIC ="Pharmacologic Substance";
$CHEMSTRUC = "Chemical Viewed Structurally";

$MAXPROBL = 5;
$MAXMEDIC = 5;

$MAXPROX = 9999;

1;
```

A.2 db_connect.pl

```
#!/usr/local/bin/perl
# -*- Perl -*-

use Oraperl;

$connect_string = "*********";
$OWNER="******";

sub DBLogin {
#
# returns $lda - file handle for DB session

    local($lda) = &ora_login("",@_, "")
    || die $ora_errstr;
    return ($lda);
}

sub GetTableNames {
#
# returns @table_names
    $sqlstr="select table_name from all_tables where owner='$OWNER'";
    $doccsr = &ora_open($lda, $sqlstr)
        || die $ora_errstr;
    while (($tnam) = &ora_fetch($doccsr)) {
        @tnames = (@tnames,$tnam);
    }
    &ora_close($doccsr);
    return @tnames;
}

sub count_records {
    local($sqlstr) = @_;
    local($n);
    local($doccsr) = &ora_open($lda, $sqlstr)
        || die $ora_errstr;
    while (($foo) = &ora_fetch($doccsr)) { $n=$foo; }
    &ora_close($doccsr);
    return $n;
}

sub GetDescriptions {

    foreach $tname (@_) {
        $position = 1;
        $sqlstr="select column_name,data_type,data_length,nullable from ".
            "all_tab_columns where owner='$OWNER' and table_name='$tname'";

        $doccsr = &ora_open($lda, $sqlstr) || die $ora_errstr;

        local($first) = 1;
        while (($cname,$dtype,$dlength,$nul) = &ora_fetch($doccsr)) {

            # if the first field of the table, check the # of records
            if ($first) {
                local($count_sqlstr) =
                    "select count($tname.$cname) from $tname";
                $table_lengths{$tname} = &count_records($count_sqlstr);
                $first = 0;
            }
            $all_tables{$tname}{$cname}=[($dtype,$dlength,$nul,$position++)];
        }
    }
}

sub DBLogout
```

```
{
    &ora_logoff($lda);
}

1;
```

A.3 drstruc.pl

```
#!/usr/local/bin/perl
# -*- Perl -*-
#
# $Id: $
#

use Oraperl;

require "config.pl";
require "db_connect.pl";
require "file_io.pl";


# threshold for deciding the joined set should be taken as a key,1..100)
$KEY_THRESHOLD = 40;

#----------------------------------------------------------------------


# create the log file
log_activity("Selecting elemets started");


# login to the database
$lda = &DBLogin($connect_string);

# get all table names
$start = (times)[0];
log_activity("Getting table names started");
@table_names = &GetTableNames();
$end = (times)[0];
&log_entry("Table names",$end-$start) if ($GENERATE_TO_FILES);
&print_table_names() if ($GENERATE_LOGS);
log_activity("Getting table names finished");

undef(%table_lengths);
undef(%all_tables);

# get descriptions of all tables - names of fields and field types+lengths
$start = (times)[0];
&GetDescriptions(@table_names);
&print_tables() if ($GENERATE_LOGS);
$end = (times)[0];

&log_entry("Table descriptions",$end-$start) if ($GENERATE_TO_FILES);

undef(%relations);   # declare associative array for relations
undef (%db_keys);    # declare associative array for db keys

log_activity("Finding relations started");
$start = (times)[0];
&FindRelations();
$end = (times)[0];

# close connection to the database
&DBLogout;

&print_table_relations() if ($GENERATE_LOGS);
&print_db_keys() if ($GENERATE_LOGS);
&log_entry("All relations",$end-$start) if ($GENERATE_TO_FILES);
log_activity("Finding relations finished");

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
```

```
sub FindRelations {

# used for speeding search to avoid 2N comparisons
    @rest_tables = @table_names;

    open(FH_MATCH,">".$LOGS_DIR."all_matches.txt");
    print FH_MATCH "#All SELECTs and JOINs performed to determine relations\n\n";
    foreach $mytab (@table_names) {
        shift(@rest_tables);
        print FH_MATCH "Table: $mytab ($table_lengths{$mytab} records)\n";
        foreach $myfld (keys(%{$all_tables{$mytab}})) {
            $myftype = $all_tables{$mytab}{$myfld}[0];
            foreach $othertab (@rest_tables) {
                foreach $otherfld (keys(%{$all_tables{$othertab}})) {
                    $otherftype = $all_tables{$othertab}{$otherfld}[0];
                    # compare sets of data between two fields of the same type
                    if ($myftype eq $otherftype) {
                        $by_name = 0;
                        # first locate keys based on the same field names only
                        if ($myfld eq $otherfld) { # a match of field names
                            $by_name = 1;
                            $db_keys{$myfld}{$mytab} = 1;
                            $db_keys{$myfld}{$othertab} = 1;
                        }
                        # now do comparison based on data itself
                        $by_data = 0; $score = 0;

#                        &compare_records_by_data();

                        if ($by_name || $by_data) {
                            $relations{$mytab}{$othertab} =
                                [($myfld,$otherfld,$by_name,$by_data,$score)];
                            $relations{$othertab}{$mytab} =
                                [($otherfld,$myfld,$by_name,$by_data,$score)];
                        }
                    }
                }
            }
        }
    }
    close FH_MATCH;
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub compare_records_by_data {
    # get count for $mytab
    # $sqlstr="select count($mytab.$myfld) from $mytab";
    # &count_records($sqlstr);
    $mytab_count = $table_lengths{$mytab};

    # get count for $othertab
    # $sqlstr="select $othertab.$otherfld from $othertab";
    # $othertab_count = &count_records($sqlstr);

    #perform SQL select and join
    $sqlstr="select count(distinct $mytab.$myfld) from $mytab,$othertab ".
        "where $mytab.$myfld=$othertab.$otherfld";
    if ($n=count_records($sqlstr)) {
        $perc = $n/$mytab_count;
        $perc =~s/(\d.\d\d).*/\1/;
        $perc =$perc*100;
        # for now, it is possible to get score higher than 100 ???
        if ($perc > 99) { $perc = 100 };

        print FH_MATCH "\t$sqlstr\n";
                "\t\tMatches(n):$n\tRecords(mytab):$mytab_count\t$perc%\n\n";
        #if subset found they can be keys
        if ($perc >= $KEY_THRESHOLD) {
            # we have a candidate, set flag to 1
            $by_data = 1;
            $score = $perc;
        }
    }
}
```

A.4 elemident.pl

```
#!/usr/local/bin/perl
# -*- Perl -*-
#
# $Id: $
#

use Oraperl;

require "config.pl";
require "db_connect.pl";
require "file_io.pl";
require "umls_conn.pl";
# require "utils.pl";

undef(%all_tables);
undef(%relations);

# print '/bin/date'." -start\n";

# login to the database
$lda = &DBLogin($connect_string);

# print '/bin/date'." -end\n";

&read_master_files();
&read_tables();
&read_db_keys();

&read_table_relations();

# for debugging only
# &read_tags();

log_activity("Tagging fields started");
&TagFields();
&print_field_tags() if ($GENERATE_LOGS);
log_activity("Tagging fields finished");

undef(@m_addrs,%m_fnames,%m_lnames,@m_cities);

log_activity("Selecting elements started");
&SelectCandidates();
&print_tag_results() if ($GENERATE_LOGS);
log_activity("Selecting elements finished");

&DBLogout();

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub SelectCandidates {
    local($temp);
    undef(%dem_scores); undef(%dem_relev);
    local(@ftypes) = @DEMOGRAPH_TYPES;
    pop(@ftypes); #get rid of the "" field type at the end of the array


    foreach $temp (@ftypes) { $dem_scores{$temp} = [("","",0)]; }
    foreach $temp (0..$MEDS) { $med_scores[$temp] = [("","",0)]; }
    ($n_prob,$n_med,$n_dem,$n_note,$total_fields) = (0,0,0,0,0);
    @problem_field = ("","",0); @note_field = ("","",0);

    foreach $table (keys(%all_tables)) {
        $dem_relev{$table}[0] = 0; $dem_relev{$table}[1] = 0;
        foreach $field (keys(%{$all_tables{$table}})) {
            $total_fields++;
            #account demographics
            &dem_accn();
            #account medications
            &med_accn();
            #account problems field
            if ($all_tables{$table}{$field}[6] > 0) {
                $n_prob++;
                if ($all_tables{$table}{$field}[6] > $problem_field[2]) {
                    @problem_field =
                        ($table,$field,$all_tables{$table}{$field}[6]);
                }
            }
            #account for notes
            if ($all_tables{$table}{$field}[8] > 0) {
                $n_note++;
                if ($all_tables{$table}{$field}[8] > $note_field[2]) {
                    @note_field =
                        ($table,$field,$all_tables{$table}{$field}[8]);
                }
            }
        }
    }


    # choose address
    ($dem_table,$my_fld) = &resolve_primkey();
    &resolve_addr($dem_table,$my_fld);
    #distinguish between medications and alleriges
    &resolve_medications();
    #find attribs for problems
#    &find_attr_problems();
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub resolve_medications {
    local($tab,$fld,$prox,$fld_idx,$tab_idx,$el);
    local(@dosage) = (0,"",$MAXPROX);
    local(@date_score) = ($MAXPROX,"");

    # the keys are table,field,dosage,dos_score,date,dat_score
    undef(%med_resolved);

    foreach $el (0..$MEDS) {
        $tab = $med_scores[$el][0];
        if ($tab ne "") {
            $tab_idx = $$all_tables{$tab}{$med_scores[$el][1]}[3];
            foreach $fld (keys(%{$all_tables{$tab}})) {
                $fld_idx = $all_tables{$tab}{$fld}[3];
                $score = 0;
                $prox = abs($tab_idx-$fld_idx);
                #look for dosage
                if (($all_tables{$med_scores[$el][0]}{$fld}[0] eq "NUMBER") ||
                    ($all_tables{$med_scores[$el][0]}{$fld}[0] eq "VARCHAR2")){
                    $sqlstr = "select distinct $tab.$fld from $tab";
                    $unique = &GetUniqueRecords($NUMBER_OF_RECORDS,$sqlstr);

                    if ($unique>0) {
                        foreach $iter (keys(%unique_recs)) {
                            $iter =~ s/[a-z]/[A-Z]/;
                            if ( $iter =~ /^(\d{2,4})(\s*(MGR|TABL))?/) {
                                $score ++;
                            };
                        }
                        $score = &normalize_score($score/$unique);
                        if (($score>$dosage[0]) ||
                ((\$score>0)&&(\$prox<\$dosage[2]))) {
                @dosage = ($score,$fld,$prox);
            }
        }
    }
    if (($all_tables{$tab}{$fld}[0] eq "DATE") &&
        ($prox<$date_score[0])) {
        @date_score = ($prox,$fld);
    }
}
local($med_fld)=$med_scores[$el][1];
#evaluate and choose only one
# the keys are table,field,dosage,dos_score,date,dat_score
if (($date_score[0]<$MAXPROX) || ($dosage[0]>0)) {
    $change_vals = 0;
    if (($date_score[0]<$MAXPROX) && ($dosage[0]>0)) {
        if (defined($med_resolved{"date"}) &&
            defined($med_resolved{"dosage"})) {
            if (($date_score[0]<$med_resolved{"dat_score"}) &&
                ($dosage[0]>$med_resolved{"dos_score"})) {
                $change_vals = 1;
            }
        }
        else { #both identified, but only one defined
            $change_vals = 1;
        }
        if ($change_vals) {
            $med_resolved{"table"} = $tab;
            $med_resolved{"field"} = $med_fld;
            $med_resolved{"date"} = $date_score[1];
            $med_resolved{"dat_score"} = $date_score[0];
            $med_resolved{"dosage"} = $dosage[1];
            $med_resolved{"dos_score"} = $dosage[0];
        }
    } # endif of both identified
    $change_date =0;
    if ($date_score[0]<$MAXPROX) {
        if (defined($med_resolved{"dat_score"})) {
            if ($date_score[0]>$med_resolved{"dat_score"}) {
                $change_date = 1;
            }
        }
        else {
            $change_date = 1;
        }
    }
    if ($change_date) {
        $med_resolved{"date"} = $date_score[1];
        $med_resolved{"dat_score"} = $date_score[0];
        $med_resolved{"table"} = $tab;
        $med_resolved{"field"} = $med_fld;
        undef($med_resolved{"dosage"});
        undef($med_resolved{"dos_score"});
    }
    $change_dosage = 0;
    if ($dosage[0]>0) {
        if (defined($med_resolved{"dosage"})) {
            if ($dosage[0]>$med_resolved{"dos_score"}) {
                $change_dosage = 1;
            }
        }
        else {
            $change_dosage = 1;
        }
    }
    if ($change_dos) {
        undef($med_resolved{"date"});
        undef($med_resolved{"dat_score"});
```

