INFO0009 - Bases de données

(E)RALI: (Extended) Relational Algebra Learning Instrument

Christophe Debruyne
c.debruyne@uliege.be

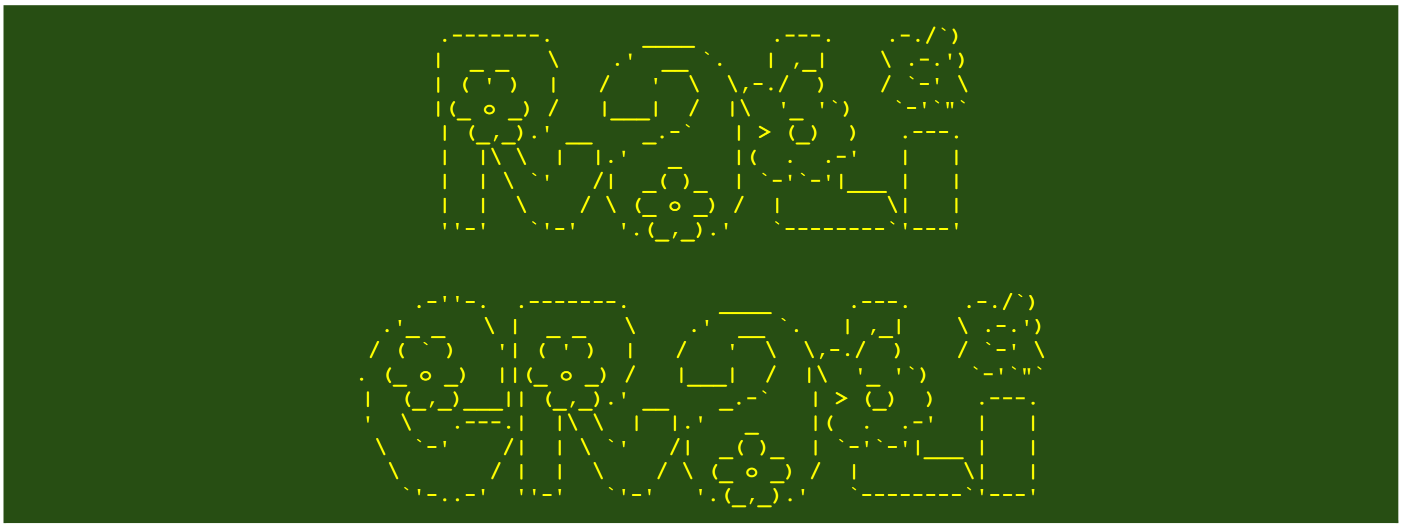

Table of Contents

Introduction

The Relational Algebra Learning Instrument (RALI) and Extended Relational Algebra Learning Instrument (ERALI) have been developed to aid students in learning (extended) relational algebra. Many existing tools deviate from the relational model (e.g., the implicit qualification of attributes). With this tool, I provide students with a relational algebra implementation that follows the syntax and semantics seen in class.[1] Another syntactical difference is that we use = to assign the result of a relational algebra expression to a relation instead of ←.

However, this implementation has a couple of important differences. First, the default datatype is assumed to be `STRING`s. Attributes that are `INTEGER`s, `DOUBLE`s, or `DATE`s must be “typed” as such. Applying a function on attributes of different types may yield and error if coercion (i.e., implicitly converting between types) is impossible.

The implementation is, furthermore, a prototype. It has been (hastily) developed to aid you in your learning. As such, there may be bugs. If you encounter a bug, feel free to send an email to c.debruyne@uliege.be. While initially inconvenient, all operations, relations, and attributes must be written in UPPER CASE. This tool aims to help you formulate queries in relational algebra. At the exam, you will be asked to formulate queries by hand, using the formalism we have seen in the course. You must use the mathematical and Greek symbols at the exam.

