Appendix B

The ReFER Attribute Grammar

This chapter lists the complete site-independent attribute grammar for the TransFER query-translation methodology. The implementation of the attribute grammar (augmented with the appropriate site-specific mappings) translates ReFER queries to site-specific relational queries encoded in the extended relational algebra. As I describe in Sections 4.7 and 4.8, the query translator subsequently optimizes the queries and translates them to SQL. Section B.1 contains the complete context-free grammar of the ReFER query language and serves as a reference for writing ReFER queries. Section B.2 repeats the context-free grammar rules of Section B.1, and lists the semantic functions that correspond to each grammar rule and that define the ReFER attribute grammar.

B.1 The ReFER context-free grammar

The context free grammar is predominantly a syntactic description of valid ReFER queries. However, several important semantic constraints also are noted where relevant.

```
<Query> ::= <Term>    % Note:  <Term> must have no free variables for
                               <Query> to be a a valid query

<Term> ::= [ <Termlist> | <Range> ]

<Term> ::= <Factor_Expr>
```

<Termlist$_1$> ::= <Term> , <Termlist$_2$>

<Termlist> ::= <Term>

<Range> ::= <Decllist> AND <Predlist>

<Range> ::= <Decllist>

<Factor_Expr$_1$> ::= <Factor> + <Factor_Expr$_2$>

<Factor_Expr$_1$> ::= <Factor> - <Factor_Expr$_2$>

<Factor_Expr$_1$> ::= <Factor> * <Factor_Expr$_2$>

<Factor_Expr$_1$> ::= <Factor> / <Factor_Expr$_2$>

<Factor_Expr> ::= <Factor>

<Factor> ::= <Variable>

<Factor> ::= <Constant>

<Factor> ::= <AggFunction> (<Term>)

<Factor> ::= (<Factor_Expr>)

<Factor$_1$> ::= <Function> (<Factor$_2$>)

<Factor$_1$> ::= <Variable> ^^ <Function> (<Factor$_2$>)

<Variable> ::= *string* % E.g. pt, md, drug1, drug2

<Constant> ::= *quoted_string* % E.g. "Jones"

<Constant> ::= *integer* % E.g. 145

<Constant> ::= *real* % E.g. 5.5

<Constant> ::= *quoted_datetime* % E.g. "93/12/02 0800"

<AggFunction> ::= *aggregate_function* % One of {AVG, MIN, MAX, SUM, CNT}

<Function> ::= *FER_function* % FER_function must exist in FER schema

<Decllist$_1$> ::= <Decl> , <Decllist$_2$>

<Decllist> ::= <Decl>

<Decl> ::= (<Variable> : <Entity>)

<Entity> ::= *FER_entity_type* % Entity type must exist in the FER schema

<Predlist$_1$> ::= <Predicate> AND <Predlist$_2$>

<Predlist$_1$> ::= <Predicate> OR <Predlist$_2$

<Predlist> ::= <Predicate>

<Predicate> ::= (<Predlist)

<Predicate$_1$> ::= NOT <Predicate$_2$>

<Predicate> ::= EXISTS [<Range>]

<Predicate> ::= <Factor_Expr$_1$> Operator <Factor_Expr$_2$>

<Predicate> ::= <Relationship> (<Factor$_1$> , <Factor$_2$>)

<Predicate> ::= <Variable> ^^ <Relationship> (<Factor$_1$> , <Factor$_2$>)