```
                    $med_resolved{"dosage"} = $dosage[1];
                    $med_resolved{"dos_score"} = $dosage[0];
                    $med_resolved{"table"} = $tab;
                    $med_resolved{"field"} = $med_fld;
                }
            }
        }
    }
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub resolve_addr {

    local($the_tab,$fld) = @_;
    local(@proxy) = @DEMOGRAPH_TYPES;
    pop(@proxy); # get rid of the blank at the end
#    shift(@proxy); #ignore ZIP codes
    undef(%dem_proximity);
    foreach $el (@proxy) {
        $dem_proximity{$el} =  [ ($MAXPROX,"") ];
    }
    local($index) =  $all_tables{$the_tab}{$fld}[3];
    local($prox_idx);
    foreach $field (keys(%{$all_tables{$the_tab}})) {
        $prox_idx = $all_tables{$the_tab}{$field}[4];
        if (defined($dem_proximity{$prox_idx})) {
            $distance = abs($index-$all_tables{$the_tab}{$field}[3]);
            if ($distance<$dem_proximity{$prox_idx}[0]) {
                $dem_proximity{$prox_idx}[0] = $distance;
                $dem_proximity{$prox_idx}[1] = $field;
            }
        }
    }
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub resolve_primkey {
    local ($fld,$temp,$max) = (0,0,0);
    local ($the_tab) = "";

    foreach $table (keys(%dem_relev)) {
        # determine the "primary key" in that table
        if ($dem_relev{$table}[1]>$max) {
            $the_tab = $table;
            $max = $dem_relev{$table}[1];
        }
    }

    $max = 0;
    foreach $field (keys(%{$all_tables{$table}})) {
        if ($db_keys{$field}{$table}) {
            local(@temp_tables) = (keys(%{$db_keys{$field}}));
            # decide on the most number of references
            if ($#temp_tables > $max) {
                $max = $#temp_tables;
                $fld = $field;
#                $the_tab = $table;
            }
        }
    }
    return($the_tab,$fld);
}

#----------------------------------------------------------------------
```

```
#
#-----------------------------------------------------------------------
sub med_accn {
    local($j,$modified);
    $modify = 1;
    if ($all_tables{$table}{$field}[7] > 0) {
        $n_med++;
        foreach $j (0..$MEDS) {
            if (($all_tables{$table}{$field}[7] > $med_scores[$j][2]) &&
                ($modify)) {
                splice(@med_scores,$j,0,
                       [ ($table,$field,$all_tables{$table}{$field}[7])]);
                $modify = 0;

#            $med_scores[0] =
#                [ ($table,$field,$all_tables{$table}{$field}[7])] ;
#            @med_scores = sort( {$med_scores[$a][2] <=>
#                                  $med_scores[$b][2]} @med_scores);
            }
        }
    }
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub dem_accn {
    if ($all_tables{$table}{$field}[5] > 0) {
        $n_dem++;
        $dem_relev{$table}[0]++;
        if ($all_tables{$table}{$field}[4] ne "ZIP") {
            $dem_relev{$table}[1]++;
        }

#        $dem_relev{$table}[1] = abs($dem_relev{$table}[1] -
#                                    $all_tables{$table}{$field}[3]);
#        if ($all_tables{$table}{$field}[5] >
#            $dem_scores{$all_tables{$table}{$field}[4]}[2]) {
#
#            $dem_scores{$all_tables{$table}{$field}[4]} =
#                [ $table,$field,$all_tables{$table}{$field}[5] ];
#        }

    }
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub TagFields {
    foreach $table (keys(%all_tables)) {
#        print "$table:\n";
        foreach $field (keys(%{$all_tables{$table}})) {
#        print "\t$field\t".`/bin/date`."\n";
            ($all_tables{$table}{$field}[4],$all_tables{$table}{$field}[5],
             $all_tables{$table}{$field}[6],$all_tables{$table}{$field}[7],
             $all_tables{$table}{$field}[8]) = ("",0,0,0,0);
            if (($all_tables{$table}{$field}[0] ne "LONG") &&
                ($all_tables{$table}{$field}[0] ne "DATE"))        {
                $sqlstr = "select distinct $table.$field from $table";
                $unique = &GetUniqueRecords($NUMBER_OF_RECORDS,$sqlstr);
                if ($unique>0) {
                    ($dem_type,$dem_score) = &IdentifyDemograph($unique);
                    $all_tables{$table}{$field}[4] = $dem_type;
                    $all_tables{$table}{$field}[5] = $dem_score;

                    ($all_tables{$table}{$field}[6],
                     $all_tables{$table}{$field}[7]) = &IdentifyPM();
                }
            }
            elsif ($all_tables{$table}{$field}[0] eq "LONG") {
                # this is a LONG field - check as a note
                $all_tables{$table}{$field}[8] = 1;
            }

#            print "\t=> $field [".join(",",@{$all_tables{$table}{$field}}).
#                "]\n";
        }
    }
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub IdentifyPM {
    local($dosage_shorts) = "UD|SOL|MG|INJ|LIQ|TAB|ML|CAP|SUSP";
    $ret_sprobl = $ret_smedic = 0;
    if         ($all_tables{$table}{$field}[0] eq "VARCHAR2") {
        &umls_connect();
        local($s_probl,$s_medic) = (0,0); local($temp);
        foreach $term (keys(%unique_recs)) {
            #get rid of numbers, parentheses, acronyms for units etc.
            $term =~s/\(.*\)//g;
            $term =~s/\b[\d\.]*\b//g;
            $term =~ s/\b([\w\.\/\d])*($dosage_shorts)[\w\.\/\d]*\b//g;
            &umls_request("-meta|-cst|$term");
            &get_semtype();
            $s_probl = $s_medic = $s_aller = 0;
            foreach $st (@semtypes) {
                &umls_request("-net|-anc|$st");
                &get_anctree();
                #ensure that we get the highest possible score
                if (($temp=&prob_iden()) > $s_probl) { $s_probl=$temp; }
                $temp = 0;
                if (($temp=&med_iden()) > $s_medic) { $s_medic=$temp; }
            }
            $ret_sprobl += $s_probl;
            $ret_smedic += $s_medic;
        }
        &umls_disconnect();
        $ret_sprobl = $ret_sprobl / ($MAXPROBL*$unique);
        $ret_sprobl =~s/(\d.\d\d).*/\1/; $ret_sprobl *= 100;
        $ret_smedic = $ret_smedic / ($MAXMEDIC*$unique);
        $ret_smedic =~s/(\d.\d\d).*/\1/; $ret_smedic *= 100;
    }
    return($ret_sprobl,$ret_smedic);
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub med_iden {
    if ($ancestors[5]=~ /$PHARMACOLOGIC/) {
        return(5);
    }
    elsif ($ancestors[4] =~ /$CHEMSTRUC/) {
        return(2);
    }
    else {
        return(0);
    }
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
```

```
sub prob_iden {
    if ($ancestors[5] =~ /$DISEASE/) {
        return(5);
    }
    elsif ($ancestors[4] =~ /$PATHOLOGIC/) {
        return(4);
    }
    elsif ($ancestors[3] =~ /$ABNORMALITY/) {
        return(4);
    }
    else {
        return(0);
    }
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub get_anctree {
    local($cont) = 1; local($line);
    undef(@ancestors);
    while($cont && ($line = <SocHandle>)) {
        chop($line);
        if ($line =~ /%%/) {
            $cont = 0;
        }
        else {
            $line =~ s/^\s+//;
            @ancestors = (@ancestors,$line);
        }
    }
#    print "    ".join(" -> ",@ancestors)."\n" if ($field eq "ALLERGY_TXT");
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub get_semtype {
    undef(@semtypes);
    local($cont) = 1;
    local($line);
    while($cont && ($line = <SocHandle>)) {
        if ($line =~ /Semantic Type:\s+(.*)/) {
#            print "==>$term\t\t$1\n";
            @semtypes = (@semtypes,$1);
        }
        $cont = 0 if ($line =~ /%%/);
    }
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub IdentifyDemograph {
# assumes the fields are stripped of first k common characters and are
# alligned left

# possible values ZIP,FNAME,LNAME,ADDR,CITY,ST

    local($n) = @_;
    local($iter,$max,$index_max,$i);
    local(@scores);
    $i=0;
    local(@ftypes) = @DEMOGRAPH_TYPES;
    foreach $iter (@ftypes) {
        $scores[$i++] = 0;
    }
    if (($all_tables{$table}{$field}[0] eq "NUMBER") ||
        ($all_tables{$table}{$field}[0] eq "VARCHAR2")) {
        foreach $iter (keys(%unique_recs)) {
            # zipcode
            $scores[0]++ if ( $iter =~ /^((\d{5,9})|(\d{5,5}-\d{4,4}))\b/);

            if ($all_tables{$table}{$field}[0] eq "VARCHAR2") {

                # first name
                $scores[1]++ if (defined($m_fnames{$iter}));

                # last name
                $scores[2]++ if (defined($m_lnames{$iter}));

                # translate to upper case
                $iter =~ tr/a-z/A-Z/;
                # address

                if ($iter=~/\b($match_addr)(\b|\.)/) {
                    $scores[3]++;
                }
            }
        }
    }
    $n++; # there are n+1 unique records selected
    foreach $i (0..$#scores) {
        $scores[$i] = $scores[$i]/$n;
        $scores[$i] =~s/(\d.\d\d).*/\1/;
        $scores[$i] = $scores[$i]*100;
    }
    $index_max = $i = $max = 0;
    foreach $iter (@scores) {
        if ($iter>$max) {
            $max = $iter;
            $index_max = $i;
        }
        $i++;
    }
    $iter = $ftypes[$index_max];
    $iter = "" if ($max==0);
    return ($iter,$max);
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub GetUniqueRecords {
    local($how_many,$sqlstr) = @_;
    local($n) = 0;
    local($doccsr) = &ora_open($lda, $sqlstr)
        || die $ora_errstr;
    local($cont) = 1;
    undef(%unique_recs);
    while (($foo) = &ora_fetch($doccsr)) {
        if ($n <= $how_many) {
            if (!defined($unique_recs{$foo})) {
                $unique_recs{$foo} = 1;
                $n++;
            }
        }
    }
    &ora_close($doccsr);
    # decrement n by one because we have a hash key ''
    return(--$n);
}
```

A.5 file_io.pl

```
#!/usr/local/bin/perl
# -*- Perl -*-
#
# $Id: $
#

require "config.pl";
require "utils.pl";