Relational Algebra Learning Instrument

Running RALI

To run RALI, you must have the Java Runtime Environment (JRE) or the OpenJRE installed. Download and extract the JAR file. To launch the program, open a terminal in the directory where the JAR file is stored and run the following command `$ java -jar rali.jar`. You should see the following screen.

```
C:\Users\chris\Desktop>java -jar rali.jar
.-------.       ____      .---.    .-./`)
|  _ _   \    .'  __ `.   | ,_|    \ .-.')
| ( ' )  |   /   '  \   \,-./  )    / `-' \
|(_ o _) /   |___|  /  |\  '_ '`)   `-'`"`
| (_,_).' __    _.-`   | > (_)  )   .---.
|  |\ \  |  |.'   _    |(  .  .-'   |   |
|  | \ `'   /|  _( )_  | `-'`-'|___ |   |
|  |  \    / \ (_ o _) /  |        \|   |
''-'   `'-'   '.(_,_).'   `--------`'---'
Welcome to RALI, the Relational Algebra Learning Instrument!

Type :quit to exit.
Type :examples to load the default relations STUDENTS, ENROLLMENTS, and COURSES.
Type :letters to load the default relations R, S, T, U, V, X and Y.
```

The interface is pretty straightforward. You have four options: existing the environment with `:quit`, loading two example databases, students and letters, with `:examples` and `:letters`, respectively, and formulating queries. RALI requires all statements to end with a semicolon (;). However, you may omit those in the terminal as the tool appends one as a little “usability hack.”

When you launch RALI, all CSV files that follow respect a particular format will be loaded as a relation. Each CSV file will be loaded as a relation as follows:

1. The file's name becomes the relation's name (e.g., `foo.csv` will yield the relation `FOO`).
2. The first row must contain the names of the attributes (which must be unique and may be typed).
    a. Attributes are, by default, of the type `STRING.`
    b. Attributes may be typed by adding a colon (‘:’) followed by one of the following types: `STRING`, `DATE`, `INTEGER`, and `DOUBLE`.
3. Values cannot be NULL.
4. `STRING`s may be quoted (unless they contain quotes and commas)
    a. Use backslash to quote quotes that are part of the `STRING` value.
    b. `DATE`s must follow the following convention `yyyy-mm-dd`.

Here is an example of a CSV file `foo.csv` and the program’s behavior on startup:

Contents of `foo.csv`

```
A, B : STRING, C : INTEGER, D: DATE
a1,"b1 1",1,2023-01-05
a2,"b2, 2",2,2023-02-06
a3,b3,3,2023-03-07
```

[1] Aside from mathematical operator notations and Greek symbols, which have been replaced by the keywords `PRODUCT`, `TIMES`, … and `SELECT`, `PROJECT`, `RENAME`, … respectively

The relations appear upon startup:

```
Relation FOO created.
```

Upon formulating the query FOO:

```
FOO
+----+-------+-------------+------------+
| A  | B     | C : INTEGER | D : DATE   |
+----+-------+-------------+------------+
| a1 |  b1 1 |           1 | 2023-01-05 |
+----+-------+-------------+------------+
| a2 | b2, 2 |           2 | 2023-02-06 |
+----+-------+-------------+------------+
| a3 |    b3 |           3 | 2023-03-07 |
+----+-------+-------------+------------+
```

The implementation of RALI relies on an in-memory H2 database. This relational database implementation, see Chapter 4, will automatically order tuples based on the values of attributes `A1, …, An` if no indices and `ORDER BY` directives are provided. Remember that relations have a set of tuples, and the order of tuples is not important.

Syntax and examples

The following section assumes you have loaded both `:examples` and `:letters`.

The precedence of relational algebra operators are, from high to low, as follows:

1. ~~The unary operators (~~σ, π, ρ)~~;~~
    a. We enforce parenthesis around the expression, so we do not have this "problem."
2. The Cartesian product and its derivatives ($\times$, $\div$, $\bowtie$, $\bowtie_\theta$);
3. The set intersection ∩;
4. The set difference − and set union ∪;

Whenever we have a sequence of operations with the same precedence, they are evaluated from left to right (left-associative). For example, $A - B \cup C - D$ is interpreted as $((A - B) \cup C) - D$. Here are other examples:

- `A UNION B MINUS C INTERSECTION D JOIN E UNION F PRODUCT G`
- The cartesian product and its derivatives have the highest priority
    - `A UNION B MINUS C INTERSECTION (D JOIN E) UNION (F PRODUCT G)`
- We then have the intersection
    - `A UNION B MINUS (C INTERSECTION (D JOIN E)) UNION (F PRODUCT G)`
- The remaining operations are of the same precedence and are thus evaluated in a left-associative manner.
    - `((A UNION B) MINUS (C INTERSECTION (D JOIN E))) UNION (F PRODUCT G)`

It is a good idea to use parentheses, even if you know what you are doing. ;-)

Simple relations

Expressions in relational algebra start from simple relations (or constant relations; see later). Expressions in relational algebra can also be surrounded with parentheses.