<Operator> ::= *operator_string* % One of { =, <>, >, ≥, <, ≤ }

<Relationship> ::= *FER_relationship* % Relationship must exist in FER schema

B.2 The ReFER attribute grammar

Each context-free grammar rule is associated with a set of semantic functions that define the attribute values of certain symbols in the grammar rule. In the semantic function definitions, each attribute of a symbol follows the symbol and is delimited by a period. For example, the term "<Predicate>.ERA" denotes the value of the attribute "ERA" for the symbol "<Predicate>". In certain cases, the grammar rules in the following attribute grammar do not match exactly the grammar rules presented in Section 4.6 of the text, because the latter rules were simplified for expository purposes. In the semantic functions that define ERA attributes, extended relational algebra operators are boldfaced for clarity.

```
<Query> ::= <Term>

   <Query>.ERA = MProj([<Term>.Id],<Term>.ERA)
   <Query>.Id = <Term>.Id
   <Query>.Vid = <Term>.Vid
   <Query>.Fid = <Term>.Fid


<Term> ::= [ <Termlist> | <Range> ]

   <Termlist>.Symbol_Table = <Range>.Symbol_Table
   <Term>.ERA = Join([videq(<Termlist>.Vid,<Range>.Vid)],
                     <Term>.ERA, <Range>.ERA)
   <Term>.Id = <Termlist>.Id
   <Term>.Vid = vid_diff(vid_union(<Termlist>.Vid, <Range>.Vid),
                         <Range>.Declared),
   <Term>.Fid = Ø


<Term> ::= <Factor_Expr>

   <Factor_Expr>.Symbol_Table = <Term>.Symbol_Table
   <Term>.ERA = <Factor_Expr>.ERA
   <Term>.Id = <Factor_Expr>.Id
   <Term>.Vid = <Factor_Expr>.Vid
   <Term>.Fid = <Factor_Expr>.Fid


<Termlist1> ::= <Term> , <Termlist2>

   <Term>.Symbol_Table = <Termlist1>.Symbol_Table
   <Termlist2>.Symbol_Table = <Termlist1>.Symbol_Table
   <Termlist1>.ERA = Join([videq(<Term>.Vid,<Termlist2>.Vid)],
                          <Term>.ERA, <Termlist2>.ERA)
   <Termlist1>.Id = concat(<Term>.Id,<Termlist2>.Id)
   <Termlist1>.Vid = vid_union(<Term>.Vid,<Termlist2>.Vid)
   <Termlist1>.Fid = Ø
```

```
<Termlist> ::= <Term>

   <Term>.Symbol_Table = <Termlist>.Symbol_Table
   <Termlist>.ERA = <Term>.ERA
   <Termlist>.Id = <Term>.Id
   <Termlist>.Vid = <Term>.Vid
   <Termlist>.Fid = <Term>.Fid


<Range> ::= <Decllist> AND <Predlist>

   <Range>.Symbol_Table = <Decllist>.Symbol_Table
   <Predlist>.Symbol_Table = <Decllist>.Symbol_Table
   <Range>.ERA = Join([videq(<Decllist>.Vid,<Predlist>.Vid)],
                     <Decllist>.ERA, <Predlist>.ERA)
   <Range>.Id = Ø
   <Range>.Vid = vid_union(<Decllist>.Vid,<Predlist>.Vid)
   <Range>.Fid = Ø
   <Range>.Declared = <Decllist>.Declared


<Range> ::= <Decllist>

   <Range>.Symbol_Table = <Decllist>.Symbol_Table
   <Range>.ERA = <Decllist>.ERA
   <Range>.Id = <Decllist>.Id
   <Range>.Vid = <Decllist>.Vid
   <Range>.Fid = <Decllist>.Fid
   <Range>.Declared = <Decllist>.Declared


<Factor_Expr1> ::= <Factor> + <Factor_Expr2>

   <Factor>.Symbol_Table = <Factor_Expr1>.Symbol_Table
   <Factor_Expr2>.Symbol_Table = <Factor_Expr1>.Symbol_Table
   <Factor_Expr1>.ERA = Join([videq(<Factor>.Vid, <Factor_Expr2>.Vid)],
                            <Factor>.ERA, <Factor_Expr2>.ERA)
   <Factor_Expr1>.Id = <Factor>.Id + <Factor_Expr2>.Id
   <Factor_Expr1>.Vid = vid_union(<Factor>.Vid, <Factor_Expr2>.Vid)
   <Factor_Expr1>.Fid = Ø
```

