Chapter 1

Database Heterogeneity and Decision Support

The practice of clinical medicine demands that physicians perform complex information-processing tasks. To care for a single patient adequately, physicians must assimilate a large number of patient parameters, analyze those parameters in the context of their medical knowledge, build a mental model of the patient's condition to direct further diagnostic and treatment steps, and update this model continuously as the patient parameters change. Given that a typical physician cares for numerous patients concurrently, the information-processing demands of clinical medicine quickly approach the limits of human cognitive abilities.

Recognizing this problem, researchers in the field of medical informatics have devoted significant effort to the development of computational tools that can assist in or assume certain of the clinical information-processing tasks and thereby extend the abilities of physicians. One variety of these tools, *clinical decision-support (CDS) systems*, assists physicians by simulating the expertise of clinical consultants. Typically, these systems collect the clinical data associated with a single patient, process these data in the context of a large store of medical knowledge, and generate a recommendation, reminder, or critique intended to enhance the patient's clinical care. Although CDS systems vary widely with respect to the clinical contexts in which they are useful, the types of advice they provide, and the technical methods they employ, all share a common feature: Just like a human clinical consultant, a CDS system must acquire the data relevant to a specific patient before it can generate any advice. In colloquial terms, the system must first "learn the facts of the case."

The increasing prevalence of electronic medical record systems has created the potential for CDS applications to learn the facts of a case without human intervention. This mode of data access is often advantageous because it relieves users from tedious data entry and allows applications to operate quickly and unobtrusively. However, the complexity and heterogeneity of electronic medical record systems impedes the widespread use of CDS applications that access clinical databases automatically. Because the schemas, vocabularies, and query languages of clinical databases vary significantly across healthcare facilities, one must customize the database interfaces of CDS applications at each facility where the applications are deployed. The goal of my research has been to simplify and to standardize the way in which CDS systems acquire the facts of a case despite the complexity and heterogeneity of the electronic databases that store those facts. By providing all CDS systems with a single model of clinical data and a principled way to use that model to access heterogeneous clinical databases, I hope to eliminate the need to customize CDS applications, thereby facilitating their widespread deployment.

In this dissertation, I describe a novel technology that *automatically translates database queries to a variety of equivalent formulations* so that a single query can retrieve data directly from a variety of clinical databases. The technology includes an abstract database schema and a high-level query language that allows the developers of CDS systems to formulate queries without knowing the complex and heterogeneous features of specific databases. The key element of the technology is a mapping language that allows experts at each database site to encode a formal mapping between the *abstract database schema* that is used by CDS-system developers and the *implemented database schema* that describes the local database. A query-translation algorithm applies the mapping to automatically translate any query that appears in a decision-support application to an equivalent query that is specific to the schema, clinical vocabulary, and query language of the local database. I assert that the formal properties of this query-translation methodology distinguish it from existing methods for customizing database queries and establish it as an important new technology for sharing decision-support applications in medicine and other domains.

1.1 Expert Systems and Patient Data

Although medical informatics researchers have paid relatively little attention to the acquisition of patient data by CDS systems, the way in which a decision-support system learns the facts of a case has significant implications for that system's acceptance as a standard clinical tool and for its portability to multiple clinical environments. A comparison of two prominent CDS systems illustrates this point.

1.1.1 A case study

The MYCIN system [Buchanan and Shortliffe 1984] is a clinical consultation tool designed to assist in the diagnosis and treatment of patients with microbial infections before definitive culture results are available. MYCIN acquires all patient data by directly prompting the user of the system to enter specific information. For example, Figure 1.1 displays a sample dialogue between the MYCIN interface and a physician user.

> What is the age of the patient?> **55**
> Does the patient have a fever?> **Y**
> What is the morphology of the organism?> **ROD** ...

Figure 1.1: A sample dialogue between the MYCIN system and a physician user. Each question is a query posed by the MYCIN interface, and each statement (in boldface) is a response entered by the physician.

