Date: Tue, 24 Jan 2012 15:44:14 +0000
From: Hugh Darwen <hughdarwen@gmail.com>
To: TTM mailing list <ttm@thethirdmanifesto.com>
Subject: [TTM] A joyful experience with Rel

I hope those who agree with **the failure by most SQL implementations to support CREATE ASSERTION** (e.g., Toon Koppelaars) will read this one.

I have constructed a new database in Rel in connection with my strange hobby exhibited at my website www.doubledummy.net, which is all about a certain kind of bridge problem. I invite people to contribute composed problems to my web site and I invite people to send me solutions to those problems. Each problem is rated according to its difficulty, on a scale of 1 to 8, and that's the number of points a successful solver gets added to their life-time record.

I'll spare you a description of the entire database, but I'd like to tell you about a couple of constraints I declared with some glee.

Although nearly every problem (identified by a unique problem number) is composed by a single person, every now and again I get a joint composition. So I can't record the composer in the Problem relvar--I have to have a separate ComposedBy relvar, which is all-key. Another all-key relvar, SolvedBy, records all correct solutions.

Obviously nobody can be both solver and composer of the same problem, so I was pleased to be able to write CONSTRAINT SolverNotComposer IS_EMPTY(Solvedby MATCHING ComposedBy). **(You can't do that in SQL implementations that don't support CREATE ASSERTION or "subqueries in table constraints".)**

With that constraint in place everything was fine until one day I got an unexpected violation of SolverNotComposer. It turned out that I had accidentally inserted a second tuple for the same problem into ComposedBy, even though that problem had only one composer. **I had forgotten to change the problem number when I did the insert.** This was very annoying. If I had restricted each problem to have just one composer, the obvious key constraint would have prevented me from making that mistake.

Here's the **workaround** I came up with. I added an attribute, NumberOfComposers, to the Problem relvar. Its value is nearly always 1, of course and so far I have no problems composed by more than two people. Then I declared this:

CONSTRAINT RightNumberOfComposers IS_EMPTY (Problem NOT MATCHING SUMMARIZE ComposedBy BY { Problem# } ADD (COUNT ( ) as NumberOfComposers));

(The syntax would be slightly different in **Tutorial D** Version 2.)

This is a useful application of redundancy that hasn't occurred to me before. I'm happy to report that the use of aggregation in a constraint has not yet significantly slowed things down for me.

The experience makes me hold even more strongly to the view that SQL implementations should support such constraints. I know they aren't scalable but the first one could be supported by a special shorthand, say an "anti foreign key", to help the optimiser check it on a delta basis. In any case, the solutions using CREATE TRIGGER, given in Advance Mathematics for Database Professionals by Toon Koppelaars and the late Lex de Haan, would remain available for when performance considerations militate against declared constraints.

Hugh