Appendix A

The ERA Mapping Language

This appendix specifies the extended relational algebra (ERA) mapping language that is used to encode the site-specific portion of a TransFER attribute grammar. Section A.1 contains the context-free grammar of the mapping language in standard Backus-Naur form (BNF) [Sammet 1969, Chap. 2, Sec. 6]. A sample mapping between a specific FER schema and relational database schema appears in Section A.2. Section A.3 discusses the general strengths and limitations of the mapping language with respect to resolving heterogeneity among relational database implementations.

A.1 Context-free grammar of the ERA mapping language

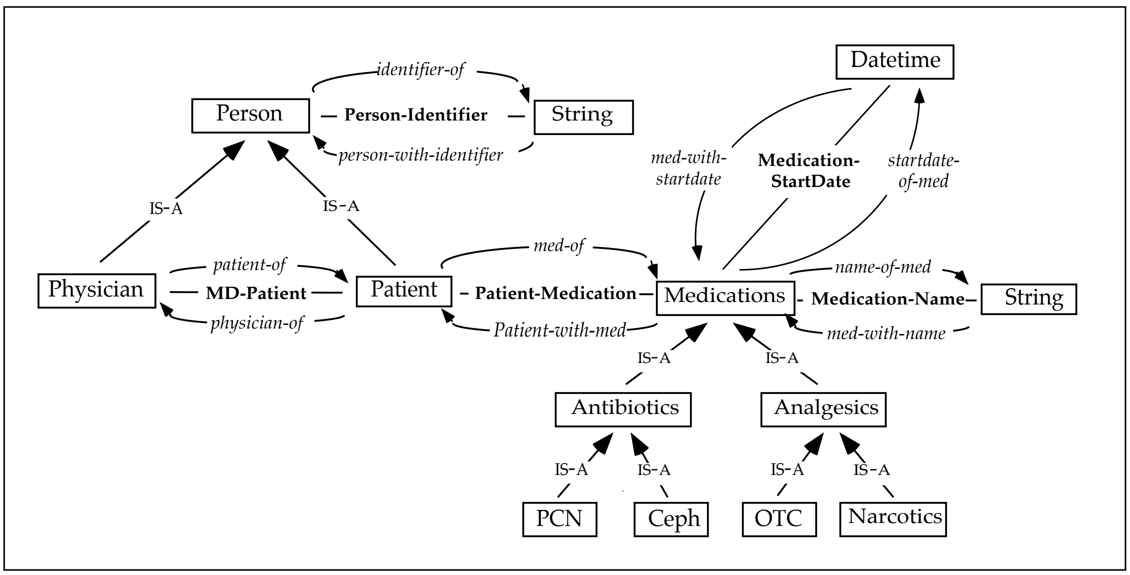
```
<mapping> ::= <defn-list>

<defn-list> ::= <defn>,<defn-list>  |
                <null>

<defn> ::= <entity-defn>  |
           <relship-defn> |
           <virt-defn>    |
           <built-in-entity-defn>

<entity-defn> ::= defEntity(name(<id-atom>),
                            era(<table-defn>),
                            id(<id-defn>)).

<relship-defn> ::= defRelship(name(<id-atom>),
                              era(<table-defn>),
                              id(<id-defn>),
                              fids([<fid-defns>])).
```

```
<virt-defn> ::= defVirtual(name(<id-atom>),   %define virtual relation
                           attrs([<virt-attr-defn>,
                                  <virt-attr-defn>])).

<built-in-entity-defn> ::= defBuiltInEntity(name(<id-atom>)).

<table-defn> ::= rel(<id-atom>)                                |
                 vrel(<id-atom>)                               |
                 rename([<rename-list>],<table-defn>)          |
                 extend([<extend-list>],<table-defn>)          |
                 proj([<id-atom-list>],<table-defn>)           |
                 sel([<sel-pred>],<table-defn>)                |
                 join([<join-pred-list>],<table-defn>,<table-defn>) |
                 union(<table-defn>,<table-defn>)              |
                 aggregate(aggparms(<parameters>),<table-defn>)

<rename-list> ::= <rename-item>                   |
                  <rename-item>,<rename-list>

<rename-item> ::= <id-atom> => <id-atom>

<extend-list> ::= <extend-item>                   |
                  <extend-item>,<extend-list>

<extend-item> ::= <id-atom> == <factor>

<fxn-designation> ::= sql | ext  %fxn is either SQL built-in or external

<factor-list>   ::= <factor>                  |
                    <factor>,<factor-list>

<id-atom-list> ::= <id-atom>                   |
                   <id-atom>,<id-atom-list>

<factor> ::= <id-atom>             |
             const(<const-atom>)   |


<const-atom> ::= <quoted-string-atom>  |  e.g. '"PT"'
                 <integer-atom>        |  e.g. '7'
                 <float-atom>          |  e.g. '6.4'
                 <date-string-atom>       e.g. '"12/02/63"'
```

```
<sel-pred> ::= and(<sel-pred-list>)    |
               or(<sel-pred-list>)     |
               not(<sel-pred>)         |
               null(<id-atom>)         |
               not_null(<id-atom>)     |
               <factor> <comp-operator> <factor>

<sel-pred-list> ::= <sel-pred>                  |
                    <sel-pred>,<sel-pred-list>

<comp-operator> ::= <> | = | > | < | >= | <=

<join-pred-list> ::= <join-pred>,<join-pred-list> |
                     <null>

<join-pred> ::= <factor> = <factor>

<id-defn> ::= [id-atom-list]

<parameters> ::= [<arg-list>],[<group-list>],
                 <agg-fxn-designation>,[<new-attr-list>]

<arg-list> ::= <id-atom-list>
<group-list> ::= <id-atom-list>
<new-attr-list> ::= <id-atom-list>

<agg-fxn-designation> ::= aggsql(<id-atom>) |  %sql built-in aggregator
                          aggext(<id-atom>)    %external aggregator

<fid-defns> ::= <fid-defn>,<fid-defns>  |
                <null>

<fid-defn> ::= fid(<id-atom>,<factor-list>)

<virt-attr-defn> ::= attrdef([<id-atom-list>],[<attr-defn>] )

<attr-defn> ::= const(<const-atom>)
                fxn(<fxn-designation>(<id-atom>),[<factor-list>]) |
                undefinedFxn([<id-atom>,  %name of virtual relationship
                             [<id-atom-list>],    %source entity type
                             [<id-atom-list>] ])  %target entity type

<id-atom> ::=  any prolog atom (see language reference)
```

