CS252 Fundamentals of Relational Databases — Solutions for Worksheet 1
Getting Started with *Rel*

1. *No questions asked.*
2. *No questions asked.*
3. *No questions asked.*
4. *No questions asked.*
5. Why do we have to write `output x ;` in full when it is part of a compound statement, instead of just `x`?

    Because otherwise *Rel* would be looking at `x end ;` and that is not a valid statement of any kind. The presence of a line break carries no significance.

    What have you learned about *Rel's* rules concerning *case sensitivity*?

    Identifiers are case-sensitive, key words are not.

6. When "Enhanced" is off, is the output of evaluating the given relation literal identical to the input?

    No. The output includes `{CourseId CHAR, Name CHAR, StudentId CHAR}` in between the key word `RELATION` and the first opening brace. Also, the character string literals are enclosed in double-quotes instead of single-quotes.

    Now delete all the tuple expressions, leaving just `RELATION { }`. What happens when *Rel* tries to evaluate that?

    You get an error message saying that "{" is expected in place of <EOF>. In other words, it expects another list enclosed in braces to follow the empty one.

    Now use < to recall the original `RELATION` expression to the input pane and re-evaluate it with "Enhanced" *off*. Use copy-and-paste to copy the result to the input pane, then delete all the `TUPLE` expressions, to leave this:

```
RELATION {StudentId CHARACTER, CourseId CHARACTER,
          Name CHARACTER} { }
```

    Study the result of that in the output pane, first with "Enhanced" off, then with it on.

    What conclusions do you draw from all this, about *Rel* and **Tutorial D**?

    The text inserted after the key word `RELATION` can be recognized as a specification of the *heading* of the relation: a list of the attribute names and their declared types (in this example, `CHARACTER` for each attribute). **Tutorial D** allows the heading to be specified, in which case each tuple specified in the *body* must be of that heading. **Tutorial D** also allows the heading to be omitted, *provided that the body is not empty*. Each tuple must of course be of the same heading, and that determines the heading of the relation.

    Rel allows `CHARACTER` literals to be enclosed in either single-quotes or double-quotes. The closing quote must match the opening one.

    Next, enter the following literal, perhaps by using the < button to recall `enrolment` and editing it:

```
RELATION {
TUPLE { StudentId 'S1', CourseId 'C1', Name 'Anne'  },
TUPLE { StudentId 'S1', CourseId 'C1', Name 'Anne'  }
 }
```

Before you press Evaluate (F5), think about what you expect to happen. Does the result meet your expectation? How do you explain it?

> The body of a relation is a *set* of tuples. A set by definition contains exactly one appearance of each of its elements. *Rel* would perhaps be justified in treating this expression as an error, but it is equally justified in just ignoring any duplicate tuples. In conventional mathematical notation, {1,2,3,1}, for example, is considered to denote the set consisting of the elements 1, 2, and 3. The redundancy can sometimes be convenient when variables are involved—the set $\{x, y\}$, for example, has cardinality 1 in the case where $x=y$.

Use < again to recall the `enrolment` literal. Insert the word `WITH` at the beginning, add `AS enrolment : enrolment` at the end, and execute. How do you understand what you have just done?

> The `WITH` expression equates the name `enrolment` with the `RELATION` expression preceding the key word `AS`. It is the expression following the colon (:) that *Rel* evaluates. So in this simple case, `WITH` defines the name `enrolment`, and `enrolment` is then the expression we ask *Rel* to evaluate when we click on Run (F5).

By inspection of `enrolment` only, write down all the cases you can find of two students such that there is at least one course they are both enrolled on.

> Anne and Boris
> Boris and Devinder
> Anne and Devinder
>
> If you included all the cases where the two students are in fact the same student, such as "Anne and Anne", well, that's a good point—the question didn't say "*distinct* students".

7. How many distinct projections can be obtained from `enrolment`?

> Eight. If you found less than eight, did you forget the empty projection, `enrolment{}`? If you found more than eight, were you perhaps thinking that, for example, `enrolment{StudentId, Name}` and `enrolment{Name, StudentId}` are distinct projections? Recall that attribute order carries no significance.

