Stored and Inherited Relations with Natural or Declared Foreign Keys

Witold Litwin
Université Paris Dauphine France
Witold.litwin@dauphine.fr

Abstract — A stored and inherited relation (SIR) is 1NF stored relation enlarged with inherited attributes (IAs). The latter make SIRs the only known view-savers for logical navigation free (LNF) or calculated attribute free (CAF) queries, without any denormalization. Recall that LN means joins among base tables, while calculated attributes serve as the virtual ones do at some popular DBSs, but can be more general, e.g., with aggregate functions or sub-queries. The overall advantage of SIRs is substantially less procedural queries and DB schemes. We now show that usual schemes of stored relations with foreign keys implicitly define natural SIRs. The major advantage of the latter is the LNF queries with zero procedurality to define the IAs. We first discuss in depth the natural SIRs and show rules inferring the IAs from the foreign keys. We then extend these rules to SIRs with explicit IAs, the calculated ones especially. We generalize the formal relational design to SIRs, including the Normal Forms. We show that generalizing a typical relational DBS to SIR-enabled one should be simple. Preexisting applications could remain not affected, while new ones could profit from LNF and CAF queries. We conclude that major relational DBSs should evolve to SIR-enabled “better sooner than later”. To make LNF and CAF queries standard, at last.

Keywords—Relational model, Foreign Key, Inheritance, Logical Navigation, SQL, Stored and Inherited Relation

I INTRODUCTION

The relational model as defined by Codd has two 1NF constructs (abstractions), [6], [7]. A stored relation (SR), also called base relation or table, consists of stored attributes, (SAs), only. Values of these attributes are not calculable from other attributes in the DB (that is why they have to be stored). An inherited relation, more commonly called view or view table, consists of (relationally) inherited attributes, (IAs), only. One calculates every IA from SAs or other IAs, through a stored (relational) query called view scheme. Originally, one supposed every IA calculable only. Later, it appeared practical sometimes to maintain a (stored) snapshot of selected IAs, refreshed whenever needed. Such views and IAs were termed materialized, [17], [16], [20]. Although stored, a materialized view is not an SR. It is indeed entirely calculable through its (view) scheme.

Recently, we proposed to add the stored and inherited relation (SIR) construct to this model, [1]-[4]. The construct roots in [21], part of the popular in nineties trend to harness inheritance in the relational DBs. E.g., see [26] or Postgres, [25], or later proposals, [12]. A SIR, say R, is a 1NF relation with both SAs and IAs, the primary key being SAs only. We refer by default to the projection of R on its all and only SAs as to R, and call it base of R. We also say that the IAs enlarge R, and refer to the IAs scheme as to Inheritance Expression (IE). The crucial advantage of SIRs as base tables over the logically the same base tables, but SRs only, as required by the present model, is that no IA may create a normalization anomaly. Unlike it would often happen if the same attributes were SAs instead. Two important advantages for queries to a DB with SIRs without any normalization anomaly follow, with respect to the equivalent queries to the DB with normalized SRs only, i.e., the queries providing for the same output, [1] - [4]:

(1) A query Q addressing any SAs or IAs of SIR R can be Logical Navigation Free (LNF), while an equivalent query Q’, addressing normalized R_ as stand-alone SR named R, would typically require some LN. Recall that the concept designates joins among base tables, typically equijoins on foreign and referenced keys, [24]. Recall also that the normalized SRs as base tables of an SQL DB suffice for every SQL query to the DB. If Q’ is such a select-project-join query, Q consists then, typically, from the select-project part of Q’ addressing SIR R only. Q is then in practice always less procedural, i.e., requiring fewer characters, than Q’. In addition, joins are often felt dreadful, the outer ones especially, [9], [19]. Not surprisingly, clients typically at least dislike the LN. We designate any SIRs free of it for some queries as SIR for LNF queries.

(2) IAs can be calculated attributes (CAs), i.e., defined through relational and value expressions or sub-queries, perhaps with scalar or aggregate functions. A query Q to SIR R with CAs may then be free of defining any of these, selecting a CA by name only. I.e., Q can be a CAF query, avoiding the procedurality of the CAs specification within the equivalent Q’ to R_. SAs with the same names and values as CAs would do in theory as well, but most often would denormalize the base table to 2NF at best (as we recall by examples later on). We designate any SIRs with CAs as SIRs for CAF queries.

A SIR R can provide for both LNF and CAF queries. At present, the only practical way to provide for the capabilities of any SIR R is view R which we call conceptually equivalent to or the canonical view of SIR R, C-view R in short. Every C-view R is simply logically, i.e., mathematically, equal to SIR R i.e., the attributes names and order are the same, as well as every tuple. The only difference is physical: every SA A in the base R_ of SIR R is IA A in view R, each of these IAs being inherited from the same stand-alone SR R_. That R_ may actually be a pre-existing base table we referred to in (1) that one had to rename somehow to create view R. Recall that SQL prohibits any same name relations in a DB.

The “price” to pay for (1) or (2) for SIR R with respect to R_ alone as a base table is the proceduralality of the IE, i.e., the minimal number of characters or keystrokes to define it. For SQL, it is some additional proceduralality for Create Table R for SIR R, [1]. The similar price for C-view R in SQL is the proceduralality of Create View R. The general advantage of every SIR R is that the IE can be less procedural than the Create View R, [1]. The rationale is that the latter has to redefine as an IA every SA of R_. This obviously must cost some proceduralality. By the same token, to create SIR R is always less procedural than to create R_ and C-view R. In popular terms, every SIR R is a view-saver for C-view R. Actually, SIRs are also less procedural to maintain, [1].