A.2 A sample ERA mapping

This section contains a small but complete ERA mapping between an FER schema and a relational database schema. Although the sample mapping does not illustrate all of the features of the mapping language, it provides a representative example of how one encodes mappings between an FER schema and a relational database schema. The FER schema appears in Figure A.1 and the corresponding relational database schema appears in Section A.2.1. The ERA mapping between the FER and relational schemas appears in Section A.2.2.



between the codes used in the relational database to denote medications and the entity-type names used in the FER schema to denote the same medications.

```
create table Patient (
        mr_number                char(11)        not null,
        last_name                varchar(15)     null,
        first_name               varchar(15)     null,
        birthdate                varchar(15)     null)

create table Physician (
        dea_number               char(6)         not null,
        last_name                varchar(15)     null,
        first_name               varchar(15)     null,
        birthdate                varchar(15)     null,
        specialty                varchar(20)     null )

create table Admission (
        mr_number                char(11)        not null,
        dea_number               char(6)         not null,
        admit_datetime           varchar(15)     not null,
        location                 varchar(10)     null,
        discharge_datetime       varchar(15)     null )

create table Therapy (
        mr_number                char(11)        not null,
        med_code                 char(3)         not null,
        start_datetime           datetime        not null,
        end_datetime             datetime        null,
        quantity                 varchar(6)      null)

/*****************************************************************/
/* The following tables are not part of the original database
   schema.  They were added to the schema later (by the schema
   mapper) to support the mapping of the database to the FER schema */
/*****************************************************************/

create table MEDICATION_CODES (
        med_code     varchar(50)      not null,    %site-specific code
        ent_type     varchar(50)      not null)    %Corresponding FER
                                                    entity type

Example contents:       med_code          ent_type
                        125               "Ceph"
                        104               "PCN"
                         70               "Narcotics"
```

```
/** ADDITIONAL TABLES FOR TYPE HIERARCHY AND VOCABULARY MAPPING  **/

create table FER_TYPE_HIERARCHY (
        ent_type          varchar(50)   not null,  %FER entity type
        desc_ent_type     varchar(50)   not null)  %descendant type
                                                    in type hierarchy

Example contents:       ent type          desc ent type
                        "Antibiotics"     "Ceph"
                        "Antibiotics"     "PCN"
                        "Analgesics"      "Narcotics"
                        "Analgesics"      "OTC"


create table ENTITY_TYPE_NAMES (
        ent_type             varchar(50)     not null,
        ent_type_name        varchar(50)     not null)

Example contents:       ent type          ent type name
                        "Ceph"            "Cephalosporin"
                        "PCN"             "Penicillin"
                        "OTC"             "over-the-counter analgesic"
```