MYCIN was designed to prompt for only those data that are contextually relevant and to avoid asking for the same information twice. Nevertheless, the system still requires that the physician user manually enter many facts of a case; often these facts are already recorded elsewhere, perhaps even in electronic form. Although formal evaluations proved MYCIN as proficient as infectious-disease specialists in generating differential diagnoses and formulating treatment plans [Buchanan and Shortliffe 1984], the system was never widely used in day-to-day practice. The data-entry burden that MYCIN places on users may be partially responsible for the system's lack of acceptance.

The HELP system [Kuperman, et al. 1991] is another clinical consultation tool; it is designed to identify and report aspects of a patient case that require attention, but that may be overlooked by caregivers. HELP and MYCIN differ in many ways, but one prominent distinction is that the HELP system acquires as many as possible of the patient data it needs from an online clinical database, rather than from the user. HELP collects patient data by issuing queries to which a clinical database can respond and by assigning the results of queries to internal variables that the HELP system can process. Figure 1.2 shows an excerpt from a HELP program that retrieves and subsequently processes the value of a patient parameter.

```
DATA:     recent_serum_sodium  = Query [ last(Serum_Sodium) ]

LOGIC:    If recent_serum_sodium > 155
            Then Alert_physician
```

Figure 1.2: Excerpt from a HELP system program. The first command instructs the system to retrieve the value of a patient parameter from the clinical database, and the second command instructs the system to compare the retrieved value to a reference value in order to reach a medically relevant conclusion.

In addition to eliminating unnecessary data reentry, automatic access to clinical data allows the HELP system to operate in the background, constantly assessing a patient's condition and level of care, and appearing only when a clinical recommendation is warranted. Although the HELP system has been in use at the Latter-day Saints (LDS) Hospital in Salt Lake City for over 20 years, and formal evaluations have proved it effective in improving care and reducing costs [Bradshaw 1988, Evans, et al. 1990], similar systems are in place today at only a handful of other institutions. One impediment to the dissemination of HELP technology is the difficulty of implementing a comprehensive clinical database. Perhaps a greater obstacle is the tremendous investment of resources required to develop, encode, and evaluate the complex clinical criteria that systems like HELP use to analyze patient data. This obstacle exists because the extensive set of clinical criteria implemented in the existing HELP system—despite being encoded in small, modular units and applying general medical knowledge—cannot be shared by any other institution without significant manual translation. This lack of portability results from strong assumptions encoded into the criteria regarding the implementation of the clinical database—assumptions currently required for HELP programs to access patient data automatically. These assumptions include the way the database is organized, the query language the database understands, and the terms used in the database to represent patient parameters. Because the assumptions are specific to the clinical database implementation at the LDS Hospital and are not likely to hold at other institutions, the encoded clinical criteria cannot be used elsewhere.

1.1.2 The data acquisition tradeoff

MYCIN and HELP are examples of technologies that, despite extensive development and proven clinical effectiveness, have not been deployed widely due (at least partially) to the way in which they acquire patient data. It is instructive to note that the data-acquisition weaknesses of each system are the strengths of its counterpart. MYCIN is less acceptable to physicians because it requires them to reenter data that are already recorded, a flaw overcome by the HELP system through its ability to retrieve data automatically from a clinical database. The HELP system cannot be used at most institutions, however, because, in order to automatically retrieve data, it must make restrictive assumptions about the way in which clinical databases are implemented, a limitation avoided by MYCIN through its reliance on the user for all data (which entails the less restrictive assumptions that the "database" speaks English and understands medical terminology). The two systems effectively illustrate the extremes of a tradeoff: automated data acquisition versus portability.