Example of querying the constants of a relation
`S;`

```
S;
+---+---+---+
| A | B | C |
+---+---+---+
| b | g | a |
+---+---+---+
| d | a | f |
+---+---+---+
```

Example of querying the constants of a relation (with parenthesis)
`(S);`

```
(S);
+---+---+---+
| A | B | C |
+---+---+---+
| b | g | a |
+---+---+---+
| d | a | f |
+---+---+---+
```

Assignments

The result of an expression can be assigned to a relation. The relation's name should not have been used before the assignment, or you will have an error. Unlike the formalism we have seen in the course, we use = instead of ←.

Example of an assignment based on a relation
`T2 = T;`

```
T2 = T;
+---+---+
| D | E |
+---+---+
| a | b |
+---+---+
| c | d |
+---+---+
```

Example of an assignment based on a complex expression
`R_AND_S = R INTERSECTION S;`

```
R_AND_S = R INTERSECTION S;
+---+---+---+
| A | B | C |
+---+---+---+
| d | a | f |
+---+---+---+
```

The fundamental binary operations

Example of a union
`R UNION S;`

```
R UNION S;
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
| b | g | a |
+---+---+---+
| c | b | d |
+---+---+---+
| d | a | f |
+---+---+---+
```

Example of a cartesian product
`R PRODUCT S;` *(should not work)* and
`R PRODUCT T;`

```
R PRODUCT S;
LHS and RHS of the Cartesian Product
share attributes: [A, B, C].
R PRODUCT T;
+---+---+---+---+---+
| A | B | C | D | E |
+---+---+---+---+---+
| a | b | c | a | b |
+---+---+---+---+---+
| a | b | c | c | d |
+---+---+---+---+---+
| c | b | d | a | b |
+---+---+---+---+---+
| c | b | d | c | d |
+---+---+---+---+---+
| d | a | f | a | b |
+---+---+---+---+---+
| d | a | f | c | d |
+---+---+---+---+---+
```

Example of a difference
`R MINUS S;`

```
R MINUS S;
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
| c | b | d |
+---+---+---+
```

The fundamental operations on one relation

As stated earlier. The implementation of RALI relies on an in-memory H2 database, which will order the tuples based on the values of their attributes. Sets have no order, and the following sets are therefore {(0, "a"),(1, "b")} and {(1, "b"),(0, "a")} equivalent. You will notice this behavior in the examples below. When using RENAME, RALI will put the renamed attributes first (in the order they have been provided) and the remainder of the attributes last. This does not affect the semantics of the relations as we are using a named perspective.

Example of a projection
`PROJECT{B,A}(R PRODUCT T);`

```
PROJECT{B,A}(R PRODUCT T);
+---+---+
| B | A |
+---+---+
| a | d |
+---+---+
| b | a |
+---+---+
| b | c |
+---+---+
```

Example of a selection
`SELECT NOT C="d" (R);`

```
SELECT NOT C="d" (R);
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
| d | a | f |
+---+---+---+
```

Important: the precedence order of logical operators is NOT, AND, and OR. Whenever we have a condition where we have a (sub-)expression containing the same operator without parentheses, they are evaluated from left to right (left-associative).