A.2.2 The ERA mapping expressions

The complete mapping between the FER schema in Figure A.1 and the relational schema in Section A.2.1 appears in this section. The mapping specifications conform to the context-free grammar defined in Section A.1. Note that each entity type and each relationship type in the FER schema of Figure A.1 has a corresponding mapping specification. A mapping compiler can process these specifications into a form that is integrated with the site-independent semantic functions of the TransFER attribute grammar (described in Appendix B). Together, the site-specific mappings and the site-independent mappings constitute a complete TransFER attribute grammar.

```
/* Built-in entity types; they require no mapping specifications */

defBuiltInEntity(name('Entity')).    % the "root" entity type
defBuiltInEntity(name('String')).
defBuiltInEntity(name('Datetime')).
```

```
/* Demographics */

defEntity(name('Person'),
          era(union(proj(['Id','Type'],
                         rename(['mr_number' => 'Id'],
                                extend(['Type' == const('"PT"')],
                                       rel('Patient')))),
                    proj(['Id','Type'],
                         rename(['dea_number' => 'Id'],
                                extend(['Type' == const('"MD"')],
                                       rel('Physician')))))),
          id(['Id','Type'])).

defRelship(name('Person-Identifier'),
           era(join(['Id' = 'IdPart',
                     'Type' = 'TypePart'],
                  vrel('Identifier-IdParts'),
                  union(proj(['Id','Type'],
                          rename(['mr_number' => 'Id'],
                              extend(['Type' == const('"PT"')],
                                     rel('Patient')))),
                        proj(['Id','Type'],
                          rename(['dea_number' => 'Id'],
                              extend(['Type' == const('"MD"')],
                                     rel('Physician')))))),
          id(['Identifier']),
          fids( [fid('Person',['Id','Type']),
                 fid('String',['Identifier'])] )).



defEntity(name('Physician'),
          era(rename(['dea_number' => 'Id'],
                     extend(['Type' == const('"MD"')],
                            rel('Physician')))),
          id(['Id','Type'])).



defEntity(name('Patient'),
          era(rename(['mr_number' => 'Id'],
                     extend(['Type' == const('"PT"')],
                            rel('Patient')))),
          id(['Id','Type'])).
```

```
defRelship(name('MD-Patient'),
           era(extend(['MD_Type' == const('"MD"'),
                        'PT_Type' == const('"PT"'), ],
                       rel('Admission'))),
           id(['dea_number','MD_Type','mr_number',
               'PT_Type','admit_datetime']),
           fids( [fid('Physician',['dea_number','MD_Type'])
                  fid('Patient',['mr_number','PT_Type']),    ] ))..

/* Medications */
defEntity(name('Medications'),
          era(rel('Therapy')),
          id( ['mr_number','med_code','start_datetime'] )).

defEntity(name('Antibiotics'),
          era(sel( ['anc_ent_type' = const('"Antibiotics"')],
                   join([ 'desc_ent_type' = 'ent_type' ],
                      rename(['ent_type' => 'anc_ent_type'],
                             rel('FER_TYPE_HIERARCHY')),
                      join([ 'med_code' = 'med_code' ],
                           rel('MEDICATION_CODES'),
                           rel('Therapy'))))),
          id( ['mr_number','med_code','start_datetime'] )).

defEntity(name('Analgesics'),
          era(sel( ['anc_ent_type' = const('"Analgesics"')],
                   join([ 'desc_ent_type' = 'ent_type' ],
                      rename(['ent_type' => 'anc_ent_type'],
                             rel('FER_TYPE_HIERARCHY')),
                      join([ 'med_code' = 'med_code' ],
                           rel('MEDICATION_CODES'),
                           rel('Therapy'))))),
          id( ['mr_number','med_code','start_datetime'] )).

defEntity(name('PCN'),
          era(sel( ['anc_ent_type' = const('"PCN"')],
                   join([ 'desc_ent_type' = 'ent_type' ],
                      rename(['ent_type' => 'anc_ent_type'],
                             rel('FER_TYPE_HIERARCHY')),
                      join([ 'med_code' = 'med_code' ],
                           rel('MEDICATION_CODES'),
                           rel('Therapy'))))),
          id( ['mr_number','med_code','start_datetime'] )).
```

```
defEntity(name('Ceph'),
          era(sel( ['anc_ent_type' = const('"Ceph"')],
                   join([ 'desc_ent_type' = 'ent_type' ],
                      rename(['ent_type' => 'anc_ent_type'],
                             rel('FER_TYPE_HIERARCHY')),
                      join([ 'med_code' = 'med_code' ],
                           rel('MEDICATION_CODES'),
                           rel('Therapy'))))),
          id( ['mr_number','med_code','start_datetime'] )).

defEntity(name('OTC'),
          era(sel( ['anc_ent_type' = const('"OTC"')],
                   join([ 'desc_ent_type' = 'ent_type' ],
                      rename(['ent_type' => 'anc_ent_type'],
                             rel('FER_TYPE_HIERARCHY')),
                      join([ 'med_code' = 'med_code' ],
                           rel('MEDICATION_CODES'),
                           rel('Therapy'))))),
          id( ['mr_number','med_code','start_datetime'] )).

defEntity(name('Narcotics'),
          era(sel( ['anc_ent_type' = const('"Narcotics"')],
                   join([ 'desc_ent_type' = 'ent_type' ],
                      rename(['ent_type' => 'anc_ent_type'],
                             rel('FER_TYPE_HIERARCHY')),
                      join([ 'med_code' = 'med_code' ],
                           rel('MEDICATION_CODES'),
                           rel('Therapy'))))),
          id( ['mr_number','med_code','start_datetime'] )).


defRelship(name('Medication-Name'),
           era(join([ 'ent_type2' = 'ent_type' ],
                  rename(['ent_type' => 'ent_type2'],
                        rel('ENTITY_TYPE_NAMES')),
                  join(['desc_ent_type' = 'leaf_ent_type'],
                     rel('FER_TYPE_HIERARCHY'),
                         join([ 'med_code' = 'med_code' ],
                              rename(['ent_type' => 'leaf_ent_type'],
                                     rel('MEDICATION_CODES')),
                              rel('Therapy'))))),
           id( ['mr_number','med_code',
                'start_datetime','ent_type_name'] ),
```

```
          fids( [ fid('Medications', ['mr_number', 'med_code',
                                     'start_datetime']),
                  fid('String', ['ent_type_name']) ] )).

defRelship(name('Patient-Medication'),
           era(extend(['Type' == const('"PT"')],
                  rel('Therapy'))),
           id( ['mr_number','med_code','start_datetime'] ),
           fids( [ fid('Patient', ['mr_number','Type']),
                   fid('Medications',['mr_number','med_code',
                                     'start_datetime']) ] )).

defRelship(name('Med-Start_Time'),
           era(rel('Therapy')),
           id( ['mr_number','med_code','start_datetime'] ),
           fids( [ fid('Medications', ['mr_number','med_code',
                                      'start_datetime'] ),
                   fid('Datetime',['start_datetime']) ] )).

defRelship(name('Med-Stop_Time'),
           era(rel('Therapy')),
           id(['mr_number','med_code','start_datetime','end_datetime']),
           fids( [ fid('Medications', ['mr_number','med_code',
                                      'start_datetime'] ),
                   fid('Datetime',['end_datetime']) ] )).

/* VIRTUAL RELATIONS */

defVirtual(name('Identifier-IdParts'),
           attrs( [ attrdef(['Identifier'],
                            [fxn(ext(concat_id),['Id','Type'])]),
                    attrdef(['Id','Type'],
                            [undefinedFxn(['Identifier-IdParts'],
                                          ['Id','Type'],
                                          ['Identifier']) ]) ] )).
```