<Factor_Expr$_1$> ::= <Factor> - <Factor_Expr$_2$>

<Factor>.Symbol_Table = <Factor_Expr$_1$>.Symbol_Table
<Factor_Expr$_2$>.Symbol_Table = <Factor_Expr$_1$>.Symbol_Table
<Factor_Expr$_1$>.ERA = **Join([**videq(<Factor>.Vid, <Factor_Expr$_2$>.Vid)**]**,
<Factor>.ERA, <Factor_Expr$_2$>.ERA**)**
<Factor_Expr$_1$>.Id = <Factor>.Id - <Factor_Expr$_2$>.Id
<Factor_Expr$_1$>.Vid = vid_union(<Factor>.Vid, <Factor_Expr$_2$>.Vid)
<Factor_Expr$_1$>.Fid = Ø

<Factor_Expr$_1$> ::= <Factor> * <Factor_Expr$_2$>

<Factor>.Symbol_Table = <Factor_Expr$_1$>.Symbol_Table
<Factor_Expr$_2$>.Symbol_Table = <Factor_Expr$_1$>.Symbol_Table
<Factor_Expr$_1$>.ERA = **Join([**videq(<Factor>.Vid, <Factor_Expr$_2$>.Vid)**]**,
<Factor>.ERA, <Factor_Expr$_2$>.ERA**)**
<Factor_Expr$_1$>.Id = <Factor>.Id * <Factor_Expr$_2$>.Id
<Factor_Expr$_1$>.Vid = vid_union(<Factor>.Vid, <Factor_Expr$_2$>.Vid)
<Factor_Expr$_1$>.Fid = Ø

<Factor_Expr$_1$> ::= <Factor> / <Factor_Expr$_2$>

<Factor>.Symbol_Table = <Factor_Expr$_1$>.Symbol_Table
<Factor_Expr$_2$>.Symbol_Table = <Factor_Expr$_1$>.Symbol_Table
<Factor_Expr$_1$>.ERA = **Join([**videq(<Factor>.Vid, <Factor_Expr$_2$>.Vid)**]**,
<Factor>.ERA, <Factor_Expr$_2$>.ERA**)**
<Factor_Expr$_1$>.Id = <Factor>.Id / <Factor_Expr$_2$>.Id
<Factor_Expr$_1$>.Vid = vid_union(<Factor>.Vid, <Factor_Expr$_2$>.Vid)
<Factor_Expr$_1$>.Fid = Ø