All this is our rationale for SIRs. We follow the general trend in DB-science and in entire CS in fact. Recall that this is why the relational model took over the Coddasyl one, although the latter was already in use, e.g., in Oracle Coddasyl DBS. Likewise, is why it took finally over every other earlier DB model. The assertional (declarative…) relational algebra queries or, better, the predicative ones, were indeed in general considerably less procedural than any equivalent navigational ones in any of these models. Also, it is the lower proceduralality of the higher-level programming languages for general programming that wiped out the use of assemblers for it…. See oldies on the subject, e.g., early editions of [8].

In [1], we proposed Create Table extended to SIRs, providing thus also for the IE. One specifies there every SA as one would do within Create Table at present. However the former may interlace with the IAs. Create Table of a SIR includes similarly any table on- or off-line may define at present. Recall that these specify the primary key, foreign keys (FKs), etc. The IE may define every IA as C-view R would do. We term every such IE explicit. Accordingly, we call explicit every Create Table R for SIR R with
an explicit IE.

An explicit Create Table R for SIR R first enumerates thus all the SAs and IAs, perhaps interlacing. The clauses From with, perhaps, its sub-clauses follows, as well as any table options. We qualified of SIR-enabled every DBS (or DBMS, as some prefer) supporting any Create Table with an explicit IE. We showed that making SIR-enabled a popular DBS should be simple.

Additional rules for SIRs in [1], including SQL ‘*’, providing for implicit IAs for queries and views at present, may provide analogously for an implicit IE in Create Table for SIRs. In practice, an implicit IE may have implicit IAs or an implicit From clause. The latter may lack of sub-clauses clauses necessary for the C-View or may even be entirely omitted (an empty IE). We call implicit everyCreate Table R with an implicit IE. We also talk about the implicit scheme of R. An implicit IE is always less procedural than the explicit one.

We supposed further that every SIR-enabled DBS typically transparently preprocesses an implicit IE into the explicit one for any further processing, e.g., as for ‘*’ at present. An exception is every (implicit) CA declared as if it would be at present a so-called in [3]. We have shown that it holds provided that (a) SIR R is a so-called in [3] natural SIR and (b) one basically considers FKs as E. Codd originally in [7], thus more generally at present.

Below, we continue analyzing SIRs. We first illustrate the above overview of SIRs with motivating examples. We also recall our definition of FKs, based upon Codd’s one [7], [13]. Next, we discuss natural SIRs more in depth as in [3]. We focus particularly on the rules deriving the explicit scheme from the R, scheme with FKs as the implicit one. We then extend these rules to SIRs other than the natural ones, with CAs in particular. The goal is always a less procedural implicit scheme.

Afterwards, we generalize Alter Table for SIRs defined in [1] to all the implicit schemes proposed here. In particular, altering an SR R by declaring an FK or by refreshing the IE as we propose enlarges R into the natural SIR R. One may provide accordingly LNF queries to a DB that preexisted the upgrading of its DBS, while basically keeping the legacy applications running. We expand the main, to our best knowledge, first formulation of SIRs By Example in Introduction we recalled that for every SIR and C-view R, p (Create View R) is the minimal number of characters (keystrokes) to express S, without convenience spacing especially.

**II. NATURAL SIRS**

II.A SIRs By Example

Our framework for motivating examples is the “biblical” S-P DB, Fig. 1. S-P seems the first DB illustrating the relational model, [8]. It is also the mold for about every present DB. Hence, properties of S-P generalize accordingly.

**II.1 SIRs By Example**

Our framework for motivating examples is the “biblical” S-P DB, Fig. 1. S-P seems the first DB illustrating the relational model, [8]. It is also the mold for about every present DB. Hence, properties of S-P generalize accordingly.
The reason seems to be the presence of SAs, hence the use of Create the concept of base table to every table with SAs and VA's as well.

instance, it would be necessarily so for SIR-enabled MS Access.

Despite that, as an IA, it is formally a part of the IE. In contrast, (4), declared VA. We recall the related details when we overview the scheme. Actually, the DBS would pass it to MySQL as is. That one WEIGHT_KG would not be preprocessed then to its explicit format of a VA only, i.e., without From P clause. This IE could be in fact declared through the same statement. Observe also that for popular DBSs providing for VAs we spoke about in introduction. WEIGHT_KG could enlarge P accordingly. E.g., at MySQL, the enlarged P could be:

(5) Create Table P As (P# Char 4, PNAME Char 20, COLOR Char 10 INT(WEIGHT * 0.454) AS WEIGHT KG) WEIGHT KInt, CITY Char (30) (From P, ) Primary Key (P#));

The obvious benefit is further reduction of procedurality, with respect to view P as well therefore. On (yet hypothetical) SIR-enabled MySQL, with the canonical implementation proposed in [1], P could be in fact declared through the same statement. Formally, it is then SIR with (5) as the implicit scheme. The (implicit) IE consists of the definition of WEIGHT K in the format of a VA only, i.e., without From P clause. This IE could be preprocessed to the explicit scheme (4) on every SIR-enabled DBS. In practice, as already hinted to, we suppose (5) exclusive to a SIR-enabled DBS with the kernel DBS supporting VAs, e.g., MySQL. WEIGHT_KG would not be preprocessed then to its explicit scheme. Actually, the DBS would pass it to MySQL as is. That one would consequently processes it as it would do for WEIGHT K declared VA. We recall the related details when we overview the implementation of a SIR-enabled DBS in Section VI.

Incidentally, this specific processing of WEIGHT K in (5) is the reason why there is no { } brackets around WEIGHT K there. Despite that, as an IA, it is formally a part of the IE. In contrast, (4), with { } thus would be the only possibility for SIR on P a SIR-enabled DBS with the kernel DBS not supporting VAs, [1]. For instance, it would be necessarily so for SIR-enabled MS Access.

Besides, the vocabulary of the kernels supporting VAs extends the concept of base table to every table with SAs and VAs as well. The reason seems to be the presence of SAs, hence the use of Create Table still, although conveniently extended. This reason motivated us to extend the terminology to tables with SAs and any IAs, i.e., to any SIRs, [1].

