To: ttm@thethirdmanifesto.com
From: Derek Asirvadem <derek.asirvadem@gmail.com>
Subject: SQL is not "Broken", the Detractors Are
Cc:
Bcc:

> From: Derek Asirvadem <derek.asirvadem@gmail.com>
>
> > From: "erwin@EDPNet" <e.smout@edpnet.be>
> >
> > > On Wed, 01 Feb 2012 16:03:12 +0100, Derek Asirvadem
> > > <derek.asirvadem@gmail.com> wrote:
> > >
> > > > I'm guessing you can fix the name-resolution conflict around the column
> > > > names somehow.
> > >
> > > Sorry. What name resolution conflict ? I don't have (never had) a name
> > > resolution conflict. Please identify. I will fix it immediately and publicly.
> >
> > I have a vague idea that Toon was referring to that @ he wrote in
> >
> > WHERE ProblemId = @ ProblemId
> >
> > You need a way to make one of the PROBLEMID's reference the column of the table that
> > the constraint is on, instead of the PROBLEMID column of the table that the SELECT is
> > on. A bit akin to how you can reference columns of an outer select in correlated
> > subqueries, only, alas, in this case there just isn't any outer select.
>
> Oh, I see. thanks for explaining.
>
> No, I don't have to do any of that mundane thinking, that any programmer can do for me if
> he had his eyes and ears open, and at least one coffee. Sybase does that automatically. Has
> done for years. There are many other places where it resolves what you have to ploddingly
> type as "aliases" quite automatically. It has a capability called "context", it can figure out
> the difference between "inner" and outer" all by itself. Just one of the reasons it is precious
> to me. There are well over a thousand.
>
> I don't have time now, but when I get time tonight, I will post the results of actual tested
> code. Even though it is identical to at least 110 functions that I have running in production.

Right. The syntax is really for Toon's benefit, or anyone else genuinely interested in the use of Functions to overcome the absence of complex checking in the SQL syntax (since the committe absurdly demand such specifically in the syntax, which you have confirmed).

The testing is for those on the list who are addicted to questioning the veracity of my statements, and attacking things they do not know. Not that I expect them to curtail their addiction in the future. I enjoy enough credibility in my occupation (elsewhere in the industry) such that the attacks on this list (which is isolated from the industry) have little effect. Same as if you call me a serial killer or paedophile.

So in order to provide some context, let's restate my point. or points (since the related posts have covered a few related topics).

1. This is in response to a number of people on this list who keep stating things about what SQL can, and cannot do, which implies a knowledge of SQL, of which they evidently (ie. by virtue of the evidence of their own posts) have none. Zero. Zilch. Nada.

   • They may have some knowledge of the SQL syntax dictate, from the SQL Committee, or from reading the spec, as you recently have, since you wave it around like a flag and use it to attack real things.

   • the SQL spec is unreal, it does not exist in the real world. It has no value, or its value in the real world is somewhere between toilet paper used to blow my nose, and toilet paper used to dry my armpits. Notice that I did not mention the toilet paper used to wipe my bottom.

2. Anyone who thinks that SQL, ie, an implementation of the SQL spec by a vendor, and the millions of

databases and SQL code segments that have been used to support the IT needs of millions of corporations, is the same as the SQL Spec, has more than seven holes in their head.

> • What is plain to any technician (not academics, they love it and wave it around as if has value), and I have posted a fair amount about it, is that the Spec cannot be implemented, to do anything except make people cry, and give them sweaty armpits. No, vendors have technicians, and technicians have brains, as do the customers who buy SQL implementations, and their coders who use it; the SQL implementations do somewhat more than that.
>
> > • the fundamental requirement was to "make the SQL Spec work", and thus overcome the double-binds and idiocy in the SQL Spec (which was produced by non-technicians, fighting and politicking on the Committe).
> >
> > > • Chief among that was a stable and sensible implementation of Null handling. Going on and on about The Null Problem which SQL does not have, which the SQL Spec has, and therefore the high end SQLs simply do not have, and the low-end SQLs have to varying degrees (non-technical implementors, separated by geography, allegedly "collaborating", as they do, to form yesterdays version of wikipedia).
> >
> > • In addition to providing a solid working SQL that technicians can use, the vendors technicians have gone further, and implemented the extensions to the Relational Model (apart from the notion of RM discussed on this list), faithful to the spirit and intent of Codd.
>
> • I make this statement after the scores of arguments that have been had on this list, and resolved, which while being ostensibly about something else, have turned out to be about this: the people on this list who make statements about SQL, are profoundly clueless about SQL. They mean their statements about the SQL Spec, only.
>
> • henceforth, if we are going to be honest on this list (ie. discontinue making dishonest allegations and statements about SQL, when the statements actually apply to the SQL Spec, only):
>
> > • such people should clearly make their statements about the SQL Spec (used toilet paper, of no value to anyone in the real world).
> >
> > • only people who actually have experience with an high end commercial implementation of SQL are qualified to make statements about what SQL is, does, and does not do.
>
> ---
>
> The low-end SQLs, the freeware non-SQLs fraudulently masquerading as SQL, are all excluded. No one purchasing software for a corporation that is going to be around for more than a few years cares what 5,000 imbeciles across the planet code, and badge as "SQL". If the list is supposed to be relevant, then we should not care either.
>
> ---
>
> • we are now well aware that what the SQL Spec does or does not do, has no relevance whatsoever to the SQL supplied by vendors, and used by technicans, to build systems in the real world. Soiled toilet paper.
>
> • continued berating about the SQL Spec is a waste of the life force that God gave you, henceforth you will have to obtain energy for that from the devil, and only devil worshippers will hang around and listen, the believers will walk away..
>
> • Likewise, comments about vendors being "stupid" because they have not implemented some absurd facet of SQL Spec syntax (when they have implemented the feature in another form, which is not known to you), or that they "still cannot implement" some feature or syntax, are grossly incorrect. Such statements merely prove your ignorance of SQL.