8. Your renaming should look like this:

```
WITH RELATION {
TUPLE { StudentId 'S1', CourseId 'C1', Name 'Anne'  },
TUPLE { StudentId 'S1', CourseId 'C2', Name 'Anne'  },
TUPLE { StudentId 'S2', CourseId 'C1', Name 'Boris' },
TUPLE { StudentId 'S3', CourseId 'C3', Name 'Cindy' },
TUPLE { StudentId 'S4', CourseId 'C1', Name 'Devinder' }
} AS enrolment,
enrolment RENAME (StudentId as SID1, Name as N1 ) AS E1:
E1
```

9. Here is the expression you should have evaluated:

```
WITH RELATION {
TUPLE { StudentId 'S1', CourseId 'C1', Name 'Anne'  },
TUPLE { StudentId 'S1', CourseId 'C2', Name 'Anne'  },
TUPLE { StudentId 'S2', CourseId 'C1', Name 'Boris' },
TUPLE { StudentId 'S3', CourseId 'C3', Name 'Cindy' },
TUPLE { StudentId 'S4', CourseId 'C1', Name 'Devinder' }
} AS enrolment,
enrolment RENAME (StudentId as SID1, Name as N1 ) AS E1,
enrolment RENAME (StudentId as SID2, Name as N2 ) AS E2:
E1 JOIN E2
```

How do you interpret the result? How many tuples does it contain? Replace the key word `JOIN` by `COMPOSE`. How do you interpret *this* result? How many tuples are there now? How do you account for the difference?

The result of the join gives pairs of students, shown by their names and ids, enrolled on the same course, along with the course id of that course. There are 11 tuples, including several in which the two students are in fact the same person!

The result of the compose gives pairs of students such that there is at least one course they are both enrolled on. This time there are only 10 tuples, because Anne is enrolled on two courses and therefore appears twice, paired with herself, in the join but only once in the composition. (The composition is equivalent to the join followed by `{ ALL BUT CourseId }`).

10. Add `WHERE NOT ( SID1 = SID2 )` to end of the expression you evaluated in Step 9 (***see Lecture HACD.5, slides 3-6***). Examine the result closely. Now place parentheses around `E1 COMPOSE E2` and evaluate again. Confirm that you get the same result.

Repeat the experiment, replacing `WHERE NOT ( SID1 = SID2 )` by `{ SID1 }`. Do you get the same results this time? If not, why not?

What does all this tell you about operator precedence rules in *Rel*?

Because presence of parentheses around `e1 COMPOSE e2` makes no difference when that is followed by an invocation of `WHERE`, it appears that `COMPOSE` takes precedence over restriction. And so does `JOIN`. However, when we replace the restriction by a projection that specifies an attribute of `E1`, we find that it fails unless the `COMPOSE` invocation is enclosed in parentheses. We conclude that projection takes precedence over `COMPOSE` (and `JOIN`). *On the whole you are left to discover Rel's operator precedence rules for yourself. Of course you can always use parentheses to override them, as in most computer languages.*

Why was it probably a good idea to add that `WHERE` invocation? Does it completely solve the problem? If not, can you think of a better solution?

It eliminates the cases of the two students paired together being the very same student. However, we are still left with Anne being paired with Boris in one tuple, and Boris being paired with Anne in another tuple. Obviously if Anne and Boris are both enrolled on some course, we don't really want to be told so twice. It seems that the relation for the predicate "x is enrolled on the same course as y" is *reflexive* (true whenever $x = y$) and symmetric (if it is true when $x = a$ and $y = b$, then it is true when $x = b$ and $y = a$).

We can eliminate the redundant cases by using the `WHERE` condition `SID1 < SID2`, sneakily taking advantage of the fact that character strings are ordered (in **Rel**, as in most programming languages, of course).

What connection, if any, do you see between this exercise and Exercise 6?

See the last paragraph of Exercise 6.

11. *We value your comments. If you wrote some, thank you!*

End of Solutions