Example of a selection
`SELECT ((NOT (AGE < 20) AND MAJOR = "CS") OR "Bob" = NAME) (STUDENTS);`

```
SELECT ((NOT (AGE < 20) AND MAJOR = "CS") OR "Bob" = NAME) (STUDENTS);
+----------------------+-------+---------------+-------+
| STUDENT_ID : INTEGER | NAME  | AGE : INTEGER | MAJOR |
+----------------------+-------+---------------+-------+
|                    1 | Alice |            20 |    CS |
+----------------------+-------+---------------+-------+
|                    2 |   Bob |            21 |  Math |
+----------------------+-------+---------------+-------+
|                    5 | Emily |            20 |    CS |
+----------------------+-------+---------------+-------+
```

Example of a rename
`RENAME A<-X,C<-Y (R);`

```
RENAME A<-X,C<-Y (R);
+---+---+---+
| X | Y | B |
+---+---+---+
| a | c | b |
+---+---+---+
| c | d | b |
+---+---+---+
| d | f | a |
+---+---+---+
```

Example of a rename
`RENAME C<-Y,A<-X (R);`

```
RENAME C<-Y,A<-X (R);
+---+---+---+
| Y | X | B |
+---+---+---+
| c | a | b |
+---+---+---+
| d | c | b |
+---+---+---+
| f | d | a |
+---+---+---+
```

Derived operations

Example of an intersection
`R INTERSECTION S;`

```
R INTERSECTION S;
+---+---+---+
| A | B | C |
+---+---+---+
| d | a | f |
+---+---+---+
```

Example of a natural join
`R JOIN U;`

```
R JOIN U;
+---+---+---+-------------+
| B | A | C | G : INTEGER |
+---+---+---+-------------+
| a | d | f |           1 |
+---+---+---+-------------+
```

Example of a division
`X DIVISION Y;`

```
X DIVISION Y;
+---+---+
| A | B |
+---+---+
| a | b |
+---+---+
```

Example of a natural join. When the two relations do not share an attribute, then it behaves as a Cartesian product.
`R JOIN T;`

```
R JOIN T;
+---+---+---+---+---+
| A | B | C | D | E |
+---+---+---+---+---+
| a | b | c | a | b |
+---+---+---+---+---+
| a | b | c | c | d |
+---+---+---+---+---+
| c | b | d | a | b |
+---+---+---+---+---+
| c | b | d | c | d |
+---+---+---+---+---+
| d | a | f | a | b |
+---+---+---+---+---+
| d | a | f | c | d |
+---+---+---+---+---+
```

When both relations have the same set of attributes, then the natural join behaves as an intersection.
`R JOIN S;`

```
R JOIN S;
+---+---+---+
| A | B | C |
+---+---+---+
| d | a | f |
+---+---+---+
```

Example of a Theta-Join
`R JOIN C=D OR B=E T;`

```
R JOIN C=D OR B=E T;
+---+---+---+---+---+
| A | B | C | D | E |
+---+---+---+---+---+
| a | b | c | a | b |
+---+---+---+---+---+
| a | b | c | c | d |
+---+---+---+---+---+
| c | b | d | a | b |
+---+---+---+---+---+
```

The use of parentheses makes the query arguably more readable.
`R JOIN (C=D OR B=E) T;`

```
R JOIN (C=D OR B=E) T;
+---+---+---+---+---+
| A | B | C | D | E |
+---+---+---+---+---+
| a | b | c | a | b |
+---+---+---+---+---+
| a | b | c | c | d |
+---+---+---+---+---+
| c | b | d | a | b |
+---+---+---+---+---+
```

Constant relations

You can use constant relations in your queries or assign its result to a relation. Its syntax closely follows that of the course. The only differences are that attributes can be types and that `STRING`s must be quoted.

Example of a constant relation

```
[COURSE_ID : INTEGER,  COURSE_NAME]{(101,"Calculus"),(102,"Biology")};
```

```
[COURSE_ID : INTEGER,  COURSE_NAME]{(101,"Calculus"),(102,"Biology")};
+---------------------+-------------+
| COURSE_ID : INTEGER | COURSE_NAME |
+---------------------+-------------+
|                 101 |    Calculus |
+---------------------+-------------+
|                 102 |     Biology |
+---------------------+-------------+
```

Creating a relation based on a constant relation

```
BAR = [COURSE_ID : INTEGER,  COURSE_NAME]{(101,"Calculus"),(102,"Biology")};
```

```
BAR = [COURSE_ID : INTEGER,  COURSE_NAME]{(101,"Calculus"),(102,"Biology")};
+---------------------+-------------+
| COURSE_ID : INTEGER | COURSE_NAME |
+---------------------+-------------+
|                 101 |    Calculus |
+---------------------+-------------+
|                 102 |     Biology |
+---------------------+-------------+
```