Although MYCIN and HELP have demonstrated some utility despite the limitations imposed by this trade-off, other kinds of CDS systems may be rendered completely infeasible. For example, several centralized agencies are currently developing standard protocols for directing cost-effective clinical care, which they intend to disseminate widely [Kelly and Swartwout 1990, VanAmringe and Shannon 1992]. Some investigators have suggested that decision-support systems could serve as a vehicle for the widespread distribution and unobtrusive application of these protocols [Bronzino, et al. 1989, van der Lei and Musen 1991]. Under such a model, standard protocols would be encoded as computer programs and downloaded from the publishing source to CDS systems at numerous provider sites. The CDS systems would subsequently execute the protocols based on local clinical data. Directed by the logic of a protocol, each CDS system would access patient data from the local clinical database and use it to determine the clinical context of a patient's case. Based on that context, the system would apply further protocol knowledge to generate advice regarding the diagnostic and therapeutic steps appropriate for the case. Without *both* the database-querying capabilities needed to establish clinical contexts automatically *and* the portability needed to implement the system at a variety of provider sites, the essential objectives of such a system would be compromised.

To make feasible future systems such as the protocol-dissemination tool, as well as to facilitate the wider use of existing systems such as MYCIN and HELP, the trade-off between data acquisition and portability must be eliminated. Developers of decision-support systems can overcome this trade-off only by designing their systems to query clinical databases in a manner that is independent of the databases' specific implementation features. Until now, accessing clinical databases in this manner has been impossible given existing technology. The fictitious scenario in the following section illustrates some of the difficulties.

1.2 Query Models

Sam is a clinical protocol designer at a large research institute near Washington, D.C. He is currently working on a protocol for diagnosing and managing patients with infectious gastroenteritis. The protocol consists of a flowchart that suggests diagnostic tests based on the nature and duration of symptoms, and that recommends therapeutic interventions based on the results of these diagnostic tests. Because all clinical protocols are distributed and applied via decision-support systems, Sam is encoding his protocol as a computer program, using a standard programming language for representing medical decision logic. The language has been adopted universally, so that the programs he writes can be run at any provider site in the United States, provided that the site stores patient data electronically.

One part of Sam's protocol checks whether a patient is taking antibiotics, because antibiotic use can predispose to gastrointestinal infections. If the patient is taking antibiotics, the protocol will recommend their discontinuation. Sam has almost completed this portion of the protocol and is looking forward to going home for the evening. All that remains is the formulation of a database query to inform the protocol whether the patient is, in fact, taking any antibiotics. Sam pauses for a moment to gather his thoughts before hitting the next keystroke.

1.2.1 Query-model components

What does Sam need to know to formulate this simple query? The answers to this question lie at the heart of the trade-off between data acquisition and portability. To formulate an effective query, Sam must know:

1. The abstract *model* of data representation that applies to the clinical database (for example, can the data be thought of as represented by unstructured text files, by uniform tables, or by a tree structure?)

2. The *schema* of the specific data that are represented in the clinical database (for example, are the names of patients and the names of the medications taken by those patients represented in a single file, which the query can directly access, or are they represented in different files, which the query must compare?)

3. The *language* for specifying queries that can be processed by the clinical database (for example, does the database require low-level commands that instruct it precisely where to find the names of all drugs associated with a patient name, or can it process high-level commands that only describe which patients and drugs to retrieve?)

4. The *format* of the data that are represented in the clinical database (for example, are the names of drugs represented as the full trade names used by clinicians, as the abbreviated names used by the hospital formulary, or as the numerical codes used by a national drug-coding system?)

Together, these four components constitute the *query model* that is used to formulate data-retrieval requests. Informally, a query model is the model of data storage and information retrieval that is in the mind of a database programmer when he encodes the conceptual notion of an information request into the executable commands of a formal query language. Figure 1.3 illustrates the components of a query model and how the components fit into a conceptual whole.

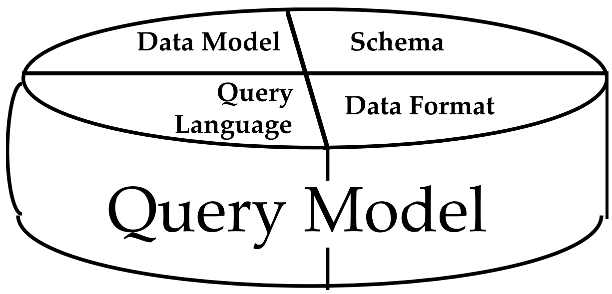


Figure 1.3: The components of a query model: an abstract data model for conceptualizing the data, a schema for organizing the data, a format for representing the data, and a query language for retrieving the data.