Note: The virtual relation "Identifier-IdParts" applies the (one-directional) function "concat_id," which concatenates the pair of string values "Id" and "Type" into the single string "Identifier." The function is required to generate a single string that uniquely identifies each person, as is represented in the FER schema.

A.3 The Power of the ERA Mapping Language

The objective of the TransFER methodology is to allow applications to access heterogeneous relational databases through a uniform conceptual data model. The ERA mapping language, therefore, is designed to transform a wide variety of relational database implementations to a single canonical representation—the FER global schema. The mapping language derives power to accommodate widely varying relational implementations through (1) its use of extended relational algebra operators that can exploit the full capabilities of SQL and (2) its ability to apply arbitrarily complex external functions that execute in the host environment.

Several database researchers have characterized the forms of heterogeneity that may exist among relational databases containing similar information [Agarwal 1991, Deen, et al. 1987, Kim and Jungyun 1991]. These forms of heterogeneity may be generally classified into structural differences, naming differences, semantic differences, and content differences. In this section, I analyze how the ERA mapping language handles each form of heterogeneity. Although the analysis is informal and based largely on examples, it serves to characterize the capabilities and the limitations of the ERA mapping language.

A.3.1 Structural Differences

Structural differences entail different decompositions of semantically similar objects into the tables and fields of a relational schema. The flexibility of the relational model allows database designers to model the same semantic object in a variety of relational representations, depending on local performance requirements, local data constraints, and local normalization preferences [Batini, et al. 1992]. At least three kinds of structural differences may exist among relational databases that represent similar information: alternative table decompositions, varying data and meta-data representations, and differing structured and free-text encodings.