Using constant relations in a query

```
[A : INTEGER]{(1),(5)} JOIN A < B [B : INTEGER]{(3),(4)};
```

```
[A : INTEGER]{(1),(5)} JOIN A < B [B : INTEGER]{(3),(4)};
+-------------+-------------+
| A : INTEGER | B : INTEGER |
+-------------+-------------+
|           1 |           3 |
+-------------+-------------+
|           1 |           4 |
+-------------+-------------+
```

Extended Relational Algebra Learning Instrument

Important differences between relational algebra and extended relational algebra include:

- Operating on multisets (or bags)
- The introduction of *null* values
- Ordering tuples
- Generalized projections
- Grouping values of attributes to compute aggregates
- Outer joins
- Defining operators so that they work on multisets and null values

This part of the manual will assume that you re familiar with relational algebra (and RALI).

Running ERALI

To run ERALI, you must have the Java Runtime Environment (JRE) or the OpenJRE installed. Download and extract the JAR file. To launch the program, open a terminal in the directory where the JAR file is stored and run the following command `$ java -jar erali.jar.` You should see the following screen.

```
C:\Users\chris\Desktop>java -jar erali.jar
    .-''-.  .-------.       ____      .---.    .-./`)
  .'_ _   \ |  _ _   \    .'  __ `.   | ,_|    \ .-.')
 / ( ` )   '| ( ' ) |   /   '  \  \,-./  )    / `-' \
. (_ o _)  ||(_ o _) /   |___|  /  |\  '_ '`)   `-'`"`
|  (_,_)___|| (_,_).' __    _.-`   | > (_)  )   .---.
'  \   .---.|  |\ \  |  |.'   _    |(  .  .-'   |   |
 \  `-'    /|  | \ `'   /|  _( )_  | `-'`-'|___ |   |
  \       / |  |  \    / \ (_ o _) /  |        \|   |
   `'-..-'  ''-'   `'-'   '.(_,_).'   `--------`'---'
Welcome to RALI, the Extended Relational Algebra Learning Instrument!

Type :quit to exit.
Type :examples to load the default relations STUDENTS, ENROLLMENTS, and COURSES.
Type :letters to load the default relations R, S, T, U, V, X and Y.
```

The interface is pretty straightforward. You have four options: existing the environment with `:quit`, loading two example databases, students and letters, with `:examples` and `:letters`, respectively, and formulating queries.

Loading CSV files on startup

ERALI also supports the loading of CSV files. The only differences are that null values and multiple occurrences of the same tuple are allowed.

Here is an example of a CSV file `foo.csv` and the program's behavior on startup:

Contents of `bar.csv.` The first tuple contains a null value for B and the second tuple contains a null value for D. The last two tuples are duplicates.

```
A, B : STRING, C : INTEGER, D: DATE
a1,,1,2023-01-05
a2,"b2, 2",2,
a3,b3,3,2023-03-07
a3,b3,3,2023-03-07
```

The relations appear upon startup:

```
Relation BAR created.
```

Upon formulating the query FOO:

```
BAR
+----+-------+-------------+------------+
| A  | B     | C : INTEGER | D : DATE   |
+----+-------+-------------+------------+
| a1 |  NULL |           1 | 2023-01-05 |
+----+-------+-------------+------------+
| a2 | b2, 2 |           2 |       NULL |
+----+-------+-------------+------------+
| a3 |    b3 |           3 | 2023-03-07 |
+----+-------+-------------+------------+
| a3 |    b3 |           3 | 2023-03-07 |
+----+-------+-------------+------------+
```

Syntax and Examples

Set operations

Set operations now take into account multisets. We will exemplify all of the set operations. We have also provided “set versions” of each operation. We will use the following relations for the examples:

```
R1;
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
| a | b | c |
+---+---+---+
| d | a | f |
+---+---+---+
| d | a | f |
+---+---+---+
| d | a | f |
+---+---+---+
```

```
R2;
+---+---+---+
| A | B | C |
+---+---+---+
| b | g | a |
+---+---+---+
| d | a | f |
+---+---+---+
| d | a | f |
+---+---+---+
```

```
R3;
+---+---+---+
| C | D | E |
+---+---+---+
| c | d | e |
+---+---+---+
| c | d | e |
+---+---+---+
```

