A Quick Guide to EDDI

Introduction

EDDI is the Eden Definition Database Interpreter. It is a definitive database notation.

The EDDI notation is provided with the standard Eden distribution. However it is not enabled by default. To make the EDDI notation available in an Eden tool (eg `tkeden` or `ttyeden`), invoke the Eden function `installeddi` by typing `installeddi();` and pressing Accept (`tkeden`) or just return (`ttyeden`). It will then be possible to set the current notation to EDDI by typing `%eddi`, or by pressing the appropriate radio button in `tkeden`. This is possible in versions of Eden >= 1.30. The need to type `installeddi();` each time when starting Eden can be slightly eased by using the "execute" command line option: use (for example) `tkeden -e "installeddi();"` to start the tool with EDDI pre-installed.

The integrated version of the EDDI parser was developed from the older `eddip` pipeline-based translator, which is implemented in lex and yacc. This can still be used, but the convenience gained by the integration of the translator into Eden is great, so we recommend the integrated method.

Data definition commands, attribute types

The EDDI interpreter supports a range of data manipulation and data definition commands. There are five basic data definition commands: to create relation tables, to add or delete tuples from tables, and to drop or truncate tables.

The data definition commands to create a table and to insert tuples into a table have the form:
table_name **(** *comma separated list of attribute_name attribute_type pairs* **)**
table_name **<<** *comma separated list of tuples with appropriately typed fields*
The possible attribute types are limited to real (specified by the keyword REAL in upper or lower case), integer (INT) and string (CHAR). The use of these **create** and **insert** commands is illustrated by the definition of the tables `allfruits`, `apple` etc in the `FRUITS` database given as an appendix to this document. Note that string values are enclosed in double quotes. A **delete** command of the form:
table_name **!!** *comma separated list of tuples for deletion*
can be used to delete tuples from a table.

The EDDI interpreter supports both tables and views, and includes two commands to create tables and views from existing tables and views. These involve using a relational expression to specify the value of a variable as a relational table either through assignment or through definition. The syntax for an **assignment** is:
table_name **=** *relational expression whose operands are specified tables and views*
and that for a **definition** is:
view_name **is** *relational expression whose operands are specified tables and views*
In an assignment, the value of the variable on the left-hand side (LHS) is *assigned the current value* of the relational expression on the right-hand side (RHS). In a definition, the value of the LHS is *maintained to be the up-to-date value* of the relational expression on the RHS (cf. the definition of a cell in a spreadsheet). Assignment and definition in EDDI are the counterparts of the SQL commands that create tables and views as the results of a query. Table and view names in EDDI are alphanumeric strings with an initial alphabetic character (they should not include underscore characters).

The specification of a table as a view is illustrated in the following definitions of views `fruits` (specifying the names of fruits) and `popcitrus` (specifying the popular citrus as assessed by the volume of sales):

```
fruits is allfruits % name;
popcitrus is (fruits.citrus % name).(soldfruit : initsold > 50 % name);
```

Operators and operands

As illustrated in these definitions, the operands in a relational expression are relation names that refer to existing

tables or views. Six relational operators can be used in relational expressions. They are:

union (+), difference(-), intersection (.), projection(%), selection(:), join(*).

These operators have standard interpretations and are subject to precedence rules and semantic rules. As listed above, the operators are in increasing order of precedence. It is only possible to construct the union, difference or intersection of two relation tables if they have the same type, as determined both by the attribute names and the domains associated with their corresponding columns.

The **union** of two tables X+Y consists of those tuples that are to be found either in X or in Y or in both.

The **difference** between two tables X-Y consists of those tuples that are to be found in X but not in Y.

The **intersection** of two tables X.Y consists of those tuples that are to be found in both X and Y.

A **projection** of X on to the attributes A,B,C, ... (X%A,B,C,...) lists the distinct tuples that are generated from tuples in X by selecting the values from the columns associated with A,B,C ... Projection selects columns from a table.

The **selection** from X defined by the predicate p (X:p) comprises those tuples in X that satisfy the predicate p. Selection selects rows from a table.

The **join** of two relations X*Y consists of tuples that are derived from pairs of tuples in X and Y that agree on common attributes by concatenating them to form a new tuple and eliminating redundant columns. Note that the implementation of join and possibly some other operators is currently right-associative: X*Y*Z is X*(Y*Z), not (X*Y)*Z as might be expected.