1.2.2 The query-model mismatch

Sam cannot formulate the query he has in mind without making a commitment to a specific query model, that is, without specifying each component of the query model he intends to use. However, the provider sites that will receive Sam's protocol have implemented their clinical databases using a variety of query models, which are likely to differ from Sam's query model and from one another. Although they all represent similar information, the clinical databases at provider sites implement different data models, assume different database schemas, process different query languages, and employ different data formats. Therefore, there necessarily will be a query-model *mismatch* at some sites when Sam's protocol is downloaded, and the mismatch may prevent Sam's protocol from accessing patient data correctly and from functioning as intended.

Until now, there has been only one way to bridge this mismatch and thus to confer automated data acquisition and portability upon Sam's protocol: For almost every provider site, the antibiotic-medication query had to be translated manually from Sam's query model to the query model of the local implemented database. Performing this translation entailed understanding the differences between the assumed and the implemented query models with respect to each of the four query model components, and translating all portions of the query such that the meaning of the whole was preserved. Depending on the degree of query-model mismatch, this process required careful, skilled, and extensive labor. Assuming N different query models, the translation needed to be performed N times by Sam prior to disseminating his protocol, or once at each of N sites when Sam's protocol was downloaded. In either case, the total workload was the same.

1.2.3 Bridging query models

Fortunately for Sam, he is using a new technology that allows him to encode database queries so that they can be translated *automatically* to a variety of query models. The technology available to Sam consists of three components:

1. An *abstract query model* that allows clinical data requests to be formulated in a general way, independent of the implemented query model at any provider site

2. A set of *mappings* that relates each element of a local implemented query model to its counterpart in the standard abstract query model

3. A *translating compiler* that uses the mappings to translate queries formulated in the abstract query model to equivalent queries in a local implemented query model

The abstract query model comprises the same four components as shown in Figure 1.3. The mappings encode the correspondence between the abstract query model and a specific implemented query model. These mappings are encoded such that a translating compiler can use them to translate any abstract query to an equivalent implementation-specific query. Figure 1.4 illustrates how a translating compiler uses the set of mappings to allow Sam's protocol to retrieve patient data from an arbitrary clinical database implementation.

