CS252 Fundamentals of Relational Databases — Worksheet 4

Constraints, Catalog, Virtual Relvars, and Types in *Rel*

If you get any unpleasant error messages you may wish to study the *Rel* forum at http://shark.armchair.mb.ca/~dave/relforum/index.php, in the section named Report a Bug.

1. Start up *Rel* . Instructions are at http://www.dcs.warwick.ac.uk/~acristea/courses/CS252/rel. You will be using the Suppliers-and-Parts database you set up for exercises in Worksheet 3.

2. Write **Tutorial D** integrity constraints for the suppliers-and-parts database to express the following requirements:

   a. Every shipment tuple must have a supplier number matching that of some supplier tuple.
   b. Every shipment tuple must have a part number matching that of some part tuple.
   c. All London suppliers must have status 20.
   d. No two suppliers can be located in the same city.
   e. At most one supplier can be located in Athens at any one time.
   f. There must exist at least one London supplier.
   g. The average supplier status must be at least 10.
   h. Every London supplier must be capable of supplying part P2.

   One of these (which one?) can be expressed as a `KEY` specification added to the relevant relvar definition. For the others, you will need to write a **Tutorial D** constraint definition, whose general format is:

   ```
   CONSTRAINT constraint-name Boolean-expression ;
   ```

   In each case use a Boolean expression of one of the following forms:

   Comparison of two invocations of `COUNT`

   ```
   IS_EMPTY ( relation-expression )
   ( relation-expression1 ) = ( relation-expression2 )
   ( relation-expression1 ) <= ( relation-expression2 )
   ```

   (The <= operator here means "is a subset of". The official **Tutorial D** symbol is ⊆ but *Rel* wisely chooses something you can easily type with a conventional keyboard.)

   Which of your constraint definitions are rejected by *Rel* for being `FALSE` at the time of definition? For any that are accepted, undefine them right away by using:

   ```
   DROP CONSTRAINT constraint-name ;
   ```

3. Explore *Rel*'s catalogue. It consists of a relvar named `sys.Catalog`. Use the following trick to see `sys.Catalog`'s heading only:

   ```
   sys.Catalog WHERE FALSE
   ```

   From their names, you might be able to guess which attributes are of most interest (possibly `Name, Owner,` and `isVirtual`?).

   Create a virtual relvar named `myvars` giving the `Name, Owner,` and `isVirtual` of every relvar not owned by 'Rel'. Virtual relvars are created like this:

   ```
   VAR name VIRTUAL relation-expression ;
   ```

   Test your virtual relvar definition by entering the queries

```
myvars
myvars WHERE isVirtual
(myvars WHERE NOT isVirtual){ALL BUT isVirtual}
```

4. Load /local/java/Rel/Scripts/OperatorsChar.d into *Rel*'s input pane and execute that script. As a result several useful user-defined operators will be available to you. One of the relvars mentioned in sys.Catalog is named sys.Operators. Display the contents of that relvar. How many attributes does it have? What is the declared type of the attribute named `Implementations`?

5. Evaluate the expression

```
(sys.Operators ungroup (Implementations)
where Language = 'JavaF')
{ ALL BUT Language, CreatedByType, Owner, CreationSequence}
```

   What are the "ReturnsTypes" of `LENGTH`, `IS_DIGITS`, and `SUBSTRING`?

6. Note that if *s* is a value of type `CHAR`, then `LENGTH`(*s*) gives the number of characters in *s,* `IS_DIGITS`(*s*) gives `TRUE` if and only if every character of *s* is a decimal digit. `SUBSTRING`(*s*,0,l) gives the string consisting of the first *l* characters of *s* (note that strings are considered to start at position 0, not 1). `SUBSTRING`(*s*,*f*) gives the string consisting of all the characters of *s* from position *f* to the end.

   What is the result of `IS_DIGITS('')`? Is it what you expected? Is it consistent with the definition given above?

7. Using operators defined by OperatorsChar.d, define types for supplier numbers and part numbers, taking the example shown in lecture HACD.2, Slides 14 and 15. *Note:* Those two slides were revised *after* the 2009 presentation of that lecture.

   Define relvars `Srev`, `Prev`, and `SPrev` as replacements for `S`, `P` and `SP`, using the types you have just defined as the declared types of attributes `S#` and `P#`.

   Write relvar assignments to copy the contents of `S`, `P` and `SP` to `Srev`, `Prev`, and `SPrev`, respectively. Note that if `SNO` is the type name for supplier numbers in `S` and `Srev`, then `SNO(S#)` "converts" an `S#` value in `S` to one for use in `Srev`.

End of worksheet