All the above generalizes in fact to every SIR with IAs declared as would be VAs for the kernel. Thus, the example illustrates that every present base table with VAs is in fact a specific SIR. Given the lasting popularity of VAs, since Sybase in the eighteens to our knowledge, we may rationally hope for the future popularity of SIRs providing more generally for CAF or LNF queries.

Ex. 5. Suppose that P.WEIGHT is in pounds, while queries 

Ex. 6. Suppose that, in addition to LNF queries, SP clients wish for queries selecting QTY for some supplies that the former is for QTY there constitutes with respect to the entire supply of the part supplied. Having to specify the value expression calculating PERCENTAGE in every query in need, e.g., by a sub-query that follows, with its substantial procedurality thus, would be anything but practical for the clients. To simply add PERCENTAGE as an SA to SP after QTY, clearly would not make DBA happy. The only practical approach is to make it a CA of SIR SP or of C-View SP providing for LNF queries as well. The queries could invoke PERCENTAGE simply by name, becoming CAF queries for. One could create the required SIR SP, e.g., through the explicit scheme as follows:

(6) Create Table SP (SP# Char 5, [NAME, S.CITY, STATUS] P# Char 5 [PNAME, COLOR, WEIGHT, P.CITY] QT Int

Round(100*QTY/SELECT sum(X.qty) from SP X where X.p# = SP_p#), 3) as PERCENTAGE From SP Left Join S On (SP.S#= S.S#) Left JOIN P On (P.p# = P.P#)); Primary Key (SP#, P#));

One may easily double check that IE above would be again less procedural than Create View SP for C-view SP. Recall also that PERCENTAGE cannot be a VA for any relational DBS at present. Hence SIR SP with and, more generally, any SIRs with CAs that cannot be VAs, are the only known view-savers for ‘their’ C-views at present. Hence, again, they would always be better choices for the DBA as well. @

II.B Foreign Keys for SIRs

Despite being fundamental to the relational model, the concept of the foreign key appears still surprisingly imprecise. The original definition is in [7]. Codd amended it later several times, [13]. The present definitions in textbooks or for popular DBSs differ from the original and are not all equivalent. For SIRs, we basically stick to the original. We thus call foreign key (FK) an SA and an SA only, perhaps composite, that (i) is usually not a (stored) primary key, (PK), but every of its values could be that of some uniquely chosen PK. Then, (ii), FK “cross-references” its (own) relation and the one with PK. One qualifies then usually the latter also of the referenced key (RK). Likewise, one qualifies so the relation with the RK, say R'. In turn, the FK may be qualified of the referencing one, as well as its relation, say R.

As [7] details, (i) implies that FK domain is a sub-domain of that of RK. Originally, it meant in particular that FK and RK shared also the proper name, as there was no distinction between domains and attributes by then. Then (ii) means that every FK-value v, related to a tuple t with PK-value w, provided that t exists. FK idea realizes thus the “cross-referencing” through logical pointers, unlike through the physical one, the basic mode for referencing by the times of [7] (and unfortunately still at Internet). The benefit claimed is the logical/physical data independence. In particular, as known, if a query needs some values in R together with some referenced ones in R', then the (left) FK-join: R left outer join R' on FK = RK in the query expresses the referencing, regardless of underlying physical data structures and changes to these. Likewise does the equivalent right FK-join, or, sometimes, the equivalent inner FK-join if the referential integrity is enforced. Notice that FK-joins in queries constitute the already mentioned LN. See oldies for more on the theme.

Below, we consider that for any SIR R an SA (named) F, perhaps composite, is an FK for either of the two reasons:

a. F is declared so through the familiar FOREIGN KEY clause in Create Table R or Alter Table R. Every such F is a declared FK. For a declared F, RK may be a candidate key on some popular DBSs. This is nevertheless at best, a debatable choice, since error-prone in the absence of the table option for candidate keys in SQL, unlike for the PK. In every case, a declared F is subject to the usual referential integrity. As at present, for every FK besides, since F can be an FK iff one declares F so, according to any definitions of the concept we are aware of, especially for SQL.

b. F is atomic and is neither the primary key of R nor a declared FK or within the latter. Also, prior to the processing of Create Table should provide for every supply selected, the percentage that the QTY there constitutes with respect to the entire supply of the part supplied. Having to specify the value expression calculating PERCENTAGE in every query in need, e.g., by a sub-query that follows, with its substantial procedurality thus, would be anything but practical for the clients. To simply add PERCENTAGE as an SA to SP after QTY, clearly would not make DBA happy. The only practical approach is to make it a CA of SIR SP or of C-View SP providing for LNF queries as well. The queries could invoke PERCENTAGE simply by name, becoming CAF queries for. One could create the required SIR SP, e.g., through the explicit scheme as follows:

(6) Create Table SP (SP# Char 5, [NAME, S.CITY, STATUS] P# Char 5 [PNAME, COLOR, WEIGHT, P.CITY] QT Int

Round(100*QTY/SELECT sum(X.qty) from SP X where X.p# = SP_p#), 3) as PERCENTAGE From SP Left Join S On (SP.S#= S.S#) Left JOIN P On (P.p# = P.P#)); Primary Key (SP#, P#));
Natural FKS are the most popular, perhaps surprisingly for some. The rationale is the least procedural FK-joins in queries. Atomic declared FKS do the same, but require the declarations, while the referential integrity is not always the must. Observe also that our definition of an FK implies that every composite FK must be declared. The actual rationale is that the referencing FK -> RK is by attribute position at present, not by name sharing. It may happen however that the ordered set of the proper names forming a composite FK is the same as the ordered set of the referenced proper names. One may consider then that FK and RK share a (proper) name. Also, even the composite FK is usually the PK of R'. We call accordingly PK-Named (PKN) every FK, atomic or composite, that shares the name of an RK being a PK. A PKN FK can be natural or declared, while every natural FK is PKN by definition.