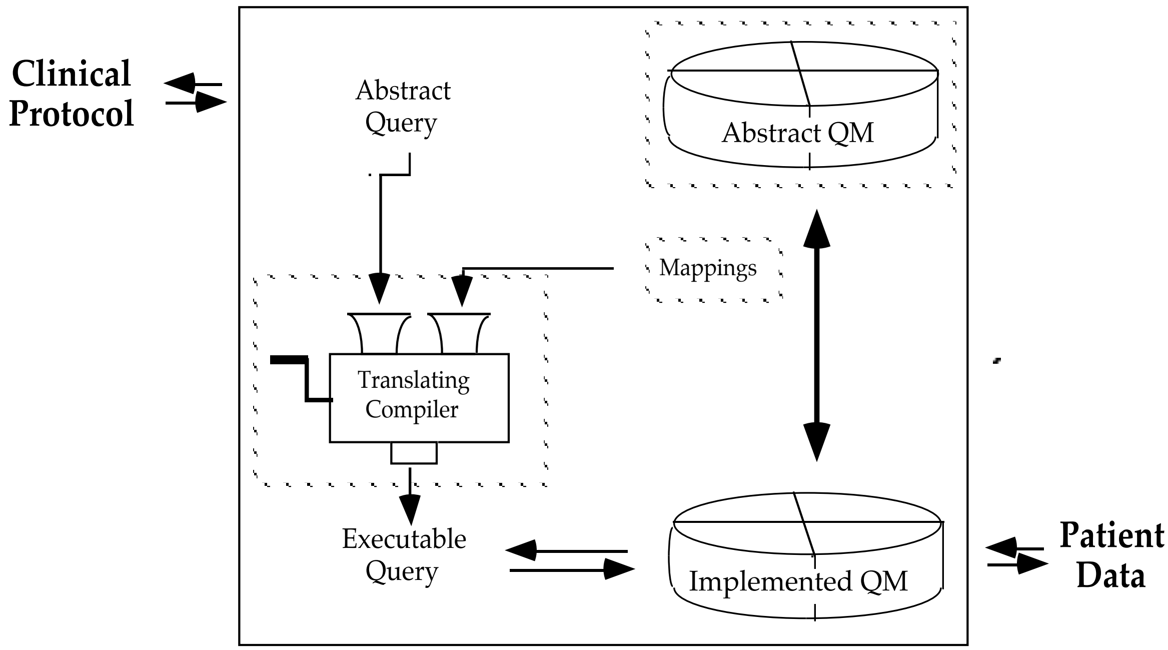


Figure 1.4: Schematic of the query-translation technology. A translating compiler uses encoded mappings between query models to translate any query formulated in the standard abstract query model (QM) to an equivalent query that can be executed by an implemented QM. The hatched boxes denote the three primary components of the query-translation technology.

Because Sam is using the abstract query model to encode his query, he does not need to make commitments to any implemented query model—commitments that will constrain the environments in which the query can be executed. The only commitments made in the abstract query model concern features of databases that are common to all the databases with which clinical applications may interface. For example, the model safely assumes that the concepts *patients*, *medications*, and *prescriptions* are represented in each clinical database (in some, probably implementation-specific, way). The model also assumes that a query to retrieve all the medications prescribed for a particular patient can be formulated in the query language of each clinical database (again, in some implementation-specific way). Sam is not reluctant to use a query model that makes such general assumptions about clinical databases, because he knows that each clinical database can satisfy them. In addition, Sam knows that there is a way for his abstract query to be translated at each client site into an equivalent query that is consistent with the implemented database there, and that returns the requested results to his protocol. The translation is possible because personnel at each site have specified and represented mappings between the query model implemented at their site and the standard query model that Sam's protocol understands. A translating compiler uses these machine-readable mappings to translate his queries automatically into equivalent queries that can be executed directly by the local clinical database. Sam does not need to worry about how the translation of his specific query is performed when he writes the protocol, and the administrators of each local database do not need to worry about how the translation is performed when they download the protocol. In fact, as long as each provider site keeps its mappings up to date, Sam can write and the site can download an arbitrary number of protocols with no additional overhead. Comfortable in this knowledge, Sam quickly encodes the antibiotic query and leaves for home.

1.3 Addressing the Problem: The TransFER Approach

The scenario described in Section 1.2.3 depends on the existence of two critical technologies:

1. A standard abstract query model that is sufficiently general to subsume implemented query models with respect to each of the query-model components, and that is sufficiently rich to include most of the modeling concepts, medical objects, and query operations needed by clinical decision-support systems

2. A model and a mechanism for translating automatically queries formulated with respect to the standard query model into queries consistent with an arbitrary clinical database implementation, such that the only inputs to the translation process are the query itself and a representation of the mappings between the two query models

The first technology allows clinical expert systems to specify the database queries that they require without limiting the database implementations that they can access. The second technology allows CDS systems to execute the database queries within them across environments with disparate database implementations by performing automated query translation. I define the *standard query model framework* as the research paradigm concerned with specifying a standard query model and with developing techniques to automatically translate queries based on this model. The specific goals of this framework are to develop more general and expressive abstract query models and more powerful query-translation methods. As these goals are realized, shared decision-support systems will be able to retrieve a wider array of patient data automatically from a wider variety of clinical databases, and the tradeoff between data acquisition and portability will be gradually eliminated. Figure 1.5 shows the logical conclusion of this evolution: a general technology that allows CDS applications to access patient-specific data in a uniform way from disparately implemented databases.