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub log_entry {
    local($str,$time) = @_;
    open(FH_LOG,">>$activity_log");
    printf FH_LOG "$str obtained in %2f CPU seconds\n",$time;
    close FH_LOG;
}

sub log_activity {
    local($text) = @_;
    if ($GENERATE_TO_FILES) {
        open(FH_LOG,">>$activity_log") ;
        local($day,$month,$date,$time,$zone,$year) = split(/ +/,`/bin/date`);
        print FH_LOG $text." at $time on $month $date, $year";
        close FH_LOG;
    }
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub print_table_relations {
    if ($GENERATE_TO_FILES==0){
        open(FL,">&STDOUT");
    }
    else {
        open(FL,">$table_relations_file");
    }


    foreach $thetab (keys(%relations)) {
        print FL "$thetab:\n";
        foreach $otab (keys(%{$relations{$thetab}})) {

            ($fld1,$fld2,$by_name,$by_data,$score) =
                @{$relations{$thetab}{$otab}};
            print FL "\t$otab [$fld1->$fld2]  ".
                "(NAME[$by_name],DATA[$by_data],$score)\n";

        }
        print FL "\n";
    }
    close FL;
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub print_table_names {
    if ($GENERATE_TO_FILES==0){
        open(FL,">&STDOUT");
    }
    else {
        open(FL,">$table_names_file");
    }
    foreach $tname (@table_names) {
        print FL "$tname\n";
    }

    close FL;
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub print_tables {
    if ($GENERATE_TO_FILES==0){
        open(FL,">&STDOUT");
    }
    else {
        open(FL,">$all_tables_file");
    }
    foreach $key (keys(%all_tables)) {
        print FL "$key: $table_lengths{$key} records\n";
        foreach $field (sort({ @{$all_tables{$key}{$a}}[3] <=>
                                   @{$all_tables{$key}{$b}}[3] }
                             (keys(%{$all_tables{$key}})))) {
            print FL "\t$field [".
                join(",",@{$all_tables{$key}{$field}})."]\n";
        }
        print FL "\n";
    }
    close FL;
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub print_field_tags {
    if ($GENERATE_TO_FILES==0){
        open(FL,">&STDOUT");
    }
    else {
        open(FL,">$tags_file");
    }

    format FL =
 @<<<<<<<<<<<<<<<<<<<<<<<<< @<<<<<<    @>>>      @>>>       @>>>    @>>>
$field,$all_tables{$key}{$field}[4],$all_tables{$key}{$field}[5],$all_tables{$key}{$field}[6],
$all_tables{$key}{$field}[7],$all_tables{$key}{$field}[8].

    foreach $key (keys(%all_tables)) {
        $top_line =  <<EOF
          Field             DemType  DemScore  Problems  Medications Notes
------------------------------------------------------------------------------
$key: $table_lengths{$key} records
EOF
    ;
        print FL $top_line;
        foreach $field (sort({ @{$all_tables{$key}{$a}}[3] <=>
                                   @{$all_tables{$key}{$b}}[3] }
                             (keys(%{$all_tables{$key}})))) {
            write(FL);
        }
        $- = 0;
        print FL "\n";
    }
    close FL;
}

#----------------------------------------------------------------------
#
```

```
#-----------------------------------------------------------------------
sub print_db_keys {
    if ($GENERATE_TO_FILES==0){
        open(FL,">&STDOUT");
    }
    else {
        open(FL,">$db_keys_file");
    }
    local($el,$tab);
    foreach $el (keys(%db_keys)) {
        print FL "$el:\n   ".join(",",(keys(%{$db_keys{$el}}))). "\n";
    }
    close FL;
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub print_tag_results {
    if ($GENERATE_TO_FILES==0){
        open(FL,">&STDOUT");
        open(SELFH,">&STDOUT");
    }
    else {
        open(FL,">$tag_result_file");
        open(SELFH,">$select_file");
    }
    local($el,$fl);
    print FL "DEMOGRAPHICS: $n_dem fields out of $total_fields identified.\n";
    foreach $el (keys(%dem_scores)) {
        print FL "\t$el in $dem_scores{$el}[0].$dem_scores{$el}[1] ".
            "with the score of $dem_scores{$el}[2]\n"
                if ($dem_scores{$el}[2]>0);
    }
    foreach $el (keys(%dem_relev)) {
        if ($dem_relev{$el}[1] > 0) {
            local($skore) = $dem_relev{$el}[1] / $dem_relev{$el}[0];
            $skore=~s/(\d.\d\d).*/\1/;
            $skore *= 100;

            print FL "   Table $el has $dem_relev{$el}[0] fields with the ".
                "relevancy factor of $skore.\n";  # $dem_relev{$el}[1].\n" ;

            foreach $fl (keys(%{$all_tables{$el}})) {
                if ($all_tables{$el}{$fl}[5]>0) {
                    print FL "      $fl"."[".$all_tables{$el}{$fl}[4]."]\n";
                }
            }
        }
    }
    print SELFH "DEMOGRAPHICS: ";
    local(@flds);
    print FL "   After proximity to 'primary key' ($dem_table):\n";
    foreach $el (keys(%dem_proximity)) {
        if ($dem_proximity{$el}[0] < $MAXPROX) {
            print FL "\t$el: $dem_proximity{$el}[0],$dem_proximity{$el}[1]\n";
            push(@flds,$dem_table.".".$dem_proximity{$el}[1]);
                print SELFH "$el ";
        }
    }
    print SELFH "\n";
    print SELFH "\t".&create_select_stmt($dem_table,$dem_table,@flds).
        " <clause>\n\n";

    print SELFH "MEDICATIONS:";
    print FL "MEDICATIONS & ALLERGIES: $n_med fields out of $total_fields identified.\n";
    foreach $el (0..$MEDS) {
        print FL "\t$el: located in $med_scores[$el][0].$med_scores[$el][1] ".
```

```
            "with the score of $med_scores[$el][2].\n"
                if ($med_scores[$el][2]>0);
    }
    local(@temp_keys) = keys(%med_resolved);
    if ($#temp_keys > 0) {
        print FL "   Medications identified as ".
            $med_resolved{"table"}.".".$med_resolved{"field"}."\n";
        foreach $el (keys(%med_resolved)) {
            print FL"\t$el: $med_resolved{$el}.\n" if (($el ne "table") &&
                                                       ($el ne "field"));
        }
    }
    print SELFH "\n";
    print SELFH "\t".&create_select_stmt($dem_table,$med_resolved{"table"},
                                       ($med_resolved{"table"}.".".
                                        $med_resolved{"field"})).
        " AND <clause>\n\n";

    print SELFH "PROBLEMS:";
    print FL "PROBLEMS: identified as $problem_field[0].$problem_field[1] ".
        "with the score of $problem_field[2]\n" if ($problem_field[2]>0);

    print SELFH "\n";
    print SELFH "\t".&create_select_stmt($dem_table,$problem_field[0],
                                       ($problem_field[0].".".
                                        $problem_field[1])).
        " AND <clause>\n\n";

    print SELFH "NOTES:";
    print FL "NOTES: identified as $note_field[0].$note_field[1] ".
        "with the score of $note_field[2]\n" if ($note_field[2]>0);
    print FL "\n";

    print SELFH "\n";
    print SELFH "\t".&create_select_stmt($dem_table,$note_field[0],
                                       ($note_field[0].".".$note_field[1])).
        " AND <clause>\n\n";


    close FL;
    close SELFH;
}


#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub read_tables {
    open(FH,"$all_tables_file");
    while (<FH>) {
        if (/^(\w*): (\d*) records/) {
            $tname = $1;
            @table_names = (@table_names,$1);
            $table_lengths{$1} = $2;
        }
        if (/^\t(\w*) \[(\w*),(\d*),(\w),(\d*)\]/) {
            $all_tables{$tname}{$1}=[($2,$3,$4,$5)]
        }
    }
    close FH;

}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub read_db_keys {
    open(FH,"$db_keys_file");
    local(@tables);
    while (<FH>) {
        if (/^(\w*):/) {
            $kname = $1;
        }
        if (/^\s+(.*)/) {
            @tables = split(/,/,$1);
            foreach $el (@tables) {
                $db_keys{$kname}{$el} = 1;
            }
        }
    }
    close FH;

}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub read_table_relations {

    open(FH,"$table_relations_file") || die "Could not open $table_relations_file\n";
    local(@thetab);
    while (<FH>) {
        if (/^(\w*):/) {

            $thetab = $1;
        }
        if (/^\t(\w*)\s+\[(\w*)->(\w*)\]\s+\(NAME\[(\d+)\],DATA\[(\d+)\],(\d+)\)/) {
            $relations{$thetab}{$1} = [ ($2,$3,$4,$5,$6) ];
        }

    }
    close FH;
}
#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub read_tags{
    open(FL,"$tags_file") || die "Could not open $tgs_file\n";
    local(@thetab);
    $asgn = 0;
    while (<FL>) {
        if (/^(\w*):/) {  $thetab = $1;        }
        if (/(\w*)\s+([A-Z]*)\s*(\d+)\s+(\d+)\s+(\d+)\s+(\d+)/) {
            $all_tables{$thetab}{$1}[4] = $2;
            $all_tables{$thetab}{$1}[5] = $3;
            $all_tables{$thetab}{$1}[6] = $4;
            $all_tables{$thetab}{$1}[7] = $5;
            $all_tables{$thetab}{$1}[8] = $6;
            $asgn = 0;
        }
    }
    close FL;
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub read_master_files() {
    local($foo);
    #read states
    local($fname) = $MASTERS_DIR."states.txt";
    open(FL,$fname) || die "Could not open file $fname\n";
    while (<FL>) { chop; @m_states = (@m_states,$_); }
    close FL;
    #read address
    $fname = $MASTERS_DIR."addresses.txt";
    open(FL,$fname) || die "Could not open file $fname\n";
    while (<FL>) { chop;@m_addrs = (@m_addrs,$_); }
    close FL;
    $match_addr = join("|",@m_addrs);
    #read cities
    $fname = $MASTERS_DIR."cities.txt";
    open(FL,$fname) || die "Could not open file $fname\n";
    while (<FL>) { chop; @m_cities = (@m_cities,$_); }
    #read names
    local($names_fname);
    foreach  $names_fname ("female.txt","male.txt") {
        $fname = $MASTERS_DIR.$names_fname;
        open(FL,$fname) || die "Could not open file $fname\n";
        while (<FL>) {
            chop;
            #remove ";" from the files
            s/;//g;
            #compress spaces
            foreach $foo (split(' ',$_)) { $m_fnames{$foo} = 1; }
        }
    }
    $fname = $MASTERS_DIR."surname.txt";
    open(FL,$fname) || die "Could not open file $fname\n";
    while (<FL>) {
        chop;
        #remove ";" from the files
        s/;//g;
        #compress spaces
        foreach $foo (split(' ',$_)) { $m_lnames{$foo} = 1;}
    }
}


1;
```

A.6 umls_conn.pl

```
#!/usr/local/bin/perl

# $Source:$
#
# Sample program that tests the fuctionality of the UMLS Metathesaurus
#
# (c) Jiri Schindler, 1996
#
# $Id:$
#

use Socket;

# port to use for TCP connection to VSP
$PORT = 8042;

# name of the host to which we want to connect
$HOST = "isis.nlm.nih.gov";

# -------------------------------------------------------------
# Connects to the UMLS server
# -------------------------------------------------------------
sub umls_connect{
    $iaddr = inet_aton($HOST);
    $paddr = sockaddr_in($PORT,$iaddr);
    $proto = getprotobyname('tcp');
    socket(SocHandle,PF_INET,SOCK_STREAM,$proto) or die $!;

    # Call up the server.
    connect(SocHandle,$paddr)        or die $!;

# Set socket to be command buffered.
    select(SocHandle);
    $| = 1;
    select (STDOUT);
}

sub umls_request{
    local($command) = @_;
    print SocHandle "$command\n";
}

sub umls_read {

    local($cont) = 1;
    while($cont && ($line = <SocHandle>)) {
        print STDOUT $line;
        if ($line =~ /%%/) {
            $cont = 0;
        };
    }
}
sub umls_disconnect {
    shutdown(SocHandle,2);
}