<Factor_Expr> ::= <Factor>

<Factor>.Symbol_Table = <Factor_Expr>.Symbol_Table
<Factor_Expr>.ERA = <Factor>.ERA
<Factor_Expr>.Id = <Factor>.Id
<Factor_Expr>.Vid = <Factor>.Vid
<Factor_Expr>.Fid = <Factor>.Fid

```
<Factor> ::= <Variable>

   <Variable>.Symbol_Table = <Factor>.Symbol_Table
   <Factor>.ERA = <Variable>.ERA
   <Factor>.Id = <Variable>.Id
   <Factor>.Vid = <Variable>.Vid
   <Factor>.Fid = <Variable>.Fid


<Factor> ::= <Constant>

   <Factor>.ERA = constant_relation(<Constant>.Value)
   <Factor>.Id = "constant_attribute_name"
   <Factor>.Vid = Ø
   <Factor>.Fid = Ø


<Factor> ::= <AggFunction> ( <Term> )

   <Term>.Symbol_Table = <Factor>.Symbol_Table
   <Factor>.ERA = Agg([<Term>.Id,                %function argument
                       <Term>.Vid,               %grouping attributes
                       <AggFunction>.Value,      %function name
                       NewAttrList],             %automatically generated
                       <Term>.ERA)
   <Factor>.Id = NewAttrList
   <Factor>.Vid = <Term>.Vid
   <Entity>.Fid = Ø


<Factor1> ::= <Function> ( <Factor2> )

   <Factor2>.Symbol_Table = <Factor1>.Symbol_Table
   <Factor1>.ERA = Join([idpred(<Factor2>.Id,
                                 <Function>.Fid[source], =)],
                        <Factor2>.ERA, <Function>.ERA)
   <Factor1>.Id = <Function>.Fid[target]
   <Factor1>.Vid = <Factor2>.Vid
   <Factor1>.Fid = Ø


<Factor1> ::= <Variable> ^^ <Function> ( <Factor2> )

   <Factor2>.Symbol_Table = <Factor1>.Symbol_Table
```

```
   <Factor1>.ERA = Join([idpred(<Factor2>.Id,
                              <Function>.Fid[source], =)],
                        <Factor2>.ERA, <Function>.ERA)
   <Factor1>.Id = <Function>.Fid[target]
   <Factor1>.Vid = <Factor2>.Vid
   <Factor1>.Fid = Ø


<Factor> ::= ( <Factor_Expr> )

   <Factor_Expr>.Symbol_Table = <Factor>.Symbol_Table
   <Factor>.ERA = <Factor_Expr>.ERA
   <Factor>.Id = <Factor_Expr>.Id
   <Factor>.Vid = <Factor_Expr>.Vid
   <Factor>.Fid = <Factor_Expr>.Fid


<Variable> ::= string
   <Variable>.ERA = lookup(string, ERA, <Variable>.Symbol_Table)
   <Variable>.Id = lookup(<string, Id, <Variable>.Symbol_Table)
   <Variable>.Id = vid(string,
                       lookup(string, Id, <Variable>.Symbol_Table))
   <Variable>.Fid = lookup(string, Fid, <Variable>.Symbol_Table)


<Constant> ::= quoted_string

   <Constant>.value = quoted_string


<Constant> ::= integer

   <Constant>.value = integer


<Constant> ::= real

   <Constant>.value = real


<Constant> ::= quoted_datetime

   <Constant>.value = quoted_datetime
```

```
<AggFunction> ::= aggregate_function   % one of AVG, MIN, MAX, SUM, CNT

   <AggFunction>.Value = aggregate_function



<Function> ::= FER_function

   <Function>.Relationship = lookup_FER(FER_function, Relationship)
   <Function>.Source = lookup_FER(FER_function, Source_Entity_Type)
   <Function>.Target = lookup_FER(FER_function, Target_Entity_Type)
   <Function>.ERA = lookup_mapping(<Function>.Relationship, ERA)
   <Function>.Fid[source] = lookup_mapping(<Function>.Relationship,
                                           fid[<Function>.Source])
   <Function>.Fid[target] = lookup_mapping(<Function>.Relationship,
                                           fid[<Function>.Target])


<Decllist1> ::= <Decl> , <Decllist2>

   <Decllist1>.ERA = Join([videq(<Decl>.Vid,<Decllist2>.Vid)],
                          <Decl>.ERA, <Decllist2>.ERA)
   <Decllist1>.Id = Ø
   <Decllist1>.Vid = vid_union(<Decl>.Vid, <Decllist2>.Vid)
   <Decllist1>.Fid = Ø
   <Decllist1>.Symbol_Table  = decl_union(<Decl>.Symbol_Table,
                                         <Decllist2>.Symbol_Table)
   <Decllist1>.Declared = decl_union(<Decl>.Declared,
                                    <Decllist2>.Declared)


<Decllist> ::= <Decl>

   <Decllist>.ERA = <Decl>.ERA
   <Decllist>.Id = <Decl>.Id
   <Decllist>.Vid =<Decl>.Vid
   <Decllist>.Fid = <Decl>.Fid
   <Decllist>.Symbol_Table = <Decl>.Symbol_Table
   <Decllist>.Declared = <Decl>.Declared


<Decl> ::= ( <Variable> : <Entity> )

   <Decl>.ERA = <Entity>.ERA
```

```
   <Decl>.Id = <Entity>.Id
   <Decl>.Vid = vid(<Variable>.Value, <Entity>.Id)
   <Decl>.Fid = <Entity>.Fid
   <Decl>.Symbol_Table = entry(<Variable>.Value,
                               <Entity>.ERA,
                               <Entity>.Id,
                               vid(<Variable>.Value, <Entity>.Id),
                               <Entity>.Fid)
   <Decl>.Declared = <Variable>.Value.


<Entity> ::= string     % Must be one of the entities in the FER schema

   <Entity>.ERA = lookup_mapping(string, ERA)
   <Entity>.Id = lookup_mapping(string, Id)
   <Entity>.Vid = Ø
   <Entity>.Fid = Ø


<Predlist1> ::= <Predicate> AND <Predlist2>

   <Predicate>.Symbol_Table = <Predlist1>.Symbol_Table
   <Predlist2>.Symbol_Table = <Predlist1>.Symbol_Table
   <Predlist1>.ERA = Join([videq(<Predicate>.Vid,<Predlist2>.Vid)],
                          <Predicate>.ERA, <Predlist2>.ERA)
   <Predlist1>.Id = Ø
   <Predlist1>.Vid = vid_union(<Predicate>.Vid, <Predlist2>.Vid)
   <Predlist1>.Fid = Ø


<Predlist1> ::= <Predicate> OR <Predlist2

   <Predicate>.Symbol_Table = <Predlist1>.Symbol_Table
   <Predlist2>.Symbol_Table = <Predlist1>.Symbol_Table
   <Predlist1>.ERA = Union(<Predicate>.ERA, <Predlist2>.ERA)
   <Predlist1>.Id = Ø
   <Predlist1>.Vid = vid_union(<Predicate>.Vid, <Predlist2>.Vid)
   <Predlist1>.Fid = Ø


<Predlist> ::= <Predicate>

   <Predicate>.Symbol_Table = <Predlist>.Symbol_Table
```