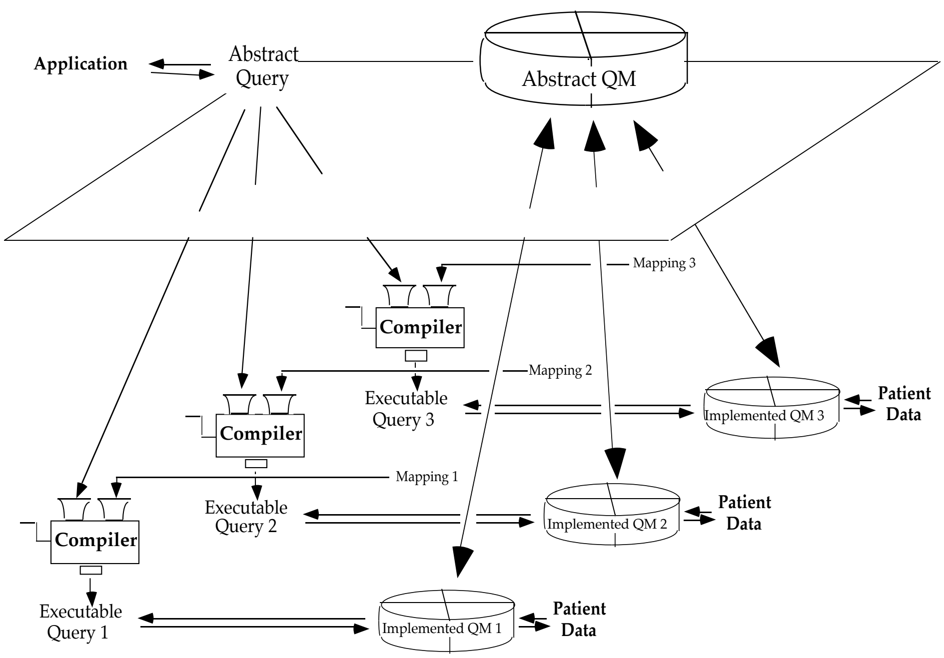


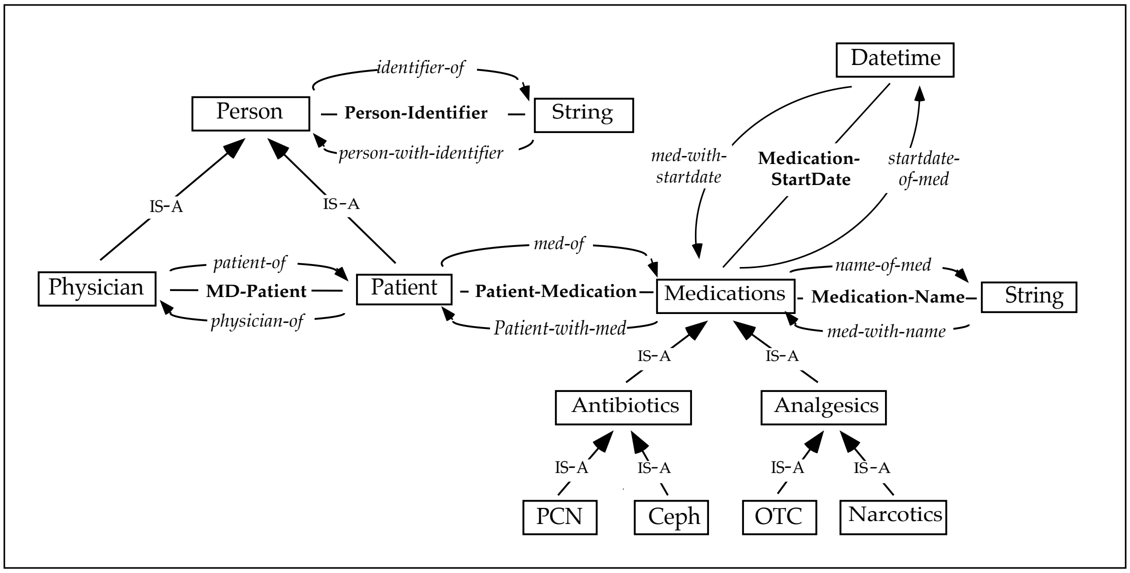
Figure 1.5: A model for supporting automated data access and portability. Application developers formulate queries using a common abstract query model. At each provider site, mappings between the local implemented query model and the abstract query model are entered into a translating compiler. The compiler converts the abstract queries appearing in the application program to executable queries that can be processed by the local clinical databases. Note that each site uses a copy of the same translating compiler.

