SQL for Stored and Inherited Relations[1]

Witold Litwin
Université Paris-Dauphine PSL
Paris, France
Witold.Litwin@dauphine.fr

Keywords: SQL Databases, Information Systems, Non-procedural Data Definition and Manipulation, Logical Navigation

Abstract: A stored and inherited relation (SIR) is a stored relation (SR) extended with inherited attributes (IAs) calculated as in a view. Without affecting the normal form of the SR, IAs can make queries free of logical navigation or of value expressions. A view of the SR can do the same. The virtual (dynamic, computed…) attributes (VAs) possibly extending SRs at major DBSs, can do as well for value expressions defining them. VAs are less procedural to declare than any alternate view. Likewise, altering any attribute of an SR with VAs leading to view altering otherwise is less procedural. We propose extensions to SQL generalizing the latter two properties to every IA. In particular, our proposal is backward compatible with the creation and altering of Vas at present. We motivate our proposals with the biblical Supplier-Part DB. We show how to implement SIRs with negligible operational overhead. We postulate SIRs standard on SQL DBSs.

[11] This e-report complements at present the successive references to its content within the article published at 21st Intl. Conf. on Enterprise Inf. Syst. (ICEIS 2019), pdf .

1. Default naming in a SIR

For every SIR R, its base has its proper *default* relation name that was assumed to be R_, we recall. As every relation name, every default name should be unique in the DB. If a base name R_ conflicts, DBA should rename either SIR R or the conflicting relation: SIR R_ or SR R_ or view R_. In practice, we presume DBA free to choose any default base naming rule. Next, every SIR we consider is an SQL relation implicitly. Furthermore, we suppose every SQL naming rule applying to SIRs. For every SIR R in particular, one may qualify every SA or IA A as R.A. In addition one may qualify every SA A of every SIR R as R_.A as well. The latter option is the default as well.

Recall furthermore that our rule for default source naming of SAs in SIR R, keeps all the clauses From… within C-view R referring there to R_, valid for SIR R as well. Instead of referring to stand-alone SR R_, i.e., defined by dedicated Create Table R, they simply refer to the base of SIR R. That one is equal to the former as an SQL relation and with respect to the full attribute naming. This whole property is the primary rationale for our specific to SIRs naming rule. One can figure out however also a less obvious one. Namely, referring to R_ rather than to R whenever possible, from some other SIR R', when R already inherits an IA from R', hence refers to R', may avoid the *circular referencing* between R and R'. We prohibit it, as it is for views. Referring to R_ may avoid such referencing since every R_ inherits from nothing by definition.

2. Procedurality Measures

We recall that the simplest measure of procedurality *gain (reduction)* is p_1/p_2. E_{SP} appears then 1.29 times less procedural than Create View SP in (1). Notice furthermore that one could alternatively prefer $(p_1\text{-}p_2) / p_1$ as measure. This would show that E_{SP} saves 22% of procedurality of (1). Finally, one could rather measure the procedurality *overhead* $(p_1/p_2 - 1)$. This would indicate 30% overhead of (1) over E_{SP} here.

3. Q-views

We recall that first the reduced procedurality of an I_R may result from new generic character we denote as '#'. There is always I_R with '#' less procedural than C-view R in form of Select * From…., unlike perhaps any E_R. Secondly, an I_R may also have element(s) (), each forming brackets around some SAs in Create Table R.

As we discussed, unlike perhaps any E_R, there is then always I_R less procedural than Create View of a view we called *query equivalent* to SIR R, Q-view in short. It is not C-view R, but an equivalent one that still provides in practice for the same non-procedural queries as C-view R, hence as SIR R. "In practice" means here that the query does not (uselessly) prefix unambiguous proper attribute names, as discussed in the Introduction. When Q-view R is a possibility, without alternate IE less procedural than Create View R, a DBA could reasonably prefer Create Table R_ and Create View R to Create Table R for SIR R.

More in depth, under some restrictive conditions on the DB, Q-view R may happen to be the same SQL relation as C-view R and SIR R, except for different source name(s) for some unique proper SA name(s) in SIR R. If so, whether an SQL query to C-view R or SIR R, or to Q-view R selects every such an attribute by its full source name in the view or SIR R, or by its proper name only, the result at every popular DBS, will have every such attribute labelled with its proper name only. Every possible result of a query to C-view R or to SIR R may then also result either from the same query to Q-view R or from the same query, except that every discussed attribute there would bear its proper name only.