1;
```

A.7 utils.pl

```
#!/usr/local/bin/perl
# -*- Perl -*-
#
# $Id: $
#

require "config.pl";


#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub create_select_stmt {

    local($begining,$ending,@fields) = @_;
    local(@equalities,@tbls,$i);
    if ($begining ne $ending) {
        &bfs_search($begining,$ending);
        for ($i=0;$i<$#table_seq;$i++) {
            push(@tbls,$table_seq[$i]);
            $key = $relations{$table_seq[$i]}{$table_seq[$i+1]}[0];
            push(@equalities,"$table_seq[$i].$key=$table_seq[$i+1].$key");
        }
        push(@tbls,$table_seq[$i]);
    }
    else {
        $tbls[0] = $begining;
    }
    return("select ".join(",",@fields)." from ".join(",",@tbls).
           " where ".join(" AND ",@equalities));
}

#----------------------------------------------------------------------
#
#----------------------------------------------------------------------
sub bfs_search {
    local($startt,$endt) = @_;
    local($el,%vertices,@queue);
    foreach $el (keys(%relations)) {
        $vertices{$el} = [(2,$MAX_SEARCH_LEVEL,"")];
    }
    $vertices{$startt}[0] = [(1,0,"")];
    push(@queue,$startt);
    while ($#queue != -1) {
        $el = shift(@queue);
        foreach $adj (keys(%{$relations{$el}})) {
            if ($vertices{$adj}[0] == 2) {
                $vertices{$adj}[0] = 1;
                $vertices{$adj}[1] += 1;
                $vertices{$adj}[2] = $el;
                if ($adj eq $endt) { # we have a match
                    undef(@table_seq);
                    while ($adj ne "" ) {
                        unshift(@table_seq,$adj);
                        $adj = $vertices{$adj}[2];
                    }
                    return(1);
                }

                push(@queue,$adj);
            }
        }
        $vertices{$el}[0] = 0; # color black
    }
    return(0);
}

#-----------------------------------------------------------------------
#
#-----------------------------------------------------------------------
sub normalize_score {
    local($scor) = @_;
    $scor =~s/(\d.\d\d).*/\1/;
    $scor *= 100;
    return ($scor);
}

1;
```

Appendix B

Result Files

B.1 activity.log

```
Selecting elemets started at 19:52:11 on May 3, 1997
Getting table names started at 19:52:17 on May 3, 1997
Table names obtained in 0.040000 CPU seconds
Getting table names finished at 19:52:17 on May 3, 1997
Table descriptions obtained in 0.530000 CPU seconds
Finding relations started at 19:52:30 on May 3, 1997
All relations obtained in 3.820000 CPU seconds
Finding relations finished at 19:52:39 on May 3, 1997
Tagging fields started at 19:56:08 on May 3, 1997
Tagging fields finished at 20:19:05 on May 3, 1997
Selecting elements started at 20:19:05 on May 3, 1997
Selecting elements finished at 20:19:15 on May 3, 1997
```

B.2 all_field_tags.txt

```
          Field                 DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
PAT_TEST_HISTV: 73421 records
 PAT_TEST_ID                ZIP           72        0          0       0
 PAT_NUM                                   0        0          0       0
 EVENT_START_DT_TM                         0        0          0       0
 EVENT_DT_TM_KEY                           0        0          0       0
 TEST_ID                                   0        0          0       0
 TEST_ABBR                                 0        0          2       0
 EVENT_STOP_DT_TM                          0        0          0       0
 TEST_PRTY_CD               ADDR          11        0          0       0
 REMOTE_SYSTEM_CD                          0        0          0       0
 REMOTE_EVENT_NUM                          0        0          0       0
 SUPERGRP_NUM               ZIP           98        0          0       0
 GRP_NUM                                   0        0          0       0
 DETAIL_NUM                                0        0          0       0
 SUPERGRP_TEST_ID                          0        0          0       0
 GRP_TEST_ID                               0        0          0       0
 PARENT_ID                  ZIP           96        0          0       0
 CHILD_LEVEL_VAL                           0        0          0       0
 ROOT_STATUS                               0        0          0       0
 DATA_CLS_CD                               0        0          0       0
 RSLT_VAL                                  0        0          0       0
 RSLT_UNIT_TXT              ZIP            5        0          0       0
 RSLT_TYPE_CD                              0        0          0       0
 ABN_STATUS                                0        0          0       0
 ABN_TYPE_CD                               0        0          0       0
 REF_LOW_VAL                               0        0          0       0
 REF_HIGH_VAL                              0        0          0       0
 REF_TYPE_CD                               0        0          0       0
 DOCMNT_PNTR_ID                            0        0          0       0
 ORD_ID                                    0        0          0       0
 ORD_PROV_NUM                              0        0          0       0
 ORD_LOCTN                                 0        0          0       0
 RSLT_STATUS                               0        0          0       0
 CONFID_STATUS                             0        0          0       0
 CMNT_STATUS                               0        0          0       0
 ARCH_STATUS                               0        0          0       0
 DOB_STATUS                                0        0          0       0
 RSLT_SCR                                  0        0          0       0
 RSLT_DT_TM                                0        0          0       0
 REMOTE_RPT_DT_TM                          0        0          0       0
 UPDT_DT_TM                                0        0          0       0

          Field                 DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
DOC_STORE: 1128 records
 DOC_ID                                    0        0          0       0
 CONTENT                                   0        0          0       1

          Field                 DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
PERSNL_PUBLIC: 615 records
 PERSNL_ID                  ZIP           92        0          0       0
 LAST_NAME                  LNAME         31        0          2       0
 FIRST_NAME                 FNAME         41        0          0       0
 MID_INITL                                 0        0          0       0
 SUR_NAME                                  0        0          0       0
 TITLE                                     0        0          0       0
 PERSNL_TYPE_DESCR                         0        9          0       0
 DEPT_CD                                   0        0          0       0
 PRIM_WORK_TYPE_CD                         0        0          0       0
 CH_PHONE_NUM                              0        0          0       0
 BEEPER_NUM                                0        0          0       0
 CLSTR_USER_NAME                           0        0          0       0
 CLSTR_ORACLE_USER_NAME                    0         0          0       0
 EMAIL_ADDR                                0         0          0       0
 PAPER_MAIL_ADDR                           0         0          0       0
 VMAIL_ADDR                                0         0          0       0
 AUTH_PERSNL_ID            ZIP            94         0          0       0
 AUTH_END_DT                               0         0          0       0
 REC_VALID_STATUS                          0         0          0       0
 LAST_NAME_SOUNDEX_CD                      0         0          0       0

          Field             DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
PPR: 1417 records
 ROLE                                      0         0          0       0
 PROVIDER_ID               ZIP            98         0          0       0
 PAT_NUM                                   0         0          0       0
 EXT_LAST                                  0         0          0       0
 EXT_FIRST                                 0         0          0       0
 EXT_PROV_NUM                              0         0          0       0
 COMMENTS                                  0         0          0       0
 START_DATE                                0         0          0       0
 END_DATE                                  0         0          0       0
 COST_CENTER                               0         0          0       0

          Field             DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
PROBLEMS: 375 records
 PROBLEM_NAME                              0        42          0       0
 START_DATE                                0         0          0       0
 END_DATE                                  0         0          0       0
 PAT_NUM                                   0         0          0       0
 REVOKE_DATE                               0         0          0       0
 COST_CENTER                               0         0          0       0
 PRIMARY_DIAG_FLG                          0         0          0       0

          Field             DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
CLINICAL_DATA: 7013 records
 DATA_NAME                                 0         0          0       0
 VALUE                                     0         0          0       0
 DATE_OBTAINED                             0         0          0       0
 DATE_MODIFIED                             0         0          0       0
 PAT_NUM                                   0         0          0       0
 TIME_OF_DAY                               0         0          0       0
 DOC_ID                                    0         0          0       0

          Field             DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
DOC_DESCRIPTION: 1128 records
 CREATION                                  0         0          0       0
 LAST_MODIFIED                             0         0          0       0
 PRIMARY_SIGNATORY         ZIP           100         0          0       0
 SECONDARY_SIGNATORY       ZIP           100         0          0       0
 STATUS                                    0         0          0       0
 COMPOUND                                  0         0          0       0
 DOCUMENT_TYPE                             0         0          0       0
 DOC_ID                                    0         0          0       0
 PAT_NUM                                   0         0          0       0
 COST_CENTER                               0         0          0       0
 ESIG_DOC_ID               ZIP           100         0          0       0

          Field             DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
PAT_FIN_ACCT: 7617 records
 PAT_NUM                                   0         0          0       0
 PAT_FIN_ENC_NUM                           0         0          0       0
 CARE_CLS_CD                               0         0         13       0
 STATUS                                    0         0          0       0
 PREADMT_REGSTR_DT_TM                      0         0          0       0
```

BAD_DEBT_DT_TM		0	0	0	0
HIST_DT_TM		0	0	0	0
ACTIV_PAT_NUM		0	0	0	0
ACTIV_PAT_FIN_ENC_NUM		0	0	0	0
ACTIV_PAT_NAME		0	0	0	0
GUAR_NUM	ZIP	98	0	0	0
GUAR_RELTN_CD		0	0	0	0
MARITAL_STATUS		0	0	0	0
GEO_CD		0	0	0	0
RELIG_CD		0	5	5	0
COURTESY_CD		0	0	0	0
SPECL_PROGRM_CD		0	0	0	0
EMPLYMT_STATUS		0	0	0	0
EMPLYR_NAME		0	0	0	0
EMPLYR_EMP_NUM		0	0	5	0
EMPLYR_ADDR		0	0	0	0
EMPLYR_CITY_NAME		0	0	0	0
EMPLYR_STATE_CD		0	0	0	0
EMPLYR_ZIP_CD	ZIP	80	0	0	0
EMPLYR_PHONE_NUM		0	0	0	0
EMPLYR_CNTCT_NAME		0	0	0	0
NURS_STN_CD		0	0	0	0
RM_NUM		0	0	0	0
BED_CD		0	0	0	0
CARE_SUBCLS_CD		0	0	0	0
PRNCPL_PAYOR_CD		0	3	0	0
AR_SUBCLS_CD		0	0	3	0
BAD_DEBT_CLS_CD		0	0	0	0
SERV_CD		0	2	4	0
ADMT_PROV_NUM	ZIP	98	0	0	0
ATND_PROV_NUM	ZIP	92	0	0	0
REF_PROV_NUM	ZIP	98	0	0	0
REF_PROV_NAME		0	0	0	0
MAJ_DRG_CD		0	0	0	0
OUTLR_STATUS		0	0	0	0
DRG_APPRV_STATUS		0	0	0	0
GRPR_REVIEW_CD		0	0	0	0
FINAL_BILL_DRG_CD		0	0	0	0
MAJ_DIAG_CATGRY_NUM		0	0	0	0
PRNCPL_PROCDR_CD		0	0	0	0
PRNCPL_PROCDR_DT		0	0	0	0
PROCDR_PROV_NUM	ZIP	96	0	0	0
PROCDR_PROV_NAME		0	0	0	0
PRNCPL_DIAG_CD	ZIP	17	0	0	0
MAJ_COST_DIAG_CD		0	0	0	0
PREADMT_DT_TM		0	0	0	0
ADMT_DT_TM		0	0	0	0
ADMT_DIAG_CD	ZIP	19	0	0	0
ADMT_DIAG_DESCR	ZIP	5	16	0	0
ADMT_CLS_CD		0	0	5	0
ADMT_SOURCE_CD		0	0	0	0
ADMT_USER_INITL		0	0	0	0
DISCH_DT_TM		0	0	0	0
ANTCPT_DISCH_DT_TM		0	0	0	0
DISCH_DISP_CD		0	0	0	0
DISCH_RELEASE_TXT		0	0	0	0
PREV_NURS_STN_CD		0	0	0	0
PREV_RM_NUM		0	0	0	0
PREV_BED_CD		0	0	0	0
PREV_CARE_SUBCLS_CD		0	0	0	0
PREV_SERV_CD		0	2	5	0
PREV_ATND_PROV_NUM	ZIP	96	0	0	0
PREV_COND_CD		0	0	0	0
PREV_TRNSFR_DT_TM		0	0	0	0
PEND_TRNSFR_DT_TM		0	0	0	0
PEND_TRNSFR_LOCTN_TXT		0	0	0	0
TRNSFR_REASON_TXT		0	0	0	0
TRNSFR_EFFCT_DT_TM		0	0	0	0