I have developed an instantiation of the standard query model framework, called **the TransFER methodology**. TransFER provides access to heterogeneous databases implemented in the *relational* model by automatically translating queries formulated with respect to a standard query model to equivalent queries formulated in the relational query language SQL. Although the relational model is not ubiquitous among existing clinical databases, I have constrained the problem to relational databases because (1) the physical data independence provided by the relational model has steadily increased the use of this model in commercial database management systems, and (2) the well-studied mathematical properties of the relational model support a formalization of the query-translation process that is essential to its correctness and efficiency. If the TransFER methodology proves successful, I contend that nonrelational database sites will be no worse off than they are currently with respect to sharing decision-support applications, whereas the significant and increasing number of clinical database sites that have relational interfaces will benefit substantially from the automated query translation that TransFER provides.

The TransFER methodology consists of four general components:

1. A **semantic data model**, called FER (Functional Entity–Relationship model), supports the modeling of clinical data in an abstract fashion that can subsume many site-specific database implementations. The formal constructs of the FER model are entity types, relationship types, relationship functions, and IS-A connections. Figure 1.6 shows a representative FER schema for a small subset of the clinical domain.

2. A **high-level query language**, called ReFER, corresponds to the FER data model and allows developers of CDS applications to specify queries that are independent of any site-specific database implementation. A sample ReFER query appears in Figure 1.7. Note that this query is formulated with respect to the objects of the FER schema in Figure 1.6.


Datetime
Person
Person-Identifier
String
identifier-of
person-with-identifier
med-with-
startdate
Medication-
StartDate
startdate-
of-med
IS-A
IS-A
Physician
patient-of
MD-Patient
physician-of
Patient
med-of
Patient-Medication
Patient-with-med
Medications
name-of-med
Medication-Name
med-with-name
String
IS-A
IS-A
Antibiotics
Analgesics
IS-A
IS-A
IS-A
IS-A
PCN
Ceph
OTC
Narcotics

the correspondence between the abstract FER schema and a specific relational implementation of it. For example, if each entity and relationship type happens to be modeled as a separate table in the relational database, then the mapping view for each entity and relationship type is simply the corresponding table itself:

> view(Medications) => table(`PRESCRIPTIONS`)
> view(Person-Identifier) => table(`MEDICAL_RECORD_NUMBERS`)
> and so forth.

If there is not a one-to-one correspondence, however, the mapping expressions must combine or decompose tables to create views that accurately represent the intended semantics of the FER entity and relationship types. For example, if a database contains separate tables for inpatient and outpatient prescriptions, the mapping view for the entity type Medications (which represents both inpatient and outpatient prescriptions) must specify a *union* of the inpatient and outpatient tables to create a set that contains all of the prescribed medications represented in that database:

> view(Medications) => table(`INPATIENT_PRESCRIPTIONS`)
> UNION
> table(`OUTPATIENT_PRESCRIPTIONS`)

The effect of specifying a union in the view is that any ReFER query that references the entity type Medications will be translated to a relational query that searches both the `INPATIENT_PRESCRIPTIONS` and `OUTPATIENT_PRESCRIPTIONS` tables.