In the same time, Create View for Q-view R may be substantially less procedural than the least procedural for any C-view R. It may become then even less procedural than E_R could ever be. The rationale is that Q-view R scheme may take advantage of *. This may especially occur when (i) for some relation F among those in From clause and different of R and of R_, (F standing for "foreign" relation thus), IE and C-view R inherit every attribute of F in the SQL order, except for some attribute F.A, perhaps composed in some order, (ii) For every such F, SIR R also has an SA R_.A identically composed, (iii) for every tuple of SIR R, R_.A has the same value as this in F.A if the latter was inherited within the tuple and (iv) in SIR R, as well as in C-view thus, for every F, all the attributes inherited from F and R_.A are in the order of F attributes. For every K, Q-view R may then have F.A instead of R_.A. For Q-view R, for every F, F.* may suffice then. In contrast, even the least procedural E_R has to name for every F all the IAs from. Thus cannot take advantage from '*'. Every possible E_R could then turn out more procedural than Q-view R. Contradicting our claim and making perhaps DBA again reasonably preferring SR R_ and Q-view R to SIR R.

One may easily verify also from Ex. 3 in the main paper that lower procedurality of I_{SP} generalizes to any multi-attribute A, common to some Ri and R_. It also generalizes to any I_R conform to Rule 2, regardless of the number of constructs Ri.* in the Select list and of relation and attribute names. Observe finally, that there is no Q-view R whose Create View is less procedural than the one of C-view R, hence, perhaps than E_R as well, other than when Rule 2 suffices.

Besides, a *natural* SIR has the scheme with all foreign keys parenthesed. It is easy to see that for every natural SIR R, its I_R with only necessary characters, say I^N_R, is the minimal one, i.e., the least procedural, with respect to every I_R of SIR R differing from the natural one by the attribute order only. According to everything already said, the procedurality of I^N_R

should be also lower than of Create View R for every Q-view R. The gain should be even greater for any C-view R.

E.g., for our example above, I^N_{SP} is I_{SP} that (7) defines except every unnecessary space there. I^N_{SP} is then clearly the minimal I_{SP} or E_{SP} of any SIR SP differing from (6) by the order of attributes only. Then, Create View for Q-view SP as in (11), is almost two times more procedural than I^N_{SP}. Likewise, Create View for C-view SP as in (11) is at best more than 2.5 times more procedural. Next, even the savviest C-view SP, mandatory when Q-view SP (11) is not possible, with outer joins thus, is by far less attractive. Create View SP becomes indeed then at best more than three times more procedural than I^N_{SP}. Finally, our example Create Table SP in (2), with every unnecessary space removed, reveals 1.32 more procedural than the similar Create Table SP for the natural SP in (6). Altogether, we may expect natural SIRs to be (naturally) the primary choice for DBAs.

4. More on Rule 3

For examples for Rule 3 that follow, formally, we see the statement with VAs as creating SIR R where I_R (i) consists of all and only VA-clauses defining each IA and (ii) has not From clause. I_R expresses then E^I_R where, (i) for every VA A, there is IA A declared: V As A and no other IA is declared, (ii) From clause is: From R_. This equivalence is the reason why to define SR R with VAs may be seen as creating a specific SIR R, unknowingly of course and since decades. Rule 3 means then simply that no such SIR R is pre-processed to E^I_R.

Observe nevertheless that E^I_R is in fact E_R of SIR R one could create as less procedural option than SR R_ and C-view R, where the latter is formally the same relation as SR R with VAs. Such a view is always a possibility instead of any SR with the VAs. But, it is precisely what the latter choice avoids, we recall. One could nonetheless always preprocess I_R to E^I_R, instead of creating SR R with VA. The client does not see indeed how SIR DBS actually deals with the statement. DBS would then further process E^I_R as any E_R, as we detail later. Rule 3 as stated seems nevertheless a more practical choice. In contrast, if the kernel does not provide for VAs, Rule 3 does not apply. SIR R with E^I_R is mandatory with, as always, E^I_R remaining less procedural than creating C-view R.

Ex. Suppose that S-P2.P.WEIGHT provides for the weight of every part in pounds, while the clients should also know the weight in KG. This, as the attribute named WEIGHT_KG and defined below, right after WEIGHT in P.

