RA Relational Algebra Syntax Guide

Introduction

RA is a relational algebra interpreter that translates relational algebra queries into SQL queries, then executes the SQL on a standard relational database system. RA was developed by Prof. Jun Yang at Duke University. The version of RA we're using for the workbench runs on the SQLite open-source relational database system.
The simplest relational algebra expression is one that returns the contents of a single relation: just write "`relName`", where `relName` is the name of the relation. Relation and attribute names are case-insensitive; for example, `pizzeria` is the same as `PIZZERIA`. Every operator starts with a backslash (\). Here is an example of a complex query; it finds all pizzas eaten by at least one person who does not frequent the 'Dominos' pizzeria.

```
\project_{pizza} (
   ((\project_{name} Person)          // all people
    \diff
    (\project_{name}                  // people who frequent Dominos
        \select_{pizzeria='Dominos'}
             Frequents)
   \join Eats))                        // join with Eats to find pizzas
```

The syntax is insensitive to whitespace, and queries may span multiple lines.

Operators

RA supports the following relational algebra operators:

- `\select_{cond}` is the relational selection operator. For example, to select people with name Amy or Ben, we write "`\select_{name='Amy' or name='Ben'} Person`". The syntax for *cond* follows SQL: string literals can be enclosed in single or double quotes, and boolean operators `and`, `or`, and `not` may be used. Comparison operators <=, <, =, >, >=, and <> work on both string and numeric types.
- `\project_{attr_list}` is the relational projection operator, where `attr_list` is a comma-separated list of attribute names. For example, to find the pizzas served by Applewood (but without the price information), we write "`\project_{pizza} (\select_{pizzeria='Applewood'} Serves)`".
- `\cross` is the relational cross-product operator. For example, to compute the cross-product of `Person` and `Frequents`, we write "`Person \cross Frequents`".
- `\join` is the relational natural join operator. For example, to join `Person(name,age,gender)` and `Frequents(name,pizzeria)` enforcing equality on the shared `name` attribute, we simply write "`Person \join Frequents`". Natural join automatically equates all pairs of identically named

attributes from its inputs (in this case, `name`), and outputs only one attribute per matching pair. The schema of the result in our example is `(name,age,gender,pizzeria)`.

- `\join_{`*`cond`*`}` is the relational theta-join operator. For example, to join the two relations `Person(name,age,gender)` and `Serves(pizzeria,pizza,price)` enforcing that the pizza price is lower than the person's age, we write "`Person \join_{age>price} Serves`". Syntax for *cond* again follows SQL; see notes above for `\select`.
- `\union`, `\diff`, and `\intersect` are the relational union, difference, and intersection operators, respectively. As a trivial example, to compute the union between `Person` and itself, we write "`Person \union Person;`", which returns the original `Person` relation. To compute the difference between `Person` and itself, we write "`Person \diff Person;`", which returns the empty relation. To compute the intersection between `Person` and itself, we write "`Person \intersect Person;`", which again returns the original `Person` relation. **Warning:** RA allows these operators to be applied to any two subexpressions that produce an equal number of attributes, even if the corresponding attribute names don't match. (See also the note about attribute order under **Limitations** below.) This allowance is typical of most SQL implementations but violates the requirements of pure relational algebra. As good practice, and for unambiguous attribute names in the result, we suggest using the `\rename` operator (next) as needed to enforce matching schemas whenever `\union`, `\diff`, or `\intersect` is used.
- `\rename_{`*`new_attr_name_list`*`}` is the relational renaming operator, where *`new_attr_name_list`* is a comma-separated list of new names, one for each attribute of the input relation. For example, to rename the attributes of relation `Person` and compute the cross-product of `Person` with itself, we write "`\rename_{name1,age1,gender1} Person \cross \rename_{name2,age2,gender2} Person;`". Note we could equivalently write "`\rename_{name1,age1,gender1,name2,age2,gender2} (Person \cross Person);`".

Limitations

Currently, RA has the following limitations:

- `\rename` only supports renaming of attributes; it does not support renaming of relations.
- RA expressions may yield multiple attributes with the same name, but it is not possible to refer to the ambiguously-named attributes "later" in the expression. If a subexpression yields multiple attributes with the same name and you wish to refer to one or more of them, you must use the `\rename` operator to make them unique.
- The standard *`relName.attrName`* notation for referencing an attribute is neither needed nor supported. Using the `\rename` operator, attribute name names can always be made unambiguous.
- As in most SQL implementations, attribute order is relevant in relations and in the result of expressions. This property is significant primarily for set operators, which as mentioned above do not consider attribute names. For example, if we take the `\union` of relations `R(A,B)` and `S(B,A)`, the result contains two columns: (1) the union of the `A` values in `R` and the `B` values in `S`; (2) the union of the `B`

values in `R` and the `A` values in `S`. As mentioned above, as good practice we suggest using the `\rename` operator to enforce matching schemas on `R` and `S` before applying the `\union`.

- Error messages in response to ill-formed RA expressions may not be especially meaningful. Recall that RA translates relational algebra expressions into SQL queries. Often an incorrect RA expression simply results in incorrect SQL queries. In these cases, RA just passes back the error messages from the underlying DBMS, without attempting to create RA-specific messages.