In the EDDI interpreter, union, difference, intersection and join operate as standard binary operators whose syntax is similar to that used to specify multiplication and addition in ordinary arithmetic expressions. Projection and selection have a special syntax. The syntax for projection is

relation_name (or relation_expression) **%** *comma separated list of attribute names*

and that for selection is

relation_name (or relation_expression) **:** *simple predicate*

where the predicate takes the form of a comparison `X op Y` where `X` refers to the value of an attribute and `Y` is either an explicit value or refers to the value of an attribute. The valid comparators that can be substituted for `op` are:

= = , <= , >= , !=

A variant of projection is **project-and-rename**. This is specified by replacing an attribute name by an expression of the form

attribute name **>>** *attribute name*

in the comma separated list of attribute names that follows the % in a projection. Where joins, selections and projections are combined in a relational query, the default sequence in which these operations are executed is 'projection from a selection from a join', and brackets have to be used where another sequence is intended.

Query command

If you simply want to inspect the result of a relational algebra expression, you can use a **query** command of the form

? *relational expression;*

This is equivalent to making a one-off query in SQL. An alternative way to capture the current value of a relation expression is to use an assignment. For instance, with `fruits;` as defined above, the assignment:

```
X = fruits;
```

will assign to the variable `X` the table consisting of a single column listing the names of fruits cited in `allfruits`. Assigning a table or defining a view has the merit of recording the results of your previous queries for inspection or re-use.

Cyclic definitions are not allowed

Definition and assignment is subject to some semantic restrictions. It is only possible to make definitions and assignments if the value of the relational expression on the RHS is defined. A view cannot be defined in a cyclic

manner, so that for instance the pair of definitions

```
X is Y; Y is X;
```

is invalid.

The catalogue

The EDDI interpreter maintains a simple catalogue of the relations currently in the database in the special table CATALOGUE. The CATALOGUE relation has just two attributes: it records known relation names and specifies whether they are tables or views.

The command

#

is a shorthand for

? *CATALOGUE*

A **describe** command of the form

?? *relation_name*

can be used to determine the current status of relations in the database (cf the use of DESCRIBE in SQL). The information returned by describe includes the type of the relation, as determined by the list of domain types and attribute names of its columns, the list of views (if any) that are defined in terms of the relation, and its current size. Where appropriate, describe also returns the current definition of the relation.

Dropping data

Relation names that appear in the CATALOGUE cannot be reused in a definition or assignment. In order to redefine or reassign a relation name, it is first necessary to drop the relation from the CATALOGUE using additional data definition commands for managing the relations in the database. Apart from the CATALOGUE itself, which cannot be directly manipulated, tables and views that are not currently used in the definition of views can be dropped using the **drop** command of the form:

~~*relation_name*

All the tuples can be deleted from a table by using the **truncate** command of the form:

~*table_name*

Comments

Single lines prefixed by ## in an EDDI file are treated as comments. Since EDDI commands are for the most part very concise, a semi-colon (;) or a newline serve interchangeably as statement separators. This precludes multi-line EDDI input.

Known bugs and limitations

The EDDI interpreter offers no support for specifying keys or constraints on relations. Its usefulness in the Windows environment is limited by the size of the dos console window in which output is given. The user does not always get feedback when actions are performed (e.g. when deleting tuples or dropping tables etc).

Because it uses an unconventional observation-oriented parser, the interpreter sometimes deals in unexpected ways with invalid input. For the most part, syntactic errors are trapped by EDDI itself, but occasionally errors will be reported via the underlying eden interface. In exceptional circumstances, it is possible for the parser itelf to request input from the user in the form of a string to match a particular syntactic construct. The most appropriate response to such a request is to type 'q' in the console window to cancel. Please attempt to reproduce and document any more serious errors you encounter, such as result in crashing the interpreter, and report them to wmb@dcs.

Appendix: The EDDI commands to generate the FRUITS database

```
%eddi

allfruits (name CHAR, begin INT, end INT);
allfruits << ["granny",8,10],["lemon",5,12],["kiwi",5,6],["passion",5,7];
allfruits << ["orange",4,11],["grape",3,6],["lime",4,7],["pear",4,8];
allfruits << ["cox",1,12],["red",4,8];

apple (name CHAR, price REAL, qnt INT);
apple << ["cox",0.20,8],["red",0.35,4],["granny",0.25,10];

citrus (name CHAR, price REAL, qnt INT);
citrus << ["lime",0.30,3],["orange",0.55,8],["kiwi",0.75,5],["lemon",0.50,2];

soldfruit (name CHAR, unitsold INT);
soldfruit << ["cox",100],["granny",153],["red",70];
soldfruit << ["kiwi",23],["lime",15],["lemon",55],["orange",78];

fruits is allfruits % name;
popcitrus is (fruits.citrus % name).(soldfruit : unitsold > 50 % name);
```