```
 OUTPAT_CARE_CLS_CD                      0          0         13        0
 ACCDNT_DT_TM                            0          0          0        0
 ACCDNT_LOCTN_CD                         0          0          0        0
 LAST_OUTPAT_VISIT_DT_TM                 0          0          0        0
 EMER_DEPT_ARRIV_MODE_CD                 0          0          3        0
 EMER_DEPT_DISCH_DT_TM                   0          0          0        0
 EMER_DEPT_DISP_CD                       0          0          0        0
 EMER_DEPT_POLICE_NOTIFY_ST              0          0          0        0
 HEALTH_BOARD_NOTIFY_STATUS              0          0          0        0
 REGSTR_OTHER_TXT                        0          0          0        0
 USER_FLD_TXT                            0          0          0        0
 MED_REC_CHART_NUM                       0          0          0        0
 MED_REC_CHART_LOCTN_CD                  0          0          0        0
 MED_REC_CMNT_TXT                        0          0          0        0
 BLOOD_PROGRM_STATUS                     0          0          0        0
 BLOOD_FURN_CNT                          0          0          0        0
 BLOOD_REPLACE_CNT                       0          0          0        0
 BLOOD_ORGNZTN_NAME                      0          0          0        0
 UB82_LOCTR_02                           0          0          0        0
 UB82_LOCTR_09                           0          0          0        0
 UB82_LOCTR_27                           0          0          0        0
 UB82_LOCTR_45                           0          0          0        0
 OCCUR_SPAN_CD                           0          0          0        0
 OCCUR_BEGIN_DT                          0          0          0        0
 OCCUR_END_DT                            0          0          0        0
 TOTL_GROSS_CHRG_AMT                     0          0          0        0
 LATE_ACTIV_AMT                          0          0          0        0
 LATE_ACTIV_ADJST_AMT                    0          0          0        0
 LAST_PAT_CYCLE_BILL_DT_TM               0          0          0        0
 LAST_INS_CYCLE_BILL_DT_TM               0          0          0        0
 FINAL_BILL_DT_TM                        0          0          0        0
 LAST_ACTIV_DT_TM                        0          0          0        0
 REBILL_STATUS                           0          0          0        0
 LAST_PRORAT_DT_TM                       0          0          0        0
 LAST_POST_DT_TM                         0          0          0        0
 BAD_DEBT_WRITE_OFF_AMT                  0          0          0        0
 BAD_DEBT_WRITE_OFF_RECOVER              0          0          0        0
 ACCT_REPRSNT_CD                         0          0          0        0
 DIRECT_AR_STATUS                        0          0          0        0
 LAST_STMT_DT_TM                         0          0          0        0
 STMT_SENT_CNT                           0          0          0        0
 STMT_THREAT_CD                          0          0          0        0
 RECLS_SUPPRESS_STATUS                   0          0          0        0
 FIRST_STMT_STATUS                       0          0          0        0
 FIRST_STMT_DAY_CNT                      0          0          0        0
 OTHER_STMT_STATUS                       0          0          0        0
 OTHER_STMT_DAY_CD                       0          0          0        0
 STMT_SMALL_BAL_STATUS                   0          0          0        0
 STMT_FORM_CLS_CD                        0          0          0        0
 STMT_MSG_SUPPRESS_STATUS                0          0          0        0
 GUAR_CNTRCT_CTRL_NUM                    0          0          0        0
 GUAR_CNTRCT_NUM                         0          0          0        0
 CONVR_STATUS                            0          0          0        0
 CYCLE_UPDT_DT_TM                        0          0          0        0
 MED_REC_UPDT_STATUS                     0          0          0        0
 FACIL_TRNSFR_FROM_CD                    0          0          0        0
 FACIL_TRNSFR_TO_CD                      0          0          0        0
 UPDT_USER_INITL                         0          0          0        0
 UPDT_DEPT_CD                            0          4          6        0
 UPDT_DT_TM                              0          0          0        0

         Field            DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
PAT_DEMOGRAPH: 300 records
 PAT_NUM                                 0          0          0        0
 LAST_NAME                LNAME         27          0          0        0
 FIRST_NAME               FNAME         88          0          0        0
 MID_INITL                               0          0          0        0
 TITLE                                0        0          4       0
 SSN                                  0        0          0       0
 PREV_NUM                             0        0          0       0
 PREV_NAME                            0        0          0       0
 STREET_ADDR               ADDR      43        0          0       0
 OTHER_ADDR                           0        0          0       0
 CITY_NAME                 LNAME     88        0          0       0
 STATE_CD                  ADDR       9        0         10       0
 ZIP_CD                    ZIP      100        0          0       0
 PHONE_NUM                            0        0          0       0
 SEX_CD                               0        0          0       0
 DOB                                  0        0          0       0
 HGHT_VAL                             0        0          0       0
 WGT_VAL                              0        0          0       0
 BIRTH_PLACE_NAME                     0        0          0       0
 CH_EMP_STATUS                        0        0          0       0
 RACE_CD                              0        0          5       0
 ALLERGY_TXT                          0        0         28       0
 DEATH_DT_TM                          0        0          0       0
 BAD_ADDR_STATUS                      0        0          0       0
 SPOUSE_FIRST_NAME                    0        0          0       0
 MAIDEN_NAME                          0        0          0       0
 FAMILY_PROV_NAME                     0        0          0       0
 DIS_EXPOSR_STATUS                    0        0          0       0
 LAST_OUTPAT_FIN_ENC_NUM              0        0          0       0
 LAST_INPAT_FIN_ENC_NUM               0        0          0       0
 EMER_NOTIFY_NAME                     0        0          0       0
 EMER_NOTIFY_PHONE_NUM_1              0        0          0       0
 EMER_NOTIFY_PHONE_NUM_2              0        0          0       0
 EMER_NOTIFY_RELTN_CD                 0        0          0       0
 NEXT_KIN_NAME                        0        0          0       0
 NEXT_KIN_ADDR                        0        0          0       0
 NEXT_KIN_CITY_NAME                   0        0          0       0
 NEXT_KIN_STATE_CD                    0        0          0       0
 NEXT_KIN_ZIP_CD                      0        0          0       0
 NEXT_KIN_PHONE_NUM                   0        0          0       0
 NEXT_KIN_RELTN_CD                    0        0          0       0
 NEXT_KIN_EMPLYR_NAME                 0        0          0       0
 NEXT_KIN_EMPLYR_CITY_NAME            0        0          0       0
 NEXT_KIN_EMPLYR_STATE_CD             0        0          0       0
 NEXT_KIN_EMPLYR_ZIP_CD               0        0          0       0
 NEXT_KIN_EMPLYR_PHONE_NUM            0        0          0       0
 MEHC_FAMILY_NUM                      0        0          0       0
 MEHC_MEMBR_NUM                       0        0          0       0
 OTHER_HEALTH_CENTR_NUM               0        0          0       0
 OTHER_HEALTH_CENTR_ABBR              0        0          0       0
 NATIVE_LANG_CD                       0        0          0       0
 INTERP_NEED_STATUS                   0        0          0       0
 USER_FLD_TXT                         0        0          0       0
 PREV_OUTPAT_FIN_ENC_NUM              0        0          0       0
 PREV_INPAT_FIN_ENC_NUM               0        0          0       0
 LAST_ASSIGN_FIN_ENC_NUM              0        0          0       0
 LAST_DISCH_DT                        0        0          0       0
 UPDT_USER_INITL                      0        0          0       0
 UPDT_DEPT_CD                         0        0          0       0
 UPDT_DT_TM                           0        0          0       0

         Field             DemType  DemScore  Problems  Medications  Notes
-----------------------------------------------------------------------------
CHILD_DOCS: 1619 records
 DOC_ID                               0        0          0       0
 CHILD_ID                             0        0          0       0
 CHILD_NAME                           0        0          0       0

         Field             DemType  DemScore  Problems  Medications  Notes
-----------------------------------------------------------------------------
DOC_ATTRIBUTES: 3394 records
 ATTRIBUTE                            0        0          0       0
 VALUE                                    0         0          0       0
 DOC_ID                                   0         0          0       0

          Field             DemType  DemScore  Problems  Medications Notes
-------------------------------------------------------------------------------
PHARMACY_TABLE: 3668 records
 PATIENT_RECORD_NUMBER                    0         0          0       0
 PATIENT_VISIT_NUMBER                     0         0          0       0
 PATIENT_NAME                             0         0          0       0
 FUNCTION_CODE                            0         0          0       0
 DATE_OF_SERVICE                          0         0          0       0
 FORMULARY_CODE_1           ZIP         100         0          0       0
 FORMULARY_CODE_2           ZIP         100         0          0       0
 RX_CODE                                  0         0          0       0
 MEDICATION_NAME            ADDR          3         0         60       0
 SERVICE_QUANTITY                         0         0          0       0
 UNIT_PRICE                               0         0          0       0
 DOSE_FEE                                 0         0          0       0
 LABOR_FEE                                0         0          0       0
 TOTAL_PRICE                              0         0          0       0
 MED_TYPE                                 0         0          0       0
 DRG_CODE                                 0         0          0       0
 ORDERING_DOC                             0         0          0       0
 LABOR_EXPENSE_CODE                       0         0          0       0
 DRUG_DOSE                                0         0          0       0
 FORMULARY_CLASS_1                        0         0          0       0
 FORMULARY_CLASS_2                        0         0          0       0
 DRUG_ROLE                                0         0          0       0
 ETXG_SEQUENCE              ZIP         100         0          0       0
 NURS_LOC                                 0         0          0       0
```

B.3 all_matches.txt

```
#All SELECTs and JOINs performed to determine relations

Table: CHILD_DOCS (1619 records)
Table: CLINICAL_DATA (7013 records)
Table: DOC_ATTRIBUTES (3394 records)
Table: DOC_DESCRIPTION (1128 records)
Table: DOC_STORE (1128 records)
Table: PAT_DEMOGRAPH (300 records)
Table: PAT_FIN_ACCT (7617 records)
Table: PAT_TEST_HISTV (73421 records)
Table: PERSNL_PUBLIC (615 records)
Table: PHARMACY_TABLE (3668 records)
Table: PPR (1417 records)
Table: PROBLEMS (375 records)
```

B.4 all_tables.txt

```
CLINICAL_DATA: 7013 records
        DATA_NAME [VARCHAR2,80,Y,1]
        VALUE [NUMBER,22,Y,2]
        DATE_OBTAINED [DATE,7,Y,3]
        DATE_MODIFIED [DATE,7,Y,4]
        PAT_NUM [NUMBER,22,Y,5]
        TIME_OF_DAY [VARCHAR2,5,Y,6]
        DOC_ID [NUMBER,22,Y,7]