Furthermore, we suppose that, in every Create Table R submitted to a SIR-enabled DBS, every PKN FK F and only such F implies specific IAs in the explicit Create Table R. The latter is the actual R scheme, we recall. We call these IAs Natural Inheritance, (NI), in R from R' or through F and define them as follows. Let Δ be the set of all the non-PK attributes of R'. Let also ‘…’ mean some or no attributes. Then, let us denote the submitted scheme defining the relation as R(Δ'…, F, …, Δ') and let us consider that for any composite F (F1, F2…), the notation R.F = R'.F means R.Δ1 = R'.Δ1 and R.F2 = R'.Δ2. Then, NI consists of the IAs Δ'1 enlarging R1 to R through the following pseudo-SQL C-view scheme: select …, F1, Δ'1… From R left join R' on R.F1 = R'.F1. In contrast, we consider that any non-PKN FKS in R scheme imply the referential integrity only.

It follows from the above that every declared PKN FK F implies both the referential integrity and the NI. If one does not want the former, one should not declare such F. Provided that, as usual, there is only one relation with PK F in the DB, for any atomic PKN FK F a natural FK F will result and fit the goal. Otherwise, one should change in addition the names of all PKN F other than R.F. Notice that this need should be rare, as having several relations sharing a primary key is usually not the best design of a relational DB. Anyhow, all this cannot work in contrast for any composite PKN FK F. One solution is to add to R a surrogate that is an atomic FK named, let us say, C, uniquely for a PK in the DB. The composite RK becomes consequently a candidate key only. Then, it suffices to replace F with C in R. The latter will be a natural FK hence provide for the NI only, as wished. The classical Ex. 14 illustrates the whole technique.

Finally, we suppose that, as at present, no Create Table R’ or Alter Table R’ propagates to any existing table R ≠ R’. In particular no SA F of some R can become implicitly a natural FK, since one issued Create Table R’ with PK named F or issued Alter Table R’ that ended up with the PK R’.F. In practice, it means that no such statement can enlarge R with the NI from R’. A dedicated Alter Table R we discuss in Section is necessary.

Ex. 7. Natural FKS are present in S-P, assuming S and P created first. SP.S# and SP.P# are the natural PKN FKS then, with S.S# and P.P# being the respective RKs. The original verbal description of S-P scheme indicates that each pair has a common domain. Finally, as the natural FK, SP.S# in SP scheme (1), will imply the NI from S consisting of {SNAME…SCITY} in (3). Likewise, SP.P# will imply the NI from P. As it will appear formally in next section, SP scheme (1) will lead then to SP scheme (3) as the explicit scheme, with (1) as an implicit one.

The original description of S-P also does not mention any referential integrity. Nevertheless, at Fig. 1, every SP tuple respects that constraint for each FK. Regardless, one may insert, e.g., P7 into SP, without the presence of P7 in P. The feature can be useful, e.g., if DBA allows for the data for P7 in P to be inserted later.

If the referential integrity was in contrast required for a pair, e.g., (SP.S#, S.S#), one should declare it in Create Table SP or Alter Table SP. This, using the usual: Foreign Key (S#) References S(S#)… with, also usual, On Delete and On Update options perhaps. SP.S# would be a declared (atomic) PKN FK then. On the other hand, if in S-P as on Fig. 1, S would have been created after SP, then SP.S# would not be a natural FK anymore. Consequently, there would not be the NI through it in the explicit SIR SP scheme resulting from Create Table SP (1).@
Prop. 2. Suppose that Create Table R in some DB defines at present an SR R with PKN FKs. Accordingly, consider the following generic formula for such Create Table R, where ‘...’ designates some SAs or none:

(7) Create Table R (..., F1, ..., F2, ..., <table options>);

Then, (a) the following formula defines the explicit Create Table R for the natural SIR R, with From clause from C-view R above:

(8) Create Table R (..., A1, ..., A2, ..., <From clause>, <table options>);

Also, (b) one may consider Create Table R (7) as the implicit one of the natural SIR R (8). Moreover, the implicit IE is then empty, i.e., p(IE) = 0, we recall. Next, (c) the SAs in R (7) and the table options there, define also the base R_r of R (8). Finally, (d) for every SR R as above and every base R_r thus, there is only one natural R in the DB and vice versa.@

Proof. (a) is obvious. To prove (b), one should provide a deterministic algorithm for the explicit IE in (8) given (7). We sketched the latter in [3]. In the next section, we provide a more complete formulation. IE is empty, since by definition of an SR, there cannot be an IA or From clause. For (c), SR R scheme above, except for the relation name is also by definition the one of R_r in (8). Finally, (d) is obvious.@

Accordingly, given (d), for every natural SIR R, we may say sometimes that R is so for R_r or for SR R.

Ex. 9. Create Table SP (1) and Create Table SP (3) are clearly conform to Prop 1. Hence SP (3) is the natural SIR for SP (1) and for the base SP_r of SP (3). Finally, it is the unique natural SIR for both.@

Notice that, usually, DBS would process Q_2 by the standard query modification approach that would walk backward the above steps. For every Q_1, renames SR R to R_r and replaces all the FK-joins of Q_1 with the FK-joins in (5). The relation defined by these joins contains the same attributes as the natural SIR R explicitly defined by (5), except, perhaps, that (i) some attribute names became qualified or (ii) the order of the attributes is different. Given the properties of left outer equi-joins, the modified Q_r is equivalent to Q_1. Accordingly, view R is a view for LNF queries for any such Q_r. SIR R is consequently a view-saver for any such LNF queries. R scheme is possibly the least procedural one, since DBA may choose the implicit one with empty IE (what every DBA will likely do in practice then).@