A.3.1.1 Alternative table decompositions

This form of structural heterogeneity has two subtypes: alternative *horizontal* and *vertical* table decompositions. Alternative horizontal decompositions entail different distributions of the same fields among one or more tables. Fields may be distributed across multiple tables to normalize a database or fields may be aggregated into a single table to improve performance [Batini, et al. 1992]. For example, Figure A.2 shows how the fields of the table `PATIENT` in Database B may be distributed into the tables `PATIENT` and `PHYSICIAN` in Database A to achieve third normal form. The ERA mapping language easily can convert from Database A to Database B by applying the JOIN operator to the tables `PATIENT` and `PHYSICIAN` (joining on the field `MD_SSN`), and the mapping language can convert from Database B to Database A by applying the PROJECT operator twice to the table `PATIENT`.

Database A

PATIENT

MRN	LName	DOB	MD_SSN
147564773	Harmonic	12-2-57	249633805
216426077	Flynn	3-25-62	249633805

PHYSICIAN

MD_SSN	MD_LName	MD_DOB
249633805	Mason	11-17-58

Database B

PATIENT

MRN	LName	DOB	MD_SSN	MD_LName	MD_DOB
147564773	Harmonic	12-2-57	249633805	Mason	11-17-58
216426077	Flynn	3-25-62	249633805	Mason	11-17-58

Database A

INPATIENTS

MRN	LName	DOB
147564773	Harmonic	12-2-57
216426077	Flynn	3-25-62

OUTPATIENTS

MRN	LName	DOB
598044361	Collins	2-14-61
815466387	Hart	10-8-29

Database B

PATIENTS

MRN	LName	DOB	Type
147564773	Harmonic	12-2-57	INPT
216426077	Flynn	3-25-62	INPT
598044361	Collins	2-14-61	OUTPT
815466387	Hart	10-8-29	OUTPT

Database A

SODIUM_TESTS

MRN	Date	Value
147564773	10-08-95 0800	137.2
216426077	11-14-95 2200	144.8

POTASSIUM_TESTS

MRN	Date	Value
147564773	10-08-95 0800	4.3
216426077	11-14-95 2200	3.8

Database B

LAB_TESTS

MRN	Date	Na_Value	K_Value
147564773	10-08-95 0800	137.2	4.3
216426077	11-14-95 2200	144.8	3.8

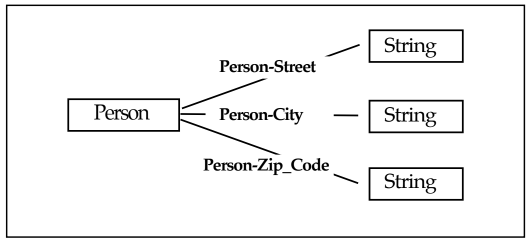
Database C

LAB_TESTS

MRN	Type	Date	Value
147564773	Sodium	10-08-95 0800	137.2
216426077	Sodium	11-14-95 2200	144.8
147564773	Potassium	10-08-95 0800	4.3
216426077	Potassium	11-14-95 2200	3.8

A.3.1.3 Structured versus free-text encodings

An entity or a relationship in an FER schema may be modeled in a corresponding relational database as either a *structured atomic value* or as a component of a *complex free-text value*. For example, the following FER schema specifies that a Person entity is associated with a street, a city, and a zip code:



"extract_street," "extract_city," or "extract_zip," which accepts a text string and returns the value of the desired address component. An ERA mapping expression invokes the external function by specifying a join between the database table `CONTACT_INFORMATION` and a virtual table that references the external function (see Section 4.1.3.6 for a discussion of virtual tables). For example, the mapping expression for the relationship type Person-Street is:

```
Person-Street::=   ERA = "Join([Complete_Address = input_address],
                               CONTACT_INFORMATION,
                               ADDRESS-TO-STREET)"
                   Fid = { Person([Soc_Sec_Number]),
                           String([street]) }.
```

The definition of the virtual table `ADDRESS-TO-STREET` is:

ADDRESS-TO-STREET =	<u>Field</u>	<u>Functional definition</u>
	`street:`	extract_street(`input_address`)
	`input_address:`	<no function>.

By allowing arbitrary external functions in mapping expressions, the ERA mapping languages significantly extends the power of the relational algebra to resolve the representation of entities and relationships within complex text strings.