3. I made the statement that in normal modern SQLs, we do not have many of teh problems or limitations that are suggested on this list. In particular, various forms of constraints (that are either not provided by the SQL syntax or the SQL Spec syntax (hilariously trying to)

• can be implemented as `CHECK CONSTRAINTS` which call a Function.

• while I have used this technique for years (decades ?), and have hundreds implemented in databases that run in production (meaning, real world, income dependent, for years, as intented, without error), I mostly use them for honest Constraint or constraint-like checking

• I had not previously used them for limiting the count of a relation. Noting that, I supplied code that was practically (for all intents and purposes) identical to the 110+ I had actually tested, that were in production, but with the caveat that the supplied code was untested.

4. Let's take it from the beginning

> On Tue, Jan 24, 2012 at 4:44 PM, Hugh Darwen <hughdarwen@gmail.com> wrote:
>
> I hope those who agree with the failure by most SQL implementations to support CREATE ASSERTION (e.g., Toon Koppelaars) will read this one.
>
> [...]
>
> Here's the workaround I came up with. [...] Then I declared this:
>
> CONSTRAINT RightNumberOfComposers IS_EMPTY (Problem NOT MATCHING SUMMARIZE ComposedBy BY { Problem# } ADD (COUNT ( ) as NumberOfComposers));

To which various compliant posters posted various extremely long tricks, tricks of tricks, tricks for the tricks that didn't do the trick, workarounds, workarounds for the workarounds that didn't work, workarounds for the tricks, ad nauseum.

None of that is necessary. The RM works. SQL works. No tricks. No workarounds. No Views. No triggers. Simple Declarative Constraints.

• In reality, no code. But the pedants out there will attack that, so I will state: No program or procedural code (leaving only "code" that supports declarations).

• Another way to say that is, as identified in the paper waving that you have being doing for us (thank you), it is an implementation of the spirit and intent of the SQL-92 Spec, by vendors with more brains than the entire Committee, overcoming the impossibilities that the latter have specified, by separating DDL and Declarative Code, and allowing for the requirements of each, which the Committee were oblivious to. The "@".

• We now know that Hugh and other RM/SQL detractors here, do not know SQL, as evidenced.

5. The first part of the solution is that the problem itself was a problem, relegating it therefore to toyland; so it order to give the problem some worth, I treated it like a real problem. The error was in the the model; and since any code consequent to a non-database would be in gross error, I fixed the model first, by simply following the RM, and modelling the data correctly:

We started with a Problem, that has a single Composer. Basic modelling mistake. It real life, the Composer exists before the Problem is created, and the Problem cannot exist without a Composer. Therefore:

• the Person table exists first

• Composer and Solver are not Entities; they are Roles of the Person Entity (unless you want to break more rules and duplicate the same data in two separate tables).

• Problem exists as a child of Person, with an FK to Person (in their Role as Composer)

• Redundant columns and those which break any NF are not allowed; they are certainly not needed (I will tell you a golden rule: if you ever find yourself using code to workaround the "absence" of a Rule, go back to the model and define the rule instead; if you ever find yourself adding columns to support a Rule, stop, go back to modelling, because it is not

finishe).

Corrected Data Model

• Now we have a worthwhile problem, the solution for which can be coded with confidence (the initially stated problem couldn't).

• [...] therefore, this solution will allow one Composer only. And if, and when, there appears a Problem that is dually composed, we will change the database, to allow many Composers. And if the limit is 2 or 66 simply implement that in a simple `CHECK CONSTRAINTS` (which calls a simple a Function). And then, not against a redundant column value.

> • Many people (technicians as well as business users) call all these "Business Rules". The data model, which identifies the entires database and rules, is first and foremost Normalised as per the RM; then all basic structural Ruels are implemented as Keys and Relations; the remainder as `CHECK CONSTRAINTS`
>
> • it is not only unnecessary, it breaks various Rules (RM, Open Architecture, etc), to use code outside the database to implement these Business Rules
>
> • The term `CHECK CONSTRAINT` must be used because the non-technicians on the SQL Committee have not figured out the syntax for it yet. Meanwhile, back on this planet, in the technicians universe, we've implemented Rules as `CHECK CONSTRAINTS`. They haven't figured out yet that there Good Reasons for the ancient boundaries between definition, data and code.