4. A **query-translation module** performs the translation of ReFER queries by combining in a principled way the ERA mapping views that correspond to the FER objects appearing in each query. For example, translation of the query in Fig. 1.7 to a site-specific relational query entails combining the mapping views for Patient, Medications, Person-Identifier, Medication-Name, and Medication-StartDate as specified for a particular site. If the ERA mappings are correctly specified, *the resulting relational expression is semantically equivalent to the input ReFER*

query. The translation module subsequently optimizes and translates the expression into a site-specific database program consisting of one or more SQL statements. Figure 1.8 shows the automatically generated output of the TransFER query translator for the query in Figure 1.7, based on a relational database schema that was used for testing purposes.

```
Begin DB Program {

SELECT  ENTITY_TYPE_NAMES.ent_type_name
FROM    Therapy T1, Patient, Therapy T2, MEDICATION_CODES MC1,
        FER_TYPE_HIERARCHY FTH1, ENTITY_TYPE_NAMES, Therapy T3,
        MEDICATION_CODES MC2, FER_TYPE_HIERARCHY FTH2
WHERE   T1.start_datetime > "1995/10/28 0800" AND
        Patient.mr_number = "MRN-188640" AND
        (T1.mr_number = Patient.mr_number AND
         T1.mr_number = T3.mr_number AND
         T1.med_code = T3.med_code AND
         convert(varchar(50),T1.start_datetime) =
               convert(varchar(50),T3.start_datetime)) AND
        (T3.mr_number = T2.mr_number AND
         T3.med_code = T2.med_code AND
         convert(varchar(50),T3.start_datetime) =
               convert(varchar(50),T2.start_datetime)) AND
        (ENTITY_TYPE_NAMES.ent_type = FTH1.ent_type) AND
        (FTH1.desc_ent_type = MC1.ent_type) AND
        (MC1.med_code = T2.med_code) AND
        (FTH2.ent_type = "Antibiotics") AND
           (FTH2.desc_ent_type = MC2.ent_type) AND
        (MC2.med_code = T3.med_code))

} End DB Progam
```

Figure 1.8: Output of TransFER query translator. The query shown in Figure 1.7 is translated to this database program based on mappings specific to a relational database. The database management system of the relational database can directly execute the SQL query produced by the translation.

To summarize, sharing applications using the TransFER model entails the following steps:

1. A global reference schema of clinical data is specified using the FER semantic data model. The schema represents the medical concepts and the associations among concepts typically stored in clinical information systems.

2. Based on the FER reference schema, local database administrators use the ERA mapping language to specify mappings between each entity

type and relationship type of the global FER schema and the relational tables and fields of their local database schemas.

3. Developers of CDS applications (for example, the authors of clinical alert rules) formulate all data requests as ReFER queries against the FER reference schema.

4. When CDS applications are distributed to local database sites, the TransFER compiler automatically applies the local ERA mappings to translate the ReFER queries into site-specific SQL queries. These queries can retrieve the appropriate data locally when the applications are run.

In this dissertation, I formally present the elements of the TransFER methodology and I describe an evaluation of the methodology's practical utility. The evaluation is based on a prototype implementation of TransFER that automatically translates queries formulated in the ReFER query language to database programs that can by executed by heterogeneous relational databases. The results of the evaluation show that TransFER is an effective realization of the standard query model framework for databases implemented in the relational model. Specifically, the evaluation shows that

1. The FER data model and ReFER query language are sufficiently expressive to represent queries that typically appear in CDS applications

2. The ERA mapping language is sufficiently powerful to encode the mappings between FER schemas and heterogeneous relational database implementations

3. The TransFER compiler is sufficiently robust to translate ReFER queries into SQL queries that are site-specific and semantically equivalent.

1.4 Guide to the reader

In the remainder of this document, I describe in much more detail the research problem that I addressed, the query-translation methodology that I developed, and the practical evaluation that I performed.

In Chapter 2, I create a framework for my research by characterizing the types of applications that can benefit from a standard model of database access and by describing previous research related to the development of such a model. Chapter 3 introduces and formally defines the semantic data model and the high-level query language that I propose as the standard model of database access for shared applications. Chapter 4 describes the formal methods that I have developed to translate queries formulated with respect to the standard model of database access to equivalent queries that can retrieve data from a variety of legacy clinical databases. In Chapter 5, I present the results of an evaluation that I performed to assess the practical feasibility of TransFER as a model for sharing clinical decision-support applications across heterogeneous provider sites. Chapter 6 summarizes my work, discusses the implication of the evaluation, and proposes several areas of future research.