PAT_TEST_HISTV: 73421 records
        PAT_TEST_ID [NUMBER,22,N,1]
        PAT_NUM [NUMBER,22,N,2]
        EVENT_START_DT_TM [DATE,7,N,3]
        EVENT_DT_TM_KEY [NUMBER,22,N,4]
        TEST_ID [NUMBER,22,N,5]
        TEST_ABBR [VARCHAR2,10,Y,6]
        EVENT_STOP_DT_TM [DATE,7,Y,7]
        TEST_PRTY_CD [VARCHAR2,2,Y,8]
        REMOTE_SYSTEM_CD [VARCHAR2,5,N,9]
        REMOTE_EVENT_NUM [VARCHAR2,15,N,10]
        SUPERGRP_NUM [NUMBER,22,Y,11]
        GRP_NUM [NUMBER,22,Y,12]
        DETAIL_NUM [NUMBER,22,Y,13]
        SUPERGRP_TEST_ID [NUMBER,22,Y,14]
        GRP_TEST_ID [NUMBER,22,Y,15]
        PARENT_ID [NUMBER,22,Y,16]
        CHILD_LEVEL_VAL [NUMBER,22,Y,17]
        ROOT_STATUS [VARCHAR2,1,Y,18]
        DATA_CLS_CD [VARCHAR2,2,Y,19]
        RSLT_VAL [VARCHAR2,8,Y,20]
        RSLT_UNIT_TXT [VARCHAR2,10,Y,21]
        RSLT_TYPE_CD [VARCHAR2,1,Y,22]
        ABN_STATUS [VARCHAR2,2,Y,23]
        ABN_TYPE_CD [VARCHAR2,2,Y,24]
        REF_LOW_VAL [VARCHAR2,10,Y,25]
        REF_HIGH_VAL [VARCHAR2,10,Y,26]
        REF_TYPE_CD [VARCHAR2,1,Y,27]
        DOCMNT_PNTR_ID [NUMBER,22,Y,28]
        ORD_ID [NUMBER,22,Y,29]
        ORD_PROV_NUM [NUMBER,22,Y,30]
        ORD_LOCTN [VARCHAR2,4,Y,31]
        RSLT_STATUS [VARCHAR2,2,Y,32]
        CONFID_STATUS [VARCHAR2,1,Y,33]
        CMNT_STATUS [VARCHAR2,1,Y,34]
        ARCH_STATUS [VARCHAR2,1,Y,35]
        DOB_STATUS [VARCHAR2,1,Y,36]
        RSLT_SCR [NUMBER,22,Y,37]
        RSLT_DT_TM [DATE,7,Y,38]
        REMOTE_RPT_DT_TM [DATE,7,Y,39]
        UPDT_DT_TM [DATE,7,N,40]

DOC_DESCRIPTION: 1128 records
        CREATION [DATE,7,Y,1]
        LAST_MODIFIED [DATE,7,Y,2]
        PRIMARY_SIGNATORY [NUMBER,22,Y,3]
        SECONDARY_SIGNATORY [NUMBER,22,Y,4]
        STATUS [VARCHAR2,20,Y,5]
        COMPOUND [VARCHAR2,5,Y,6]
        DOCUMENT_TYPE [VARCHAR2,20,Y,7]
        DOC_ID [NUMBER,22,Y,8]
        PAT_NUM [NUMBER,22,Y,9]
        COST_CENTER [NUMBER,22,Y,10]
        ESIG_DOC_ID [NUMBER,22,Y,11]

PPR: 1417 records
```

```
        ROLE [VARCHAR2,20,Y,1]
        PROVIDER_ID [NUMBER,22,Y,2]
        PAT_NUM [NUMBER,22,Y,3]
        EXT_LAST [VARCHAR2,50,Y,4]
        EXT_FIRST [VARCHAR2,50,Y,5]
        EXT_PROV_NUM [NUMBER,22,Y,6]
        COMMENTS [VARCHAR2,80,Y,7]
        START_DATE [DATE,7,Y,8]
        END_DATE [DATE,7,Y,9]
        COST_CENTER [NUMBER,22,Y,10]

PERSNL_PUBLIC: 615 records
        PERSNL_ID [NUMBER,22,N,1]
        LAST_NAME [VARCHAR2,20,N,2]
        FIRST_NAME [VARCHAR2,20,Y,3]
        MID_INITL [VARCHAR2,1,Y,4]
        SUR_NAME [VARCHAR2,3,Y,5]
        TITLE [VARCHAR2,10,Y,6]
        PERSNL_TYPE_DESCR [VARCHAR2,20,Y,7]
        DEPT_CD [VARCHAR2,6,Y,8]
        PRIM_WORK_TYPE_CD [VARCHAR2,5,Y,9]
        CH_PHONE_NUM [VARCHAR2,7,Y,10]
        BEEPER_NUM [VARCHAR2,7,Y,11]
        CLSTR_USER_NAME [VARCHAR2,20,Y,12]
        CLSTR_ORACLE_USER_NAME [VARCHAR2,30,Y,13]
        EMAIL_ADDR [VARCHAR2,50,Y,14]
        PAPER_MAIL_ADDR [VARCHAR2,50,Y,15]
        VMAIL_ADDR [VARCHAR2,15,Y,16]
        AUTH_PERSNL_ID [NUMBER,22,Y,17]
        AUTH_END_DT [DATE,7,Y,18]
        REC_VALID_STATUS [VARCHAR2,1,N,19]
        LAST_NAME_SOUNDEX_CD [VARCHAR2,4,N,20]

DOC_STORE: 1128 records
        DOC_ID [NUMBER,22,Y,1]
        CONTENT [LONG,0,Y,2]

PAT_FIN_ACCT: 7617 records
        PAT_NUM [NUMBER,22,N,1]
        PAT_FIN_ENC_NUM [NUMBER,22,Y,2]
        CARE_CLS_CD [VARCHAR2,1,Y,3]
        STATUS [VARCHAR2,1,Y,4]
        PREADMT_REGSTR_DT_TM [DATE,7,Y,5]
        BAD_DEBT_DT_TM [DATE,7,Y,6]
        HIST_DT_TM [DATE,7,Y,7]
        ACTIV_PAT_NUM [NUMBER,22,Y,8]
        ACTIV_PAT_FIN_ENC_NUM [NUMBER,22,Y,9]
        ACTIV_PAT_NAME [VARCHAR2,31,Y,10]
        GUAR_NUM [NUMBER,22,Y,11]
        GUAR_RELTN_CD [VARCHAR2,2,Y,12]
        MARITAL_STATUS [VARCHAR2,1,Y,13]
        GEO_CD [VARCHAR2,6,Y,14]
        RELIG_CD [VARCHAR2,3,Y,15]
        COURTESY_CD [VARCHAR2,1,Y,16]
        SPECL_PROGRM_CD [VARCHAR2,2,Y,17]
        EMPLYMT_STATUS [VARCHAR2,1,Y,18]
        EMPLYR_NAME [VARCHAR2,25,Y,19]
        EMPLYR_EMP_NUM [VARCHAR2,11,Y,20]
        EMPLYR_ADDR [VARCHAR2,30,Y,21]
        EMPLYR_CITY_NAME [VARCHAR2,15,Y,22]
        EMPLYR_STATE_CD [VARCHAR2,2,Y,23]
        EMPLYR_ZIP_CD [VARCHAR2,9,Y,24]
        EMPLYR_PHONE_NUM [VARCHAR2,11,Y,25]
        EMPLYR_CNTCT_NAME [VARCHAR2,25,Y,26]
        NURS_STN_CD [VARCHAR2,4,Y,27]
        RM_NUM [VARCHAR2,4,Y,28]
        BED_CD [VARCHAR2,2,Y,29]
        CARE_SUBCLS_CD [VARCHAR2,1,Y,30]
```

```
PRNCPL_PAYOR_CD [VARCHAR2,2,Y,31]
AR_SUBCLS_CD [VARCHAR2,3,Y,32]
BAD_DEBT_CLS_CD [VARCHAR2,3,Y,33]
SERV_CD [VARCHAR2,3,Y,34]
ADMT_PROV_NUM [NUMBER,22,Y,35]
ATND_PROV_NUM [NUMBER,22,Y,36]
REF_PROV_NUM [NUMBER,22,Y,37]
REF_PROV_NAME [VARCHAR2,25,Y,38]
MAJ_DRG_CD [NUMBER,22,Y,39]
OUTLR_STATUS [VARCHAR2,1,Y,40]
DRG_APPRV_STATUS [VARCHAR2,1,Y,41]
GRPR_REVIEW_CD [NUMBER,22,Y,42]
FINAL_BILL_DRG_CD [NUMBER,22,Y,43]
MAJ_DIAG_CATGRY_NUM [NUMBER,22,Y,44]
PRNCPL_PROCDR_CD [VARCHAR2,5,Y,45]
PRNCPL_PROCDR_DT [DATE,7,Y,46]
PROCDR_PROV_NUM [NUMBER,22,Y,47]
PROCDR_PROV_NAME [VARCHAR2,25,Y,48]
PRNCPL_DIAG_CD [VARCHAR2,6,Y,49]
MAJ_COST_DIAG_CD [VARCHAR2,6,Y,50]
PREADMT_DT_TM [DATE,7,Y,51]
ADMT_DT_TM [DATE,7,Y,52]
ADMT_DIAG_CD [VARCHAR2,6,Y,53]
ADMT_DIAG_DESCR [VARCHAR2,70,Y,54]
ADMT_CLS_CD [VARCHAR2,1,Y,55]
ADMT_SOURCE_CD [VARCHAR2,1,Y,56]
ADMT_USER_INITL [VARCHAR2,3,Y,57]
DISCH_DT_TM [DATE,7,Y,58]
ANTCPT_DISCH_DT_TM [DATE,7,Y,59]
DISCH_DISP_CD [VARCHAR2,2,Y,60]
DISCH_RELEASE_TXT [VARCHAR2,50,Y,61]
PREV_NURS_STN_CD [VARCHAR2,4,Y,62]
PREV_RM_NUM [VARCHAR2,4,Y,63]
PREV_BED_CD [VARCHAR2,2,Y,64]
PREV_CARE_SUBCLS_CD [VARCHAR2,1,Y,65]
PREV_SERV_CD [VARCHAR2,3,Y,66]
PREV_ATND_PROV_NUM [NUMBER,22,Y,67]
PREV_COND_CD [VARCHAR2,1,Y,68]
PREV_TRNSFR_DT_TM [DATE,7,Y,69]
PEND_TRNSFR_DT_TM [DATE,7,Y,70]
PEND_TRNSFR_LOCTN_TXT [VARCHAR2,10,Y,71]
TRNSFR_REASON_TXT [VARCHAR2,30,Y,72]
TRNSFR_EFFCT_DT_TM [DATE,7,Y,73]
OUTPAT_CARE_CLS_CD [VARCHAR2,1,Y,74]
ACCDNT_DT_TM [DATE,7,Y,75]
ACCDNT_LOCTN_CD [VARCHAR2,2,Y,76]
LAST_OUTPAT_VISIT_DT_TM [DATE,7,Y,77]
EMER_DEPT_ARRIV_MODE_CD [VARCHAR2,1,Y,78]
EMER_DEPT_DISCH_DT_TM [DATE,7,Y,79]
EMER_DEPT_DISP_CD [VARCHAR2,2,Y,80]
EMER_DEPT_POLICE_NOTIFY_STATUS [VARCHAR2,1,Y,81]
HEALTH_BOARD_NOTIFY_STATUS [VARCHAR2,1,Y,82]
REGSTR_OTHER_TXT [VARCHAR2,60,Y,83]
USER_FLD_TXT [VARCHAR2,100,Y,84]
MED_REC_CHART_NUM [VARCHAR2,6,Y,85]
MED_REC_CHART_LOCTN_CD [VARCHAR2,5,Y,86]
MED_REC_CMNT_TXT [VARCHAR2,30,Y,87]
BLOOD_PROGRM_STATUS [VARCHAR2,1,Y,88]
BLOOD_FURN_CNT [NUMBER,22,Y,89]
BLOOD_REPLACE_CNT [NUMBER,22,Y,90]
BLOOD_ORGNZTN_NAME [VARCHAR2,8,Y,91]
UB82_LOCTR_02 [VARCHAR2,30,Y,92]
UB82_LOCTR_09 [VARCHAR2,7,Y,93]
UB82_LOCTR_27 [VARCHAR2,8,Y,94]
UB82_LOCTR_45 [VARCHAR2,17,Y,95]
OCCUR_SPAN_CD [VARCHAR2,2,Y,96]
OCCUR_BEGIN_DT [DATE,7,Y,97]
OCCUR_END_DT [DATE,7,Y,98]
```