<Predlist>.ERA = <Predicate>.ERA
<Predlist>.Id = <Predicate>.Id
<Predlist>.Vid = <Predicate>.Vid
<Predlist>.Fid = <Predicate>.Fid

<Predicate> ::= (<Predlist>)

<Predlist>.Symbol_Table = <Predicate>.Symbol_Table
<Predicate>.ERA = <Predlist>.ERA
<Predicate>.Id = <Predlist>.Id
<Predicate>.Vid = <Predlist>.Vid
<Predicate>.Fid = <Predlist>.Fid

<Predicate$_1$> ::= NOT <Predicate$_2$>

<Predicate$_2$>.Symbol_Table = <Predicate$_1$>.Symbol_Table
<Predicate$_1$>.ERA = **Diff(**[],<Predicate$_2$>.ERA**)** %"raw" diff form
<Predicate$_1$>.Id = Ø
<Predicate$_1$>.Vid = Ø %"raw" Vid form; filled in later
<Predicate$_1$>.Fid = Ø

<Predicate> ::= EXISTS [<Range>]

<Range>.Symbol_Table = <Predicate>.Symbol_Table
<Predicate>.ERA = **Exists(**[],<Range>.ERA**)** %"raw" Exists() form
<Predicate>.Id = Ø
<Predicate>.Vid = vid_diff(<Range>.Vid,<Range>.Declared)
<Predicate>.Fid = Ø