Notice that CITY became qualified. The latter query is in turn equivalent to the following one with nested From:

(11) Select SP.#, SNAME, CITY From SP Left Join S On SP.# = S.#;

Notice that, usually, DBS would process Q_2 by the standard query modification approach that would walk backward the above steps towards Q_1. Similar conclusions will hold more generally for every select-project-join query to S-P SP or S and P, with LN through the FK joins over FKs of SP. Accordingly, view SP is a view for LNF queries, requiring the above FK-joins otherwise. SIR SP is then a view-saver for the same LNF queries. It is also possibly the least procedural one, since one can define the SAs only.@

Recall also that every IA A of a natural SIR R, is a natural one itself. By definition, it thus has the same name as an attribute of some base table R_r of R references, called also source of R.A. Thus one may consider that for every query Q to R only that we qualified of an LNF one, for every IA A that Q perhaps addresses, Q addresses then in fact some R.A. One may say then that Q is an LNF query not only to R, but also, indirectly through every R.A addressed, to every base table R_r that is the source of. For some, that meaning of an LNF query is perhaps the primary one even,
Accordingly to this terminology, one can formulate Prop. 3 in an alternative way that some may find more appealing:

Prop. 3bis. Suppose that a DB has a base table R with the scheme of the SAs that makes it the implicit one of a natural SIR R. Let R'1… be every base table that R references. Next, let Q1 be a select-project-join query addressing some SAs in R scheme and some attributes of R', or of R'2 or…, with every join being a left FK-join preserving R or with any join equivalent to. Then, for every possible Q1, there is a query Q2 addressing SAs in R scheme and such that (i) Q2 is equivalent to Q1, (ii) Q2 is an LNF query also to ever R' that Q1 addresses (through the joins) and (iii) Q2 is the select-project part of Q1 with From R clause only, except that some attribute names in Q1 may end up qualified. R is consequently a possibly the least procedural view-saver for such LNF queries. 

Observe finally that if Create Table R defining at present an SR R only, may define the natural SIR R instead, then it provides for the discussed attractive LNF queries, at no additional data definition cost for DBA. We now describe the algorithm effectively for providing that capability, i.e., of inferring the explicit natural SIR R scheme from the one of the SR R, on any popular DBS.

II.C Inferring Explicit Schemes of Basic Natural SIRs

As already stressed, suppose every referenced relation to preexist the referencing one. Suppose also that SIR-enabled DBS gets Create Table R with SAs only and, may be, with some of these being declared FKs. The scheme may be thus an implicit scheme of the natural SIR R. SIR-enabled DBS processes then the Create Table as follows. The algorithm mainly generalizes our motivating examples. The outcome is the explicit Create Table R. We specify the rules only verbally, omitting easy details the actual implementation would require. We take for granted that the implicit Create Table R defines SAs only. We also consider only the canonical implementation of a SIR-enabled DBS in [1], we recall in Section VI below. The SIR-enabled DBS creates and manages then every SR R as base table R, and C-view R within the kernel DBS.

Alg. 1. 1. (Determine every natural FK). For every (atomic) SA R.A that is neither a primary key nor a declared FK or within such FK, check in the meta-tables, provided by every popular SQL DBS at present and often named SYSTABLES for base tables and SYSVIEWS for views, e.g., in DB2, whether there is a unique relation (named) R', with the primary key sharing the name and the domain of R.A. If so, R.A is a natural FK. Next, (i) if R' (name) does not end up with ‘,’ then ‘R := R’. Else (ii) if R’(‘)’ is not in SYSVIEWS etc., then R := R’. Else R is not a basic natural SIR.

Then, check the rules for compound natural SIRs we outline later.

2. (Processing every PKN FK). For every PKN FK with R determined in (1) or analogously for a declared one, retrieve from SYSTABLES the name of every non-PK attribute of R’. Then (iii), place all these names, qualified if needed, in Create Table R, as in (a) in Def. 1.

3. (Create From clause). Let F1… be the PKN FKs enumerated in the left-to-right order in Create Table R and R'1… be the referenced relations. Suppose for the form of the string below, for every composite F, the simplified notation we indicated before. Then, after the last SA and before the table options, insert the string in the form of: From R_L Left Join R’1 On (R.F1 = R’1.F1) Left Join R’2 On (R.F2 = R’2.F2)…@

Ex. 12. We skip the easy but tedious proof of the rules. We only show that they build the explicit Create Table SP (3) from (1). Suppose thus S-P1.S and S-P1.P already created. Rule 1 produces names (S, S#) and (P, P#). For the former, Rule 2 finds S# in (1). It thus inserts SNAME…S.CITY, right after S#. Likewise, it inserts PNAME…P.CITY right after SP#. Finally, Rule 3 builds From clause in (3) and terminates the explicit IE.@

For the DBA, as already hinted to, the rules mean simply p(IE) = 0. They thus mean free bonus of zero additional time for creating, instead of S-P.SP, the natural SIR SP (3), with its p = 112 (explicit) procedurality of the IE. This, to provide the clients with also free, as then bonus of typically far less procedural LNF queries. For the DBA again, an even bigger bonus is with respect to the present situation. One saves indeed 100% of procedurality p = 152 of Create View SP (2) for the same purpose.

II.E Compound Natural SIRs

A compound natural SIR R inherits through some FKs from SIRs. These can be natural perhaps compound themselves, or others. In other words, a non-PK attribute of an R’ can now be an SA or an IA. Operationally, as usual today, we suppose again every R’ being created before R. By the same token, we suppose that later alterations of any R’ schemes do not cascade to R. Here are motivating examples of compound natural SIRs. They seem framework for frequent practical cases.