```
        TOTL_GROSS_CHRG_AMT [NUMBER,22,Y,99]
        LATE_ACTIV_AMT [NUMBER,22,Y,100]
        LATE_ACTIV_ADJST_AMT [NUMBER,22,Y,101]
        LAST_PAT_CYCLE_BILL_DT_TM [DATE,7,Y,102]
        LAST_INS_CYCLE_BILL_DT_TM [DATE,7,Y,103]
        FINAL_BILL_DT_TM [DATE,7,Y,104]
        LAST_ACTIV_DT_TM [DATE,7,Y,105]
        REBILL_STATUS [VARCHAR2,1,Y,106]
        LAST_PRORAT_DT_TM [DATE,7,Y,107]
        LAST_POST_DT_TM [DATE,7,Y,108]
        BAD_DEBT_WRITE_OFF_AMT [NUMBER,22,Y,109]
        BAD_DEBT_WRITE_OFF_RECOVER_AMT [NUMBER,22,Y,110]
        ACCT_REPRSNT_CD [VARCHAR2,3,Y,111]
        DIRECT_AR_STATUS [VARCHAR2,1,Y,112]
        LAST_STMT_DT_TM [DATE,7,Y,113]
        STMT_SENT_CNT [NUMBER,22,Y,114]
        STMT_THREAT_CD [NUMBER,22,Y,115]
        RECLS_SUPPRESS_STATUS [VARCHAR2,1,Y,116]
        FIRST_STMT_STATUS [VARCHAR2,1,Y,117]
        FIRST_STMT_DAY_CNT [NUMBER,22,Y,118]
        OTHER_STMT_STATUS [VARCHAR2,1,Y,119]
        OTHER_STMT_DAY_CD [NUMBER,22,Y,120]
        STMT_SMALL_BAL_STATUS [VARCHAR2,1,Y,121]
        STMT_FORM_CLS_CD [VARCHAR2,1,Y,122]
        STMT_MSG_SUPPRESS_STATUS [VARCHAR2,1,Y,123]
        GUAR_CNTRCT_CTRL_NUM [NUMBER,22,Y,124]
        GUAR_CNTRCT_NUM [NUMBER,22,Y,125]
        CONVR_STATUS [VARCHAR2,1,Y,126]
        CYCLE_UPDT_DT_TM [DATE,7,Y,127]
        MED_REC_UPDT_STATUS [VARCHAR2,1,Y,128]
        FACIL_TRNSFR_FROM_CD [VARCHAR2,5,Y,129]
        FACIL_TRNSFR_TO_CD [VARCHAR2,5,Y,130]
        UPDT_USER_INITL [VARCHAR2,3,Y,131]
        UPDT_DEPT_CD [VARCHAR2,3,Y,132]
        UPDT_DT_TM [DATE,7,Y,133]

PAT_DEMOGRAPH: 300 records
        PAT_NUM [NUMBER,22,N,1]
        LAST_NAME [VARCHAR2,16,Y,2]
        FIRST_NAME [VARCHAR2,11,Y,3]
        MID_INITL [VARCHAR2,1,Y,4]
        TITLE [VARCHAR2,3,Y,5]
        SSN [NUMBER,22,Y,6]
        PREV_NUM [NUMBER,22,Y,7]
        PREV_NAME [VARCHAR2,31,Y,8]
        STREET_ADDR [VARCHAR2,25,Y,9]
        OTHER_ADDR [VARCHAR2,25,Y,10]
        CITY_NAME [VARCHAR2,15,Y,11]
        STATE_CD [VARCHAR2,2,Y,12]
        ZIP_CD [VARCHAR2,9,Y,13]
        PHONE_NUM [VARCHAR2,11,Y,14]
        SEX_CD [VARCHAR2,1,Y,15]
        DOB [DATE,7,Y,16]
        HGHT_VAL [NUMBER,22,Y,17]
        WGT_VAL [NUMBER,22,Y,18]
        BIRTH_PLACE_NAME [VARCHAR2,15,Y,19]
        CH_EMP_STATUS [VARCHAR2,1,Y,20]
        RACE_CD [VARCHAR2,1,Y,21]
        ALLERGY_TXT [VARCHAR2,25,Y,22]
        DEATH_DT_TM [DATE,7,Y,23]
        BAD_ADDR_STATUS [VARCHAR2,1,Y,24]
        SPOUSE_FIRST_NAME [VARCHAR2,11,Y,25]
        MAIDEN_NAME [VARCHAR2,16,Y,26]
        FAMILY_PROV_NAME [VARCHAR2,25,Y,27]
        DIS_EXPOSR_STATUS [VARCHAR2,1,Y,28]
        LAST_OUTPAT_FIN_ENC_NUM [NUMBER,22,Y,29]
        LAST_INPAT_FIN_ENC_NUM [NUMBER,22,Y,30]
        EMER_NOTIFY_NAME [VARCHAR2,25,Y,31]
        EMER_NOTIFY_PHONE_NUM_1 [VARCHAR2,11,Y,32]
        EMER_NOTIFY_PHONE_NUM_2 [VARCHAR2,11,Y,33]
        EMER_NOTIFY_RELTN_CD [VARCHAR2,2,Y,34]
        NEXT_KIN_NAME [VARCHAR2,25,Y,35]
        NEXT_KIN_ADDR [VARCHAR2,25,Y,36]
        NEXT_KIN_CITY_NAME [VARCHAR2,15,Y,37]
        NEXT_KIN_STATE_CD [VARCHAR2,2,Y,38]
        NEXT_KIN_ZIP_CD [VARCHAR2,9,Y,39]
        NEXT_KIN_PHONE_NUM [VARCHAR2,11,Y,40]
        NEXT_KIN_RELTN_CD [VARCHAR2,2,Y,41]
        NEXT_KIN_EMPLYR_NAME [VARCHAR2,25,Y,42]
        NEXT_KIN_EMPLYR_CITY_NAME [VARCHAR2,15,Y,43]
        NEXT_KIN_EMPLYR_STATE_CD [VARCHAR2,2,Y,44]
        NEXT_KIN_EMPLYR_ZIP_CD [VARCHAR2,9,Y,45]
        NEXT_KIN_EMPLYR_PHONE_NUM [VARCHAR2,11,Y,46]
        MEHC_FAMILY_NUM [VARCHAR2,5,Y,47]
        MEHC_MEMBR_NUM [VARCHAR2,2,Y,48]
        OTHER_HEALTH_CENTR_NUM [VARCHAR2,15,Y,49]
        OTHER_HEALTH_CENTR_ABBR [VARCHAR2,4,Y,50]
        NATIVE_LANG_CD [VARCHAR2,2,Y,51]
        INTERP_NEED_STATUS [VARCHAR2,1,Y,52]
        USER_FLD_TXT [VARCHAR2,10,Y,53]
        PREV_OUTPAT_FIN_ENC_NUM [NUMBER,22,Y,54]
        PREV_INPAT_FIN_ENC_NUM [NUMBER,22,Y,55]
        LAST_ASSIGN_FIN_ENC_NUM [NUMBER,22,Y,56]
        LAST_DISCH_DT [DATE,7,Y,57]
        UPDT_USER_INITL [VARCHAR2,3,Y,58]
        UPDT_DEPT_CD [VARCHAR2,3,Y,59]
        UPDT_DT_TM [DATE,7,Y,60]

PROBLEMS: 375 records
        PROBLEM_NAME [VARCHAR2,80,Y,1]
        START_DATE [DATE,7,Y,2]
        END_DATE [DATE,7,Y,3]
        PAT_NUM [NUMBER,22,Y,4]
        REVOKE_DATE [DATE,7,Y,5]
        COST_CENTER [NUMBER,22,Y,6]
        PRIMARY_DIAG_FLG [NUMBER,22,Y,7]

PHARMACY_TABLE: 3668 records
        PATIENT_RECORD_NUMBER [NUMBER,22,Y,1]
        PATIENT_VISIT_NUMBER [NUMBER,22,Y,2]
        PATIENT_NAME [VARCHAR2,30,Y,3]
        FUNCTION_CODE [VARCHAR2,3,Y,4]
        DATE_OF_SERVICE [DATE,7,Y,5]
        FORMULARY_CODE_1 [VARCHAR2,5,Y,6]
        FORMULARY_CODE_2 [VARCHAR2,5,Y,7]
        RX_CODE [VARCHAR2,9,Y,8]
        MEDICATION_NAME [VARCHAR2,30,Y,9]
        SERVICE_QUANTITY [NUMBER,22,Y,10]
        UNIT_PRICE [NUMBER,22,Y,11]
        DOSE_FEE [NUMBER,22,Y,12]
        LABOR_FEE [NUMBER,22,Y,13]
        TOTAL_PRICE [NUMBER,22,Y,14]
        MED_TYPE [VARCHAR2,1,Y,15]
        DRG_CODE [VARCHAR2,3,Y,16]
        ORDERING_DOC [VARCHAR2,25,Y,17]
        LABOR_EXPENSE_CODE [VARCHAR2,1,Y,18]
        DRUG_DOSE [NUMBER,22,Y,19]
        FORMULARY_CLASS_1 [VARCHAR2,3,Y,20]
        FORMULARY_CLASS_2 [VARCHAR2,3,Y,21]
        DRUG_ROLE [VARCHAR2,1,Y,22]
        ETXG_SEQUENCE [NUMBER,22,Y,23]
        NURS_LOC [VARCHAR2,4,Y,24]

DOC_ATTRIBUTES: 3394 records
        ATTRIBUTE [VARCHAR2,20,Y,1]
        VALUE [VARCHAR2,230,Y,2]
        DOC_ID [NUMBER,22,Y,3]

CHILD_DOCS: 1619 records
        DOC_ID [NUMBER,22,Y,1]
        CHILD_ID [NUMBER,22,Y,2]
        CHILD_NAME [VARCHAR2,20,Y,3]
```

B.5 name_keys.txt

```
TITLE:
   PERSNL_PUBLIC,PAT_DEMOGRAPH
DOC_ID:
   CLINICAL_DATA,DOC_DESCRIPTION,DOC_STORE,DOC_ATTRIBUTES,CHILD_DOCS
USER_FLD_TXT:
   PAT_FIN_ACCT,PAT_DEMOGRAPH
END_DATE:
   PPR,PROBLEMS
LAST_NAME:
   PERSNL_PUBLIC,PAT_DEMOGRAPH
UPDT_DT_TM:
   PAT_TEST_HISTV,PAT_FIN_ACCT,PAT_DEMOGRAPH
STATUS:
   DOC_DESCRIPTION,PAT_FIN_ACCT
PAT_NUM:
   CLINICAL_DATA,PAT_TEST_HISTV,DOC_DESCRIPTION,PPR,PAT_FIN_ACCT,PAT_DEMOGRAPH,PROBLEMS
FIRST_NAME:
   PERSNL_PUBLIC,PAT_DEMOGRAPH
MID_INITL:
   PERSNL_PUBLIC,PAT_DEMOGRAPH
COST_CENTER:
   DOC_DESCRIPTION,PPR,PROBLEMS
UPDT_USER_INITL:
   PAT_FIN_ACCT,PAT_DEMOGRAPH
START_DATE:
   PPR,PROBLEMS
UPDT_DEPT_CD:
   PAT_FIN_ACCT,PAT_DEMOGRAPH
```

B.6 select_statements.txt