Multiset intersection

```
R1 INTERSECTION R2;
+---+---+---+
| A | B | C |
+---+---+---+
| d | a | f |
+---+---+---+
| d | a | f |
+---+---+---+
```

Intersection

```
R1 SET INTERSECTION R2;
+---+---+---+
| A | B | C |
+---+---+---+
| d | a | f |
+---+---+---+
```

Multiset union

```
R1 UNION R2;
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
| a | b | c |
+---+---+---+
| d | a | f |
+---+---+---+
| d | a | f |
+---+---+---+
| d | a | f |
+---+---+---+
| b | g | a |
+---+---+---+
| d | a | f |
+---+---+---+
| d | a | f |
+---+---+---+
```

Union

```
R1 SET UNION R2;
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
| b | g | a |
+---+---+---+
| d | a | f |
+---+---+---+
```

Multiset difference

```
R1 MINUS R2;
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
| a | b | c |
+---+---+---+
| d | a | f |
+---+---+---+
```

Difference

```
R1 SET MINUS R2;
+---+---+---+
| A | B | C |
+---+---+---+
| a | b | c |
+---+---+---+
```

For the division operator, we will use the following relation:

```
R4;
+---+---+
| A | B |
+---+---+
| a | a |
+---+---+
| a | a |
+---+---+
| a | b |
+---+---+
| a | b |
+---+---+
| b | a |
```

```
R5;
+---+
| B |
+---+
| a |
+---+
| b |
+---+
| a |
+---+
```

```
R6;
+---+
| B |
+---+
| a |
+---+
| b |
+---+
+---+---+
| b | a |
+---+---+
| b | b |
+---+---+
| c | a |
+---+---+
| c | b |
+---+---+
| d | b |
+---+---+
| d | b |
+---+---+
| e | e |
+---+---+
```

Multiset division 1

```
R4 DIVISION R5;
+---+
| A |
+---+
| a |
+---+
| b |
+---+
```

Division 1

```
R4 SET DIVISION R5;
+---+
| A |
+---+
| a |
+---+
| b |
+---+
| c |
+---+
```

Multiset division 1

```
R4 DIVISION R6;
+---+
| A |
+---+
| a |
+---+
| a |
+---+
| b |
+---+
| c |
+---+
```

Division 2

```
R4 SET DIVISION R6;
+---+
| A |
+---+
| a |
+---+
| b |
+---+
| c |
+---+
```

The Cartesian product is straightforward. As this is not (really) a set operation, we have not provided a SET PRODUCT operation. You must apply the DISTINCT operation to remove duplicates from the Cartesian product.

Cartesian product

```
R6 PRODUCT R3;
+---+---+---+---+
| B | C | D | E |
+---+---+---+---+
| a | c | d | e |
+---+---+---+---+
| a | c | d | e |
+---+---+---+---+
| b | c | d | e |
+---+---+---+---+
| b | c | d | e |
+---+---+---+---+
```

Removing duplicates from a Cartesian product

```
DISTINCT(R6 PRODUCT R3);
+---+---+---+---+
| B | C | D | E |
+---+---+---+---+
| a | c | d | e |
+---+---+---+---+
| b | c | d | e |
+---+---+---+---+
```

Unary operators and multisets

All unary operations introduced in relational algebra do not eliminate duplicates. We will demonstrate this for the projection and selection.

Projection

```
PROJECT{B,C}(R6 PRODUCT R3);
+---+---+
| B | C |
+---+---+
| a | c |
+---+---+
| a | c |
+---+---+
| b | c |
+---+---+
| b | c |
+---+---+
```

Removing duplicates from a projection

```
DISTINCT(PROJECT{B,C}(R6 PRODUCT R3));
+---+---+
| B | C |
+---+---+
| a | c |
+---+---+
| b | c |
+---+---+
```

Selection	Removing duplicates from a selection

```
SELECT A="d" OR B="e" (R4);
+---+---+
| A | B |
+---+---+
| d | b |
+---+---+
| d | b |
+---+---+
| e | e |
+---+---+
```

```
DISTINCT(SELECT A="d" OR B="e" (R4));
+---+---+
| A | B |
+---+---+
| d | b |
+---+---+
| e | e |
+---+---+
```