Ex. 13. Suppose one alters S-P scheme as follows. An additional relation CG (CITY, GPS) stores uniquely for each city the GPS location. Suppose further that on a SIR-enabled DBS, one creates CG first, then S and P with their S-P schemes, Fig. 1 and SP through its scheme (1), at last. CG has no FKs, hence its scheme above defines an SR. Then, the S scheme has only one FK that is the natural FK S.CITY, referencing CG.CITY. S is now therefore a basic natural SIR, with S-P scheme as the implicit one and the following explicit scheme, inferred through the rules above:

```
(13) S (S#, SNAME, STATUS, CITY \{GPS From S Left Join CG On S.CITY = CG.CITY\})
```

The explicit Create Table S scheme for (13) is obvious to figure out. P becomes a basic natural SIR analogously. But, the NI for SP defined by Prop. 1 includes now also two IAs: S.GPS and P.GPS. Hence, SP becomes the compound natural SIR. Its explicit Create Table evolves to:

```
(14) Create Table SP (SP# Char 5 \{SNAME, STATUS, S.CITY, S.GPS\} P# Char 5 \{PNAME, COLOR, WEIGHT, P.CITY, P.GPS\} QTY INT \{From SP Left Join S On (SP_.S# = S.S#) LEFT JOIN P On (SP_.P# = P.P#) Primary Key \{S#, P#\}\} LNF queries to SP may now address both P.GPS and S.GPS. But, it is easy to see that Alg. 1 does not let to infer (14) from (1) anymore. It needs the completion we show soon.
```

Finally, it’s instructive to appreciate the procedurality gain with LNF queries searching for GPS data. E.g., suppose the search for SP.S#, SNAME, S.CITY, S.GPS and SP.P#, PNAME, P.CITY, P.GPS, as well as QTY, for every supply with QTY > 100. For SP with SR CG added as base table, every SQL query Q1 expressing the search would need some LN, e.g., the nested one as follows:

```
(15) Select SP.S#, SNAME, S.CITY, S.GPS, SP.P#, PNAME, P.CITY, P.GPS, QTY From SP Left Join S (S Left Join CG On S.CITY = CG.CITY) On SP_.S# = S.S# Left Join P (Left Join CG P.CITY = CG.CITY) On SP_.P# = P.P# Where Qty > 100;
```

The LNF Q2 to S-P1 would be in contrast simply:

```
(16) Select SP.S#, SNAME, S.CITY, S.GPS, S.P#, SP.P#, PNAME, P.CITY, P.GPS, QTY From SP Where Qty > 100;
```

We have p(Q1) = 203 and p(Q2) = 83. Thus Q1 is almost 2.5 times more procedural than Q2. Besides, no wonder that the complexity of LN through nested joins in (15) is not what most clients like best. The result would not change much if, e.g., one familiar with properties of joins unnested these while formulating Q1 or replaced some with the left natural ones etc. Recall finally that all these advantages of Q1 come for free at the data definition level for DBA. I.e., if looked upon as SIR implicit schemes, Create Table S, Create Table P and Create Table SP could make possible for Q1, instead of forcing Q2 only at present. @
Here, \((S\#, P\#)\) is a declared composite PKN FK. Suppose also that \(J\#\) is not a natural FK for some reason. For a SIR-enabled DBS, the above scheme is the implicit one of SIR SPJ. Neither \(S\#\) nor \(P\#\) is a natural FK in SPJ, since both are within a declared PKN FK. Hence, SPJ would be a natural compound SIR where the explicit scheme naturally inherits every non-key SA and IA of SP.

Notice that the referential integrity between SP and SPJ would be enforced, as for every declared PKN FK and as for any FK at present. If it is not desired, then, as said generally before, one should not declare \((S\#, P\#)\) as an FK. The natural attributes in NI through \((S\#, P\#)\) that one wishes to preserve for LNF queries and which are not in the NI through \(S\#\) or through \(P\#\), should be explicit. Actually, this amounts to QTY only. A better approach to inherit QTY instead implicitly as well is through the already discussed technique of a surrogate, say SP\# here, added to SP, i.e., enlarging the implicit scheme of SP to SP (SP\#, S#, P#, QTY). The composed key \((S\#, P\#, J\#)\), becomes a candidate key only then. SPJ may become SPJ (SP\#, J\#, ALLOC), with SP\# being the PKN FK and QTY becoming an implicit IA, since in the NI of SP\#.

Anyway, whether QTY is implicit or explicit in SPJ, a SIR-enabled DBS will provide for the LNF queries addressing any SAS of SPJ and through the IAs, any SA or IA of SP. Hence, through IAs of SPJ, such query will be also able to, transtively and transparently, address every attribute of S and of P. Thus, again at\(I_A\)s of SPJ, such query will also be able to transitively and enlarging the implicit scheme of SP to SP (SP\#, S#, P#, QTY). The composed key \((S\#, P\#, J\#)\), becomes a candidate key only. Then, SPJ might become SPJ (SP\#, J\#, ALLOC), with SP\# being the PKN FK and QTY becoming an implicit IA, since in the NI of SP\#.

Select SNAME, PNAME From SPJ Where J\# = 'J1'.

In contrast, the necessary LN in Q, would make the latter clearly three times more procedural and would be evidently awkward for many clients:

Select SNAME, PNAME From SPJ Left Join SP On SP.S\# = S.S\# Left Join P On SP.P\# = P.P\#) On (SPJ.S\# = SP.S\# And SPJP\# = SP.P\#) Where J\# = 'J1'.

Ex. 15. Suppose now for S-P1, that S is S-P-S, but one enlarges S-P with the calculated attribute WEIGHT KG as in Example. Suppose further that DBA again creates S and P before SP, with P with WEIGHT KG becoming SIR P, we recall. Then SP implicitly defined through SP_scheme named Create Table SP, would remain a natural SIR. However, it will be a compound one this time, regardless one defined WEIGHT KG explicitly or implicitly as a VA in P. The explicit Create Table SP would become:

\[
(19) \text{Create Table SP (S\#, P\#, J\#, ALLOC... Primary Key (S\#, P\#, J\#), Foreign Key (S\#, P\#) Referencing SP (S\#, P\#))}.
\]

Given these examples, the enhancement to the rules for basic natural SIRs in the previous section can be as follows. The new need is to recognize for every R whether itself it is not a SIR.