```
DEMOGRAPHICS: ADDR ZIP LNAME FNAME
        select PAT_DEMOGRAPH.STREET_ADDR,PAT_DEMOGRAPH.ZIP_CD,PAT_DEMOGRAPH.
          LAST_NAME,PAT_DEMOGRAPH.FIRST_NAME from PAT_DEMOGRAPH where  <clause>

MEDICATIONS:
        select PHARMACY_TABLE.MEDICATION_NAME from  where  AND <clause>

PROBLEMS:
        select PROBLEMS.PROBLEM_NAME from PAT_DEMOGRAPH,PROBLEMS where
          PAT_DEMOGRAPH.PAT_NUM=PROBLEMS.PAT_NUM AND <clause>

NOTES:
        select DOC_STORE.CONTENT from PAT_DEMOGRAPH,CLINICAL_DATA,DOC_STORE
          where PAT_DEMOGRAPH.PAT_NUM=CLINICAL_DATA.PAT_NUM AND
          CLINICAL_DATA.DOC_ID=DOC_STORE.DOC_ID AND <clause>
```

B.7 table_names.txt

```
CHILD_DOCS
CLINICAL_DATA
DOC_ATTRIBUTES
DOC_DESCRIPTION
DOC_STORE
PAT_DEMOGRAPH
PAT_FIN_ACCT
PAT_TEST_HISTV
PERSNL_PUBLIC
PHARMACY_TABLE
PPR
PROBLEMS
```

B.8 table_relations.txt

```
CLINICAL_DATA:
        PAT_TEST_HISTV [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PPR [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        DOC_STORE [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        PAT_FIN_ACCT [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_DEMOGRAPH [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PROBLEMS [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        DOC_ATTRIBUTES [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        CHILD_DOCS [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)

PAT_TEST_HISTV:
        CLINICAL_DATA [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PPR [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_FIN_ACCT [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_DEMOGRAPH [UPDT_DT_TM->UPDT_DT_TM]  (NAME[1],DATA[0],0)
        PROBLEMS [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)

DOC_DESCRIPTION:
        CLINICAL_DATA [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_TEST_HISTV [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PPR [COST_CENTER->COST_CENTER]  (NAME[1],DATA[0],0)
        DOC_STORE [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        PAT_FIN_ACCT [STATUS->STATUS]  (NAME[1],DATA[0],0)
        PAT_DEMOGRAPH [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PROBLEMS [COST_CENTER->COST_CENTER]  (NAME[1],DATA[0],0)
        DOC_ATTRIBUTES [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        CHILD_DOCS [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)

PERSNL_PUBLIC:
        PAT_DEMOGRAPH [MID_INITL->MID_INITL]  (NAME[1],DATA[0],0)

PPR:
        CLINICAL_DATA [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_TEST_HISTV [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [COST_CENTER->COST_CENTER]  (NAME[1],DATA[0],0)
        PAT_FIN_ACCT [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_DEMOGRAPH [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PROBLEMS [START_DATE->START_DATE]  (NAME[1],DATA[0],0)

DOC_STORE:
        CLINICAL_DATA [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        DOC_ATTRIBUTES [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        CHILD_DOCS [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)

PAT_FIN_ACCT:
        CLINICAL_DATA [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_TEST_HISTV [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [STATUS->STATUS]  (NAME[1],DATA[0],0)
        PPR [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_DEMOGRAPH [UPDT_DT_TM->UPDT_DT_TM]  (NAME[1],DATA[0],0)
        PROBLEMS [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)

PAT_DEMOGRAPH:
        CLINICAL_DATA [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_TEST_HISTV [UPDT_DT_TM->UPDT_DT_TM]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PERSNL_PUBLIC [MID_INITL->MID_INITL]  (NAME[1],DATA[0],0)
        PPR [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_FIN_ACCT [UPDT_DT_TM->UPDT_DT_TM]  (NAME[1],DATA[0],0)
        PROBLEMS [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)

PROBLEMS:
        CLINICAL_DATA [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_TEST_HISTV [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [COST_CENTER->COST_CENTER]  (NAME[1],DATA[0],0)
        PPR [START_DATE->START_DATE]  (NAME[1],DATA[0],0)
        PAT_FIN_ACCT [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)
        PAT_DEMOGRAPH [PAT_NUM->PAT_NUM]  (NAME[1],DATA[0],0)

DOC_ATTRIBUTES:
        CLINICAL_DATA [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        DOC_STORE [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        CHILD_DOCS [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)

CHILD_DOCS:
        CLINICAL_DATA [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        DOC_DESCRIPTION [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        DOC_STORE [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
        DOC_ATTRIBUTES [DOC_ID->DOC_ID]  (NAME[1],DATA[0],0)
```

B.9 tag_results.txt

```
DEMOGRAPHICS: 33 fields out of 320 identified.
   Table PAT_TEST_HISTV has 5 fields with the relevancy factor of 20.
      RSLT_UNIT_TXT[ZIP]
      TEST_PRTY_CD[ADDR]
      PARENT_ID[ZIP]
      SUPERGRP_NUM[ZIP]
      PAT_TEST_ID[ZIP]
   Table PERSNL_PUBLIC has 4 fields with the relevancy factor of 50.
      PERSNL_ID[ZIP]
      FIRST_NAME[FNAME]
      AUTH_PERSNL_ID[ZIP]
      LAST_NAME[LNAME]
   Table PAT_DEMOGRAPH has 6 fields with the relevancy factor of 83.
      STREET_ADDR[ADDR]
      CITY_NAME[LNAME]
      STATE_CD[ADDR]
      FIRST_NAME[FNAME]
      LAST_NAME[LNAME]
      ZIP_CD[ZIP]
   Table PHARMACY_TABLE has 4 fields with the relevancy factor of 25.
      FORMULARY_CODE_1[ZIP]
      FORMULARY_CODE_2[ZIP]
      ETXG_SEQUENCE[ZIP]
      MEDICATION_NAME[ADDR]
   After proximity to 'primary key' (PAT_DEMOGRAPH):
        ADDR: 9,STREET_ADDR
        ZIP: 13,ZIP_CD
        LNAME: 2,LAST_NAME
        FNAME: 3,FIRST_NAME
MEDICATIONS & ALLERGIES: 17 fields out of 320 identified.
        0: located in PHARMACY_TABLE.MEDICATION_NAME with the score of 60.
        1: located in PAT_DEMOGRAPH.ALLERGY_TXT with the score of 28.
   Medications identified as PHARMACY_TABLE.MEDICATION_NAME
        dos_score: 100.
        dosage: PATIENT_RECORD_NUMBER.
        dat_score: 5.
        date: DATE_OF_SERVICE.
PROBLEMS: identified as PROBLEMS.PROBLEM_NAME with the score of 42
NOTES: identified as DOC_STORE.CONTENT with the score of 1
```

Bibliography

[1] S. I. Allen, G. O. Barnett, and P. A. Castleman. Use of a time-shared general purpose file handling system in hospital res. *Proc.IEEE*, 54, 1966.

[2] G. O. Barnett, R. D. Zielstorff, J. McLatchey, et al. Costar: A comprehensive medical information system for ambulatory care. In *Symposium on Computer Applications in Medical Care*, volume 6, pages 8–18. IEEE Press, 1982.

[3] S. M. Bobrowski. *Oracle 7 & Client/Server Computing.* Sybex, San Francisco, CA, 1994.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[5] C. J. Date. *A Guide to the SQL Standard: A User's Guide to the Standard Relational Language SQL.* Addison-Wesley Publishing Company, Reading, MA, 1987.

[6] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley Publishing Company, Reading, MA, 6th edition, 1995.

[7] R. D. Dick and E. B. Steen. *The Computer-based Patient Record: An Essential Technology for Health Care.* National Academy Press, 1991.

[8] D. H. Fisher, M. J. Pazzani, and P. Langley, editors. *Concept Formation: Knowledge and Experience in Unsupervised Learning.* Morgan Kaufmann Publishers, 1991.

[9] B. J. Hettinger and R. P. Brazile. Health level seven (HL7): standard for healthcare electronic data transmissions. *Computaion in Nursing*, pages 13–16, 1994.

[10] G. Koch and K. Loney. *ORACLE: The Complete Reference.* Osborne McGraw-Hill, Berkeley, CA, third edition, 1995.

[11] I. S. Kohane. Getting the data in: three year experience with a pediatric electronic medical record system. In *Symposium on Computer Applications in Medical Care*, volume 18, pages 457–61. Hanley & Belfus, 1994.

[12] C. Levy and C. Beauchamp. Abstraction of the relational model from a Department of Veterans Affairs DHCP database: Bridging theory and working practice. In *Journal of the American Medical Informatics Association*, volume 20. Hanley & Belfus, 1996.

[13] C. Levy, C. Beauchamp, and J. E. Hammond. A managed care workstation for support of ambulatory care in veterans health administration medical centers. *Journal of Medical Systems*, 19(5):387–396, 1995.

[14] D. A. B. Lindberg, B. L. Humphreys, and A. T. McCray. The unified medical language system. *Methods of Information in Medicine*, 32(4):281–291, 1993.

[15] A. T. McCray and A. Razi. The UMLS knowledge source server. *MEDINFO*, pages 144–147, 1995.

[16] C. J. McDonald, L. Blevins, W. M. Tierney, and D. K. Martin. The Regenstrief medical records. *M.D. Computing*, 5(5):34–47, 1988.

[17] D. Meier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[18] National Library of Medicine. *UMLS Knowledge Sources*, seventh experimental edition, 1996.

[19] Oracle Corp. *Oracle Database Designer.* http://www.oracle.com/products/tools/dbdes/html/.

[20] P. Patel and K. Moss. *Java Database Programming with JDBC.* Coriolis Group Books, Scottsdale, AZ, 1996.

[21] J. Pearl and R. Dechter. Learning structure from data: A survey. Technical Report CSD-910048, University of California, Los Angeles, July 1991.

[22] G. Piatetsky-Shapiro and W. J. Frawley, editors. *Knowledge Discovery in Databases.* MIT Press, 1991.

[23] T. A. Pryor, R. M. Gardner, P. D. Clayton, and H. R. Warner. The Help system. In *Symposium on Computer Applications in Medical Care*, volume 6, pages 19–27. IEEE Press, 1982.

[24] E. H. Shortliffe and L. E. Perreault, editors. *Medical Informatics: Computer Applications in Health Care.* Addison-Wesley Publishing Company, Reading, MA, 1990.

[25] W. W. Stead and W. E. Hammond. Computer-based medical records: The centerpiece of TMR. *M.D. Computing*, 5(5):48–62, 1988.

[26] W. R. Stevens. *Advanced Programming in the UNIX Environment.* Addison-Wesley Publishing Company, Reading, MA, ninth edition, 1992.

[27] L. Sweeney. Replacing personally-identifying information in medical records, the Scrub system. In *Journal of the American Medical Informatics Association*, volume 20, pages 333–337. Hanley & Belfus, 1996.

[28] L. Sweeney. Scout: Learning concepts from a database. Aritificial Intelligence Laboratory, MIT, Working Paper, 1996.

[29] L. G. Valiant. A theory of learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[30] F.J. van Wingerde, J. Schindler, P. Kilbridge, et al. Using HL7 and the World-Wide Web for unifying patient data from remote databases. In *Journal of the*

American Medical Informatics Association, volume 20, pages 141–145. Hanley & Belfus, 1996.

[31] L. Wall, T. Christiansen, and R. L. Schwarz. *Programming Perl.* O'Reilly & Associates, second edition, 1996.

[32] Q. E. Whiting-O'Keefe, A. Witting, and J. Henke. The STOR clinical information system. *M.D. Computing*, 5(5):8–21, 1988.