Null values in selections

We can treat null values as a special value in select (and theta-join) conditions. To demonstrate this, we use the following relation:

```
R9;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
```

```
SELECT C = NULL (R9);
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
```

```
SELECT NOT (C <> NULL) (R9);
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
```

```
SELECT NOT (C = NULL) (R9);
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
```

Generalized projection

ERALI only supports arithmetic expressions on numeric values (INTEGER and DECIMAL). We use the following two relations:

```
R7;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           2 |           3 |
+-------------+-------------+-------------+
|           4 |           1 |           6 |
+-------------+-------------+-------------+
|           3 |           2 |           4 |
+-------------+-------------+-------------+
```

```
R8;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |        NULL |           3 |
+-------------+-------------+-------------+
|           4 |           1 |           6 |
+-------------+-------------+-------------+
|           3 |           2 |        NULL |
+-------------+-------------+-------------+
```

While the rename operation is still available, we can use the generalized projection to rename attributes. We can also use it to add constants to a relation.

```
PROJECT {A->FOO,B,C}(R7);
+---------------+-------------+-------------+
| FOO : INTEGER | B : INTEGER | C : INTEGER |
+---------------+-------------+-------------+
|             1 |           2 |           3 |
+---------------+-------------+-------------+
|             4 |           1 |           6 |
+---------------+-------------+-------------+
|             3 |           2 |           4 |
+---------------+-------------+-------------+
```

```
PROJECT {A,B,C,42->D}(R7);
+-------------+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER | D : INTEGER |
+-------------+-------------+-------------+-------------+
|           1 |           2 |           3 |          42 |
+-------------+-------------+-------------+-------------+
|           4 |           1 |           6 |          42 |
+-------------+-------------+-------------+-------------+
|           3 |           2 |           4 |          42 |
+-------------+-------------+-------------+-------------+
```

We can, of course, use arithmetic expressions in the generalized project. Notice that the computed value is NULL when one of its subexpressions yields NULL.

```
PROJECT {A,B,C,(A+B)*C->ABC}(R7);
+-------------+-------------+-------------+---------------+
| A : INTEGER | B : INTEGER | C : INTEGER | ABC : INTEGER |
+-------------+-------------+-------------+---------------+
|           1 |           2 |           3 |             9 |
+-------------+-------------+-------------+---------------+
|           4 |           1 |           6 |            30 |
+-------------+-------------+-------------+---------------+
|           3 |           2 |           4 |            20 |
+-------------+-------------+-------------+---------------+
```

```
PROJECT {A,B,C,(A+B)*C->ABC}(R8);
+-------------+-------------+-------------+---------------+
| A : INTEGER | B : INTEGER | C : INTEGER | ABC : INTEGER |
+-------------+-------------+-------------+---------------+
|           1 |        NULL |           3 |          NULL |
+-------------+-------------+-------------+---------------+
|           4 |           1 |           6 |            30 |
+-------------+-------------+-------------+---------------+
|           3 |           2 |        NULL |          NULL |
+-------------+-------------+-------------+---------------+
```

Grouping and aggregations

The following aggregations on numeric values (INTEGER and DECIMAL) are supported: AVG, SUM, MIN, MAX, and COUNT. We can aggregate the values of an expression on the whole relation or groups by partitioning the relation by one attribute. We can, in theory, group on multiple attributes (as supported by SQL), but this would require an extension of our formalism. This extension is relatively easy. ;-)

We use the following two relations:

```
R7;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           2 |           3 |
+-------------+-------------+-------------+
|           4 |           1 |           6 |
+-------------+-------------+-------------+
|           3 |           2 |           4 |
+-------------+-------------+-------------+
```

```
R8;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |        NULL |           3 |
+-------------+-------------+-------------+
|           4 |           1 |           6 |
+-------------+-------------+-------------+
|           3 |           2 |        NULL |
+-------------+-------------+-------------+
```