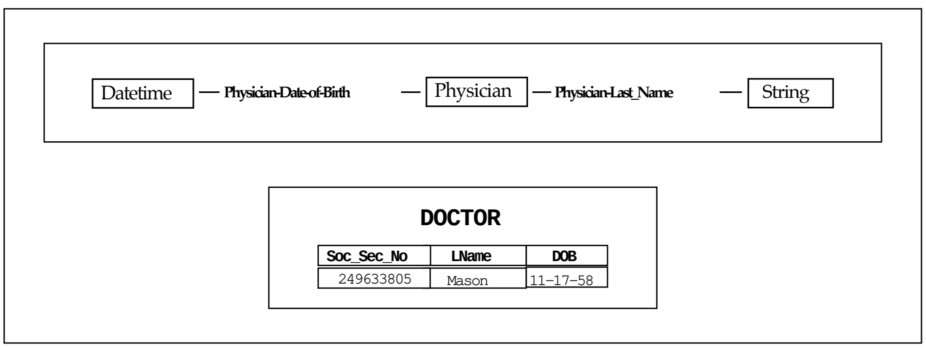
Although the address example can be handled by a simple parser, relational databases, in general, may represent clinical information in arbitrarily complex and unstructured forms that are very difficult to process automatically. For example, a relational database could store information regarding a patient's past medical problems and surgical procedures as a free-text admission note consisting of arbitrary natural-language sentences, phrases, and abbreviations. Although researchers have achieved some success in applying natural-language-processing (NLP) techniques to extract structured data from unstructured clinical documents [Sager, et al. 1994], no automated technique to process natural-language medical documents is widely available yet. It is important to note that, while this limitation constrains the set of relational implementations that the ERA mapping language can accommodate, no competing method for sharing database-intensive applications (including manual query-translation) can reliably extract clinical data that are represented as unstructured text strings.

A.3.2 Naming Differences

Naming differences are characterized by distinct lexical terms denoting the same semantic object in an FER schema and a relational database. Naming differences may be manifested as meta-data differences (that is, as variations in the *names* of relational tables and fields) or as data differences (that is, as variations in the *values* of relational fields).

A.3.2.1 Meta-data differences

Figure A.6 shows an FER schema and a relational schema that represent the same information content, specifically a set of physicians and their last names and dates of birth. The names of the tables and fields in the relational schema, however, are completely different than the names of the entity types and relationship types in the FER schema.

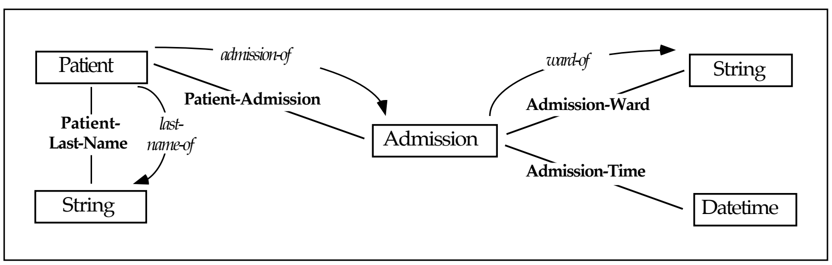


Physician-Last_Name::= ERA = `DOCTOR`
Id = [`Soc_Sec_Number`, `LName`]
Fid = { Physician([`Soc_Sec_Number`]),
String([`LName`]) }

If the table or the fields in the relational schema of Figure A.6 were renamed, the new names simply would be substituted into the appropriate slots in the mapping expressions. In this case, the absence of structural, semantic, or content differences between the FER schema and the relational schema make the mapping straightforward.

A.3.2.2 Data differences

Data differences are characterized by disparities between the symbols used to denote *instances* in an FER schema and the symbols used to denote the same instances in the fields of a relational database. For example, the FER schema in Figure A.7 specifies that each patient's admission to the hospital is associated with a hospital ward via the relationship type Admission-Ward.



ADMISSIONS

MRN	LName	Ward	Admit_date
147564773	Harmonic	Med	12-2-57
216426077	Flynn	Med	3-25-62
598044361	Collins	Peds	2-14-61
815466387	Hart	ICU	10-8-29

The SQL engine will perform the join and resolve the naming conflict at the time the translated query is executed. Because there is a one-to-one correspondence between terms denoting identical concepts, the disparity between the FER schema and the relational schema is simply a naming difference. If the correspondence between terms is not one-to-one (one-to-many, for example), a *semantic* difference exists. Semantic differences are more difficult to resolve, as discussed in the following section.

A.3.3 Semantic Differences

Semantic differences occur when the meaning of an entity type or a relationship type in an FER schema is similar to but not precisely equivalent to that of a corresponding data structure in a relational database. At least three kinds of semantic differences may exist between FER schemas and relational schemas: differences in value sets, differences in level of abstraction, and differences in relationship cardinality.