Alg. 2. (i) - Rule (1) in Alg. 1 states that R can be a compound natural R if \(R^{"^\sim_\sim}\) is in SYSVIEWS etc. Consider so now. Then, R is effectively a compound natural SIR, since \(R^{"^\sim_\sim}\) is a SIR. Hence set \(R:= R^{"^\sim_\sim}\).

(ii) - The NI in R from \(R^{"^\sim_\sim}\) is now defined through SYSVIEWS etc. This one should be, as usual, every IA of view \(R^{"^\sim_\sim}\) defined there other than every IA inherited from RK in \(R^{"^\sim_\sim}\). The latter is to be found through SYSTABLES etc.

(iii) - Perform finally rule (3).

We skip the easy proof in favor of the motivating example.

Ex. 16. Consider again S-P altered as in Ex. 13. Then rule (1) above will find \(S\#\) that S_table in SYSTABLES etc is R' with S as R'' in SYSVIEWS etc. The control will pass to rule (i) that will set S as R'. Likewise, for P, it will find R'' := P. Rule (ii) will then determine for S from SYSTABLES etc. that S.S# is the RK, hence it will find out from SYSVIEWS etc. that \{SNAME,...SP\#\} is the NI for SP.S#. In SP., likewise, it will determine \{PNAME,...P.S\#\} as the NI for SP.P#. Then, after applying rule (3) again, the end result for From clause would be the one in (14) and, altogether, Create Table SP (14) will be the explicit one in our case.

Likewise, for S-P altered as in Ex. 15, Alg. 1 for the basic natural SIR and SYSTABLES etc. alone will determine for SP, the NI from S. For P in turn, to find it out, Alg. 1 will call Alg. 2 for the compound natural SIR. The explicit Create Table SP (19) will be the overall result:@

III SIRs with PKN FKs and Explicit IAs

SIR R with some PKN FKs may have also an implicit scheme with some explicit IAs. The explicit R scheme contains then in addition every NI. The FK-joins defining the latter enlarge the implicit From clause declared for the explicit IAs. The latter is the explicit From clause without however every FK-join defining a NI in R, i.e., through a PKN FK. An empty implicit From clause may ultimately result. For every implicit IE of R with explicit IAs, we must obviously have \(p(\text{IE}) > 0\). Still, the latter may provide for substantial procedural savings with respect to the explicit IE and even more thus as the view saver.

The typical needs for the explicit IAs seem as follows. (i) R has one or more calculated IAs (CA), defined through a value expression inheriting from SAs or other IAs in R or from some R', or defined by a sub-query, or defined as if A was a VA for the kernel DBS, provided the latter supports such attributes. Then, (ii) for some FK F, R may have for privacy, only some or even none of the natural IAs of R through F. Or, (iii), F may have the same IAs as in its NI, but, for query convenience, displaced within R or renamed for some. Finally, (iv) F, R may share R'. This is contrary to the assumptions for a NI, we recall. Here are motivating examples, illustrating these needs. IE parts being within \{\} or defining IAs that could be VAs at present, the procedurality gains with the implicit IE for the DBA are obvious. Recall also that even the explicit IE would be still less procedural than C-view SP, as one may easily double-check. Finally, we skip the easy but tedious formulation of the operational inference rules.

Ex. 17. (i) Consider SIR P from Ex. 13 enlarged with WEIGHT KG placed after WEIGHT. Supposing that the kernel does not provide for VAs, the implicit Create Table P would be:

\[
(20) \text{Create Table P As (P\#, Char 4, PNAME Char 20, COLOR Char 10, WEIGHT INT} | \text{INT(WEIGHT * 0.454) As WEIGHT_KG}) \text{ CITY Char (30), Primary Key (P\#))}.
\]

SIR-enabled DBS would enlarge then (20) to the explicit Create Table as follows:

\[
(21) \text{Create Table P As (P\#, Char 4, PNAME Char 20, COLOR Char 10, WEIGHT INT} | \text{INT(WEIGHT * 0.454) As WEIGHT_KG}) \text{ CITY Char (30) \{GPS From P\# Left Join CG On P\#.CITY = CG.CITY\) Primary Key (P\#))}.
\]

The explicit IE is visibly more than twice procedural than the implicit one. Alternatively, suppose now that MySQL is the kernel. The implicit Create Table P could contain the VA WEIGHT KG:

\[
\text{CREATE TABLE SP (S\#, P\#, J\#, ALLOC... Primary Key (S\#, P\#, J\#), Foreign Key (S\#, P\#) Referencing SP (S\#, P\#)).}
\]

- 7 -
Ex. 18. (i) Suppose that for some security reasons, no attribute of Supplier should be in SP available for LNF queries, except for S#, somehow renamed so to hide its relationship to S. The implicit SP scheme could simply be:

(24) Create Table SP (X Char 5, P# Char 5, QTY INT, Primary Key (S#, P#));

The explicit one would be:

(25) Create Table SP (S# Char 5, P# Char 5, NAME Char 20, COLOR Char 5, WEIGHT Int, CITY Char 5, QTY INT) {PNAME, COLOR, WEIGHT, P.CITY} QTY INT {From SP_ Left Join P On SP_.P# = P.P#} Primary Key (S#, P#));

(ii) Suppose now that only the attributes SNAME, CITY of S-P-S should be visible to LNF queries to SP. For some convenience, they should also be placed after QTY. The implicit Create Table SP could be:

(26) Create Table SP (S≠ Char 5, P≠ Char 5, QTY INT (SNAME, CITY From SP_ Left Join S On SP_.S≠ = S.S#) Primary Key (S≠, P≠));

For both (i) and (ii), the renaming of SP.S# was necessary, as it would be a PKN FK otherwise. It would then imply N in the explicit scheme, obviously contradicting the specs.