1. Suppose MS Access as the kernel dialect. Rule 3 does not apply then. E_P is the only option, e.g. we have,

(12) E_P = WEIGHT_KG AS Round (WEIGHT*0.454,3) From P;

The WEIGHT_KG scheme should be in Create Table P immediately after SA WEIGHT scheme. From P clause in (12) should follow SA CITY. It is easy to see that E_P is less procedural than would be any Create View for C-view P. More precisely, with respect to SQL Server's SQL, E_p is 2.52 times less procedural than C-view P, assuming again only necessary spacing in both expressions.

2. Suppose now S-P2 on SQL Server. WEIGHT_KG could be a VA. Rule 3 allows declaring WEIGHT_KG as:

(13) WEIGHT_KG As Round (WEIGHT * 0.454,3);

This declaration constitutes I_P. It is clearly even less procedural than (12). Namely, for SQL Server as kernel, the gain would increase to 2.94 times. Through Rule 3, Create Table P resulting from (13) would be assimilated to SQL Server's statement creating SR P with SAs as at Figure 2 and with VA WEIGHT_KG defined by (13). In the same time, the result would be SIR P with I_P defined by (13) and (12) as E^I_P. The procedurality reduction of almost three times would be clearly of practical interest. It illustrates well why all major general-purpose relational DBSs (Oracle, SQL Server, MySQL, DB2) offered VAs. To a slightly lesser extent, the 2.52 times gain offered by E_P (12) over C-view P being the only current option on MsAccess, remains nevertheless also substantial,.

3. Suppose now still for S-P2 at SQL Server SQL as the kernel, that P should have not only WEIGHT_KG AS conceptual attribute, but, also, after CITY, should have one named T_ QTY, with the total weight of the supplies of this part. E.g., in order to foresee the requirements on the warehouse with the supplies.

As SA, T_ QTY would require impractically frequent updates and a highly procedural TRIGGER statement on SP. As IA then, T_QTY would clearly need V with an aggregate function. T_QTY cannot be then VA for SQL Server, neither for any DBS we are aware of. Rule 3 leaves E_P the only choice for SIR P. With WEIGHT_KG defined by (13), that one would be:

E_p = WEIGHT_KG…, T_Weight AS WEIGHT * (Select Sum (QTY) From SP Where SP.p# = SP.p#) FROM P_;@

Summing up, with Rule 3, creation of SIR R is never more procedural than some SQL alternate capability available at present for the same relation R or at least formally the same.

Our work here aims only at SQL clauses for SIRs providing the non-procedurality necessary and sufficient for our goal. Observe however, perhaps as further work, that there are ways to decrease IE procedurality further at the expense of additional processing. One easy way is to observe that foreign keys are usually mono-attribute. Hence if A is such key, instead of writing (A) in IE, one could rather write, say, A!. This would save one character for each key. Likewise, one can generalize Rule 3. Namely, every SIR R with every IA A resulting from VE V, could be declared through I_R with every A stated as for a view, i.e., in the form of 'V As A' and with

implicit From R. This, regardless of whether every A concerned could be VA at present. Again, at expense of more processing, there would be procedurality gain for some kernels, e.g., MySQL. Also, one could avoid From R_ clause of E^I_R, necessary with current Rule 3.

4. Other DDL Statements for SIRs

The SQL DDL statements we suppose for SIRs beyond Create Table and Alter Table are all the other popular ones, i.e., Drop Table, Alter View, Drop View and Create Index.

For Drop Table R, we simply consider it applying also to every SIR R. As usual, one should not violate the referential integrity. Likewise, the statement may cascade or may get refused. Notice that dropping SIR R is substantially less procedural than dropping SR R_ with any equivalent view R. Single Drop Table R replaces indeed here Drop View R and Drop Table R_. This makes the procedurality gain in favor of the former already of about two. In addition, unless the DB is private to DBA, these two statements should form an atomic SQL transaction. That one requires even more statements we recall later in Ex. 6, increasing the gain several times consequently.

Next, we suppose Alter View and Drop View as in the kernel. Observe then that dropping the IE in SIR R is consequently substantially less procedural than dropping C-view R. The former requires indeed only Alter Table R Drop IE statement. The latter typically requires again an atomic transaction with Drop View R and Alter Table R_ Rename to R. Interestingly, to drop VAs in contrast, may be more procedural than to drop C-view. One has to drop those individually indeed, in every kernel's Alter Table we are aware of. Still, the drawback apparently did not affect the popularity of VAs. We thus can hope the same for the SIRs. Otherwise, the additional clause Drop IE in Alter Table R could obviously solve the issue.

Finally, we suppose for Create Index for SAs or IAs the syntax of the kernel one for SAs, VAs and views.