A.3.3.1 Differences in value sets

Differences in value sets exist when there is not a one-to-one correspondence between the terms that may appear in the fields of a relational database and the terms that are designated as instances of an entity type in the FER schema. This disparity occurs because there is not a one-to-one correspondence between the *concepts* that the terms denote. For example, an FER schema may specify that values for the entity type Blood_Culture_Growth are represented qualitatively as “no growth,” “moderate growth,” and “significant growth,” whereas a clinical database contains in the field Blood_Culture_Growth the values 0, 1+, 2+, 3+, or 4+. A semantic difference exists between the two value sets because the concept of moderate bacterial growth cannot be represented by any unique term in the value set {0, 1+, 2+, 3+, 4+}. In this case, a reasonable many-to-one mapping exists between the relational field values and the FER entity-type instances:

0 → “no growth”
1+ or 2+ → “moderate growth”
3+ or 4+ → “significant growth”

Because each element of the relational value set {0, 1+, 2+, 3+, 4+} maps to a unique element of the FER value set {“no growth,” “moderate growth,” "significant growth"}, one can map the relational schema to the FER schema by specifying a relational JOIN to a simple mapping table, as in the example of Section A.3.2.2. In this case, the necessary mapping table would be

GROWTH_MAPPING_TABLE

local_growth_code	standard_growth_code
0	no growth
1+	moderate growth
2+	moderate growth
3+	significant growth
4+	significant growth

In general, however, the correspondence between the value sets of a relational schema and an FER schema may be one to many or many to many, rather than many to one. In these cases, there are no unique mappings from elements of the relational value set to elements of the FER value set. For example, a one to many correspondence would exist if the relational value set were {"no growth," "moderate growth," "significant growth"} and the FER value set were {0, 1+, 2+, 3+, 4+}, that is, the inverse of the values above. This case would necessitate mapping from a single value to a set of values, which the mapping language does not explicitly support:

"no growth" $\rightarrow$ 0
"moderate growth" $\rightarrow$ {1+, 2+}
"significant growth" $\rightarrow$ {3+, 4+}

Theoretically, it would be useful to extend the ERA mapping language to represent one-to-many mappings if the TransFER query-translation mechanism were sufficiently sophisticated to infer that a ReFER query requesting culture results with a value of 1+ or 2+ was semantically equivalent to a relational query requesting culture results with a value of "moderate growth." Such queries would be rare, however.

A many-to-many (or overlapping) correspondence between value sets presents an even more difficult mapping problem. For example, a many-to-many correspondence would exist if the relational value set were {"light or no growth," "moderate or significant growth"} and the FER value set were {"no growth," "light or moderate growth," "significant growth"}. In this case, one could not accurately map the terms "light or no growth" and "moderate or significant growth" to even a subset of the FER value set because the meanings of the terms in the value sets overlap. For example, none of the following mappings are semantically accurate:

"light or no growth" $\rightarrow$ "no growth"
"light or no growth" $\rightarrow$ "light or moderate growth"
"light or no growth" $\rightarrow$ {"no growth," "light or moderate growth"}

One-to-many and many-to-many correspondences between value sets present difficult mapping challenges because either no correct mapping exists or there is loss of information when the mapping is performed. Although the ERA mapping language is unable to map certain correspondences between value sets, no competing method for sharing database-intensive applications (including manual query translation) can generate such information either.

A.3.3.2 Differences in level of abstraction

An FER schema may represent information in more or less detail than the relational database to which the schema is mapped. For example, the FER schema may include an entity type that represents the total number of previous pregnancies of an obstetrics patient, whereas the relational database may list and describe each previous pregnancy in detail (containing information about the duration of the pregnancy, method of delivery, etc.). In this case, the FER schema represents information regarding previous pregnancies at a higher level of abstraction than the relational database. To map from the more detailed to the less detailed representation, an ERA operation must *abstract* the detailed list of previous pregnancies in the relational database into a single numerical count of previous pregnancies. The AGGREGATE operator of the ERA can perform this abstraction by invoking the SQL aggregation function "COUNT." In general, the AGGREGATE operator can invoke any SQL or external aggregation function to abstract multiple rows of a relational table into a single row or value; this capability gives the ERA language the power to map from more detailed to less detailed representations.

Conversely, mapping from less detailed to more detailed representations is often impossible because necessary information may not be available. For example, if a relational database stores only the total number of previous pregnancies, but the FER schema represents detailed information about each previous pregnancy (such as its duration), there is no operator that can generate the detailed information from the abstract information. In this case, the mapping fails and any applications that request detailed pregnancy information via the FER schema must be notified that the information is not available in the database (see the discussion of missing information in Section A.3.4.3). Although the ERA mapping language is unable to generate more detailed information than is represented in a relational database, no competing method for sharing database-intensive applications (including manual query translation) can generate such information either.

A.3.3.3 Differences in relationship cardinality

The nature of relationships between object classes in conceptual database schemas may be one to one, one to many, or many to many [Batini, et al. 1992]. This characteristics of relationships is termed *cardinality*. The cardinality of a relationship type represented in an FER schema may differ from the cardinality of a corresponding association represented in a relational database. The ERA language must resolve this semantic difference so that the meaning of an FER schema is faithfully preserved when it is mapped to a relational schema.