<Predicate> ::= <Factor_Expr$_1$> <Operator> <Factor_Expr$_2$>

<Factor_Expr$_1$>.Symbol_Table = <Predicate>.Symbol_Table
<Factor_Expr$_2$>.Symbol_Table = <Predicate>.Symbol_Table
<Predicate>.ERA = **Join([**idpred(<Factor_Expr$_1$>.Id, <Factor_Expr$_2$>.Id,
Operator.Value)**]**,
<Factor_Expr$_1$>.ERA, <Factor_Expr$_2$>.ERA**)**
<Predicate>.Id = Ø
<Predicate>.Vid = vid_union(<Factor_Expr$_1$>.Vid, <Factor_Expr$_2$>.Vid)
<Predicate>.Fid = Ø

```
<Operator> ::= operator_string   % One of { =, <>, >, ≥, <, ≤ }

   Operator.Value = operator_string

<Predicate> ::= <Relationship> ( <Factor1> , <Factor2> )

   <Factor1>.Symbol_Table = <Predicate>.Symbol_Table
   <Factor2>.Symbol_Table = <Predicate>.Symbol_Table
   <Predicate>.ERA = Join([idpred(<Factor1>.Id, <Relationship>.Fid[1],
                                 =)],
                          <Factor1>.ERA,
                          Join([idpred(<Factor2>.Id,
                                       <Relationship>.Fid[2], =)],
                               <Factor2>.ERA, <Relationship>.ERA))
   <Predicate>.Id = Ø
   <Predicate>.Vid = vid_union(<Factor1>.Vid, <Factor2>.Vid)
   <Predicate>.Fid = Ø


<Predicate> ::= <Variable> ^^ <Relationship> ( <Factor1> , <Factor2> )

   <Factor1>.Symbol_Table = <Predicate>.Symbol_Table
   <Factor2>.Symbol_Table = <Predicate>.Symbol_Table
   <Relationship>.Symbol_Table = <Predicate>.Symbol_Table
   <Predicate>.ERA = Join([idpred(<Factor1>.Id,
                                 <Relationship>.Fid[1], =)],
                          <Factor1>.ERA,
                          Join([idpred(<Factor2>.Id,
                                       <Relationship>.Fid[2], =)],
                               <Factor2>.ERA, <Relationship>.ERA))
   <Predicate>.Id = Ø
   <Predicate>.Vid = vid_union(<Factor1>.Vid, <Factor2>.Vid)
   <Predicate>.Fid = Ø

<Relationship> ::= FER_relationship

   <Relationship>.ERA = lookup(FER_relationship, ERA,
                               <Relationship>.Symbol_Table)
   <Relationship>.Id = lookup(FER_relationship, Id,
                              <Relationship>.Symbol_Table)
   <Relationship>.Vid = Ø
   <Relationship>.Fid[1] = lookup_FER(FER_relationship, Fid[1])
   <Relationship>.Fid[2] = lookup_FER(FER_relationship, Fid[2])
```

Certain operators that appear in the semantic functions of the attribute grammar require explanation:

- **vid_union**: computes the union of two sets of vids, as described in Section 4.6.1 of the text.

- **vid_diff**: computes the difference between two sets of vids, such that only the vids in the first set that are not in the second set are returned. For example, if vid_1 = { (pt, [`fname, lname`]), (md,[`first, last`])} and vid_2 = { (pt, [`fname, lname`])}, then vid_diff(vid_1, vid_2) = {(md,[`first, last`])}.

- **videq**: expands a join condition consisting of a comparison between two vids, so that the attributes in the vids are compared pairwise, as described in Section 4.6.3 of the text.

- **idpred**: expands a join condition consisting of a comparison between two *sets* of identifying attributes, so that the elements of the sets are compared pairwise, as described in Section 4.6.3 of the text.

- **decl_union**: merges two symbol tables.

- **lookup**: indexes into the symbol table (third argument) by a variable name (first argument) and retrieves the requested component (second argument). For example, lookup(pt, ERA, Symbol_Table_1) returns the ERA expression defined for the entity type that was declared as the variable pt in the symbol table Symbol_Table_1.

- **lookup_FER**: indexes into the coded representation of the current FER schema and retrieves the requested information. For example, lookup(age-of, Relationship) returns the relationship type that corresponds to the relationship function "age-of."

- **lookup_mapping**: indexes into the ERA mappings defined for the current site by an entity-type or relationship-type name and retrieves the requested mapping information. For example, lookup(Person, Id) might return the set of attributes { first_name, last_name } (that is, the key attributes of the entity type Person).

In addition to the attributes ERA, Id, Vid, and Fid that I discussed in Sections 4.2 and 4.3.2 of the text, the attribute grammar may also assign the following attributes to ReFER grammar symbols:

- **Symbol_Table**: A set of declared variables and their associated ERA mappings. The mappings corresponding to variables are assigned based on the entity types for which the variables are declared. For example, if the declaration (pt : Patient) appears in a ReFER query, the symbol table includes an entry that binds the variable "pt" to the ERA mapping for the entity type "Patient."
- **Declared**: The set of variables that were explicitly declared within an expression. For example, the set of declared variables in the ReFER expression "[name-of(pt) | (pt: Patient), (md: Physician) AND Treats(md,pt)]" is { pt, md }.