Aggregating expressions on a whole relation (left), but bear in mind that NULL values are ignored (middle). Aggregating expressions grouped by an attribute (right).

```
GROUP AVG(B)->X (R7);
+----------------------+
| X : DECIMAL          |
+----------------------+
|   1.6666666666666667 |
+----------------------+
```

```
GROUP AVG(B)->X (R8);
+----------------------+
| X : DECIMAL          |
+----------------------+
|                  1.5 |
+----------------------+
```

```
GROUP A, SUM(B*C)->X (R7 UNION R8);
+-------------+-------------+
| A : INTEGER | X : INTEGER |
+-------------+-------------+
|           1 |           6 |
+-------------+-------------+
|           3 |           8 |
+-------------+-------------+
|           4 |          12 |
+-------------+-------------+
```

Sorting relations

Sorting relations is straightforward. Given a list of attributes A1, … An, tuples are first sorted by A1, then by A2, and so forth until by An. All sorting is in ascending order. Null values are smaller than non-null values.

```
R9;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
```

```
SORT {A} (R9);
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
SORT {A,B} (R9);
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
```

```
SORT {B} (R9);
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
```

Natural joins

We can easily see how natural inner joins extend to multisets and null values. We will now demonstrate the syntax of outer joins, left outer joins, and right outer joins using the following two relations:

```
R10;
+-------------+-------------+
| A : INTEGER | B : INTEGER |
+-------------+-------------+
|           1 |           3 |
+-------------+-------------+
|           1 |           3 |
+-------------+-------------+
|           1 |           2 |
+-------------+-------------+
|           5 |           5 |
+-------------+-------------+
```

```
R11;
+-------------+-------------+
| B : INTEGER | C : INTEGER |
+-------------+-------------+
|           3 |           8 |
+-------------+-------------+
|           4 |           9 |
+-------------+-------------+
```

```
R10 JOIN R11;
+-------------+-------------+-------------+
| B : INTEGER | A : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           3 |           1 |           8 |
+-------------+-------------+-------------+
|           3 |           1 |           8 |
+-------------+-------------+-------------+
```

```
R10 LEFT OUTER JOIN R11;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
```

```
R10 RIGHT OUTER JOIN R11;
+-------------+-------------+-------------+
| B : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           3 |           3 |           8 |
+-------------+-------------+-------------+
|           3 |           3 |           8 |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
```

```
R10 OUTER JOIN R11;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | C : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           1 |           3 |           8 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
|        NULL |           4 |           9 |
+-------------+-------------+-------------+
```

Theta-joins

We can easily see how natural inner joins extend to multisets and null values. We will now demonstrate the syntax of outer joins, left outer joins, and right outer joins using the following two relations:

```
R10;
+-------------+-------------+
| A : INTEGER | B : INTEGER |
+-------------+-------------+
|           1 |           3 |
```

```
R12;
+-------------+
| D : INTEGER |
+-------------+
|           3 |
+-------------+-------------+
|           1 |           3 |
+-------------+-------------+
|           1 |           2 |
+-------------+-------------+
|           5 |           5 |
+-------------+-------------+
```

```
+-------------+
|           4 |
+-------------+
```

```
R10 JOIN B=D R12;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | D : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
```

```
R10 LEFT OUTER JOIN B=D R12;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | D : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
```

```
R10 RIGHT OUTER JOIN B=D R12;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | D : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
|        NULL |        NULL |           4 |
+-------------+-------------+-------------+
```

```
R10 OUTER JOIN B=D R12;
+-------------+-------------+-------------+
| A : INTEGER | B : INTEGER | D : INTEGER |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
|           1 |           3 |           3 |
+-------------+-------------+-------------+
|           1 |           2 |        NULL |
+-------------+-------------+-------------+
|           5 |           5 |        NULL |
+-------------+-------------+-------------+
|        NULL |        NULL |           4 |
+-------------+-------------+-------------+
```

Disclaimer

The introduction states that RALI and ERALI are prototypes that may contain bugs.

There are instances where the interpreter recognizes an error yet continues to evaluate a part of the query. For instance, ERALI's grammar does not support the following: `R10 FULL OUTER JOIN R11` as it should be `R10 OUTER JOIN R11`. ERALI "stops" at the unrecognized token "FULL" and evaluates the expression to the left.