For example, an FER schema may specify that each patient is associated with only one physician (the patient's primary physician), whereas a relational database may associate a patient with all of the physicians who are caring for her (whether the physicians are primary caregivers or consulting specialists). In this case, the FER schema specifies a many-to-one relationship between patients and physicians (because many patients may be associated with a physician, but only one physician may be associated with each patient), whereas the relational database allows a many-to-many relationship (because many patients may be associated with a physician and *vice-versa*). To preserve the many-to-one semantics of the FER schema, a mapping operation must designate one of the physicians associated with each patient as that patient's primary physician. The AGGREGATE operator of the ERA mapping language can perform this operation by invoking an external aggregation function that selects one physician from the set of physicians caring for a patient. The aggregation function must apply appropriate selection criteria to identify the primary physician, based on data available in the relational database.

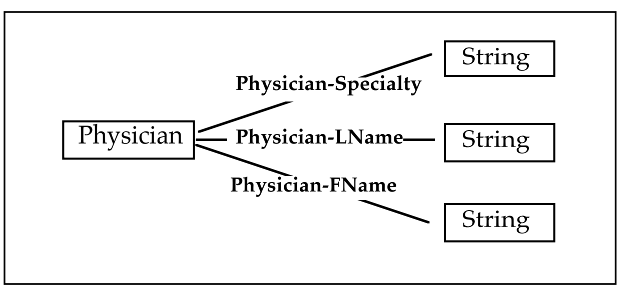
Conversely, if the FER schema specifies a many-to-many relationship between patients and physicians and the relational database represents a many-to-one relationship, no transformation is required in the mapping. This is because the relational database always will return one physician for each patient, which is a case subsumed by the many-to-many cardinality of the FER schema (that is, many to many implies many to one or more). The relational representation is, therefore, consistent with the semantics of the FER schema.

A.3.4 Content Differences

Content differences exist when information represented in the FER schema is not directly represented in a corresponding relational schema. Information that is not represented may be implicit, derivable, or simply missing.

A.3.4.1 Implicit information

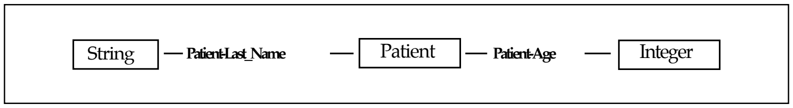
Information that is obvious and not subject to change at a particular clinical site is often omitted from that site's database. The same information, however, may not be obvious or immutable at every database site, and therefore will appear in the global FER schema. For example, the global FER schema probably will represent the clinical specialties of physicians because most institutions have multispecialty medical staffs:



The clinical database at a psychiatric hospital, however, may not explicitly represent the specialties of its staff physicians, because the hospital employs only psychiatrists. The relational table that represents the physician roster at such a facility may be modeled as follows:

PHYSICIANS

Soc_Sec_No	LName	FName
249633805	Mason	John
167453887	Collins	Laura
738099256	Hoffman	Abe



PATIENTS

MRN	LName	DOB
147564773	Harmonic	12-2-57
216426077	Flynn	3-25-62

A.3.4.3 Missing information

Despite the power of the ERA mapping language, there may be information represented in the FER schema that simply cannot be generated from data implicitly or explicitly stored in the relational database. For example, a specific clinical facility may choose to omit patients' HIV statuses from its electronic database for purposes of confidentiality, whereas the FER schema may include HIV status as an entity type. Currently, mapping expressions specify that a NULL value be returned when missing information is requested by a ReFER query. However, this convention is inadequate, because the meaning of NULL values in databases is ambiguous [Date 1989b]. In the case of HIV status, a NULL value may signify that the status is negative, the status is unknown, or the status is known but unavailable (the appropriate meaning in this case). The TransFER methodology needs an improved mechanism to represent explicitly that information requested by ReFER queries is missing and why it is missing. Further research in this area is necessary so that application-specific processing can occur when requested data is unavailable.

A.3.5 Summary

The ERA mapping language has the power to resolve many kinds of structural, semantic, naming, and content differences between FER schemas and relational databases. The addition of operators, such as AGGREGATE and EXTEND, to the "standard" relational algebra allows the ERA mapping language to exploit the full power of modern SQL query engines to resolve differences. The availability of external functions via the virtual-relation mechanism allows the ERA mapping language to apply arbitrary transformation functions to resolve differences that the SQL language is unable to handle. Despite the power afforded by these features, there exist certain types of structural, semantic, naming, and content differences that the ERA mapping language cannot resolve. In the case of most of these differences, no other method for porting queries across database platforms, including manual query translation, can accommodate them. However, research in natural language processing, medical knowledge representation, and clinical vocabulary standardization is underway that may soon overcome the obstacles to sharing clinical decision-support applications. Because the architecture of the TransFER methodology is formally specified and theoretically sound, TransFER will be able to take advantage of new technologies when they become available.