• Solution is of course an associative table between Person and Problem, where the Person has the Role of Solver.

• The full DDL and SQL as per the Model, for the database relevant to testing; the tables; the Constraints; the Rules; the Functions that support them; Test beds. I have arranged them in presentation sequence, rather than by boundary, hopefully to reduce complication:

Definition, Code, and Test Data

• Notice the absence of @ and "aliases" in the declarations. Notice the @ in the Function definitions. Notice there are no "name resolution conflicts" or whatever it is that heappens in the SQL Spec and the free-ware.

• Please feel free to implement and test in the SQL of your choice. As long as it is commercial, and post say, 1995, in order to catch SQL-92, you have a good chance.

> • If it is free-ware or "collaboratively developed" or written up in wikipedia, don't waste your time.
>
> • PostGre-non-SQL is particularly dishonest in this area, since it heralds toilet paper on absolutely every page of the manuals, uses it (drowning, not waving) as The Excuse for implementing insane behaviour. So much for verbatim implementors.

• As per other posts/discussions on the "issue", I have limited the Solutions to four per Problem, using a simple Rule.

• Proof that the stupid thing works, for the infidels (the technical term teh faithful use to describe non-believers). First the **Solver_Max_4 Rule** supported by a single-command Function:

Result Set 1
Result Set 2
Error Message

• The discussion that in SQL, a Rule cannot be implemented to exclude Composers from Solutions is also simply incorrect. I implemented a simple **Solver_Exclude_Composer Rule** supported by a single-command Function. Proof that it works:

Error Message

• Of course these Rules, and others like these, implemented as Constraints-calling-Functions, as Toon immediately recognised, are semantically the same as the contents of the Function implemented in the constraint ... the limitation is in the SQL Spec syntax. If your vendor has enough technicians, that problem will have been overcome. Oracle has no technicians.

> On Tue, Jan 24, 2012 at 4:44 PM, Hugh Darwen <hughdarwen@gmail.com> wrote:
>
> > This [storage of a Count column] is a useful application of redundancy that hasn't occurred to me before.

There is no such thing as a "useful" application of redundancy, there never is. It breaks the Rules; the RM; the NFs. It introduces an update anomaly where there was none. And of course, these statements are doubly more sober, because such a column is (a) not necessary and (b) only required for people who do not follow the precribed sequence: model the data according to the RM first; then write code. The mistake you made is typical of the millions of new-style developers: they think in terms of the problem that needs to be solved, and they solve it using a spreadsheet and code that they can understand easily; they do not know about modelling the data first, which eliminates the spreadsheet.

> I'm happy to report that the use of aggregation in a constraint has not yet significantly slowed things down for me.

It is zero overhead for us. The over 110 functions such as this that I mention are running in large banking systems, under race conditions, suffering contention on the data objects (not the code, it is pure code that does not contend). I will not be running a stress test for the non-believers: they are in that state made famous by Arthur C Clarke:

Any sufficiently advanced technology is indistinguishable from magic.

After a year or three of getting used to it, they will not need to stress-test it either.

> The experience makes me hold even more strongly to the view that SQL implementations should support such constraints.

It already does (sans the syntax). has for years.

> 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.

It is implemented as pure code, stored with, and operates exactly the same as, stored procs in Sybase and DB2, meaning it already scales with no limit. No need for triggers or whatever is in the referenced volumes. (Not a comment re Kooplaars & De Haan's work.)

• Oracle stored procs are not pure code; nor do they execute within the server (because such does not exist), they execute in client or consumer process space, each a private copy, like the old `COBOL` programs. (No offence against `COBOL`.)

For summarised data, or derived data of any form (which I believe the referenced volumes regard, re specialised cases), I never store them or use triggers or materialised views. I use only derived tables and views, meaning that the summary values are constructed at run time. Never stored. Since I have used both for decades (many databases have them in stored form, at varying degrees of efficiency), I can tell you with confidence, the unstored form blows the doors off the stored forms, by orders of magnitude. The key is in the data model.

• If anyone is interested, I have just completed a little project for one of these, allowing pivots as well as the normal summaries, they all perform at the same blinding speed. I have some benchmarkes set up, so it will not be too much work to fire up.

If and when the Sybase optimiser needs help, we will address Her. Right now, the "overhead" is so small it is difficult to detect, almost invisble.

• I appreciate that the cheap and nasty "optimisers" in the freeware end of town leave a lot to

> be desired (Oracle cacks itself; PostGre-non-SQL snorts and baulks at, subqueries). SO in those cases, sure, one has to write more code, to get the beast over the fence ... but that is not an SQL performance problem, or a declared constraint problem, it clearly is an implementation problem. You always get what you pay for.

Regards
Derek Asirvadem
Faithful RM Stalwart, Standard-compliant Practitioner
http://www.softwaregems.com.au