The explicit scheme for (26) would be again substantially more procedural, by about 50% visibly:

(27) Create Table SP (S≠ Char 5, P≠ Char 5, SNAME, CITY From SP_ Left Join S On SP_.S≠ = S.S#) Left Join P On (SP_.P# = P.P#)} Primary Key (S#, P#));

Ex. 19. Suppose that SP should be as the natural one, but with the additional IA named WEIGHT T. This one should indicate for every supply, its total weight, supposed to be WEIGHT * QTY. One wishes also WEIGHT T to follow QTY in SP. Being defined through a value expression, WEIGHT T cannot be a natural IA. Neither, it can be a VA, since WEIGHT is not in SP. Nevertheless the following implicit scheme is possible:

(28) Create Table SP (S# Char 5, P# Char 5, QTY INT WEIGHT*QTY AS WEIGHT_T} Primary Key (S#, P#));

The explicit scheme would be:

(29) Create Table SP (S# Char 5, SNAME, CITY, STATUS P# Char 5, PNAME, COLOR, WEIGHT, P.CITY) QTY INT, WEIGHT AS WEIGHT_T From SP_ Left Join S On SP_.S# = S.S#) LEFT JOIN P On (SP_.P# = P.P#)}_primary_key (S#, P#));

Indeed, both S# and P# in the implicit SP scheme continue to represent all their natural IAs. Next, WEIGHT T would need FK-join between SP_ and P in the implicit scheme if it should become an explicit one as is. But, this clause can be omitted otherwise, as defining also the NI from P. The procedurality of the implicit IE, say p1, in the implicit scheme is p1 = 23. For the explicit IE, we have p2 = 134. Thus, the explicit IE is almost six times more procedural. Notice finally that the procedurality, say p0, of Create View SP for C-view SP that is the only practical possibility for WEIGHT T at present, is p0 = 171. Hence, it is more than seven times greater than p1, making the implicitly defined SP quite a view-saver.

Ex. 20. SP should get as the last attribute, a calculated IA, say PERCENTAGE. For every supply, the latter should be the percentage that the QTY of that supply constitutes with respect to the entire supply of the part supplied. Sub-query below defines PERCENTAGE, leading to implicit SP scheme as follows:

(30) Create Table SP (s# Char 5, p# Char 5, qty Int {(select Round (100*Qty / (select sum(X.qty) from SP_ X where X.[p#] = SP_.[p#])) / 3)) as PERCENTAGE}) Primary Key (S#, P#));

Without the NI inferred, the From SP_ clause should follow the attribute list. But, it is not here, since would be redundant with the FK-joins referencing S and P. The explicit SP scheme would therefore be:

(31) Create Table SP (s# Char 5 (SNAME, S.CITY, STATUS) p# Char 5 (PNAME, COLOR, WEIGHT, P.CITY) qty Int, Round(100*Qty/sum(X.qty) from SP_ X where X.[p#] = SP_.[p#]) / 3) as PERCENTAGE From SP_ Left Join S On (SP_.S# = S.S#) LEFT JOIN P On (SP_.P# = P.P#)} Primary Key (S#, P#));

It is easy to calculate that the implicit IE is 2.1 times less procedural than the explicit one. In other words, the implicit IE saves 53% of the explicit one. Hence, it is even more efficient as the view-saver, (how much?).

Ex. 21. Consider the following DB named E-M providing data on employees. Some employees are managers. Each employee has an ID named E#. Each manager has the ID M# that is one’s E# renamed. An employee may share work time among several managers. An SA FRC indicates the fraction of work time spent by employee E# for manager M#. A manager may get M# and start to manage some employees before data of those or of her/himself are in E-M. E-M could accordingly contain the following base tables:

(32) Emp (E#..., NAME..., TEL..., DEP..., Primary Key (E#));
(33) EM (E#..., M#..., [M.NAME, M.TEL, M.DEP} FRC... (From EM Left Join Emp M On EM_M# = E.E#) Primary Key (E#, M#), Foreign Key (M#) References Emp (E#);

EMP is an SR and EM is visibly a SIR. Suppose EMP created first. E# is then a natural FK and EM scheme is an implicit one. The preprocessing would expand it to the (substantially more procedural) explicit scheme that we leave as an exercise. EM would provide for LNF queries on every attribute of employees or of managers in E-M.

IV ALTERING SIRS WITH FKS

One can alter every SIR R with FKs through the explicit Alter Table R, specified for SIRS in [1]. For even lesser procedurality of the statement, we now consider that for any R being an SR or SIR with FKs, Alter Table R can have also an implicit clause denoted IE (). The clause recalculates the IE, taking to the account every NI in R in particular. Specifically, it may add the IE to every SR R preexisting the upgrade of a DBS to a SIR-enabled. Recall that this makes then R a natural SIR, without affecting the data in R. The rewrite rules for processing Alter Table with IE () clause into the one with the explicit IE, are easy figure out.

Recall also that every natural SIR R, hence also the one resulting from Alter Table R for an SR R, brings to R the free bonus of the LNF queries to any SA of R and to any of the non-PK attributes of any R’. For every preexisting query, the outcome remains the same, except for every query referring to R through * or R.. Recall that such queries are not recommended for applications, hence rare. Notice also that for every preexisting SR R, in the absence of any of the discussed alterations, the DBS upgrade would not affect any preexisting queries.
Ex. 22. Consider S-P in use on some present DBS. Suppose this DBS becoming then SIR-enabled. Every preexisting query to S-P will provide the same outcome. Then, Alter Table SP IE (i) will upgrade SP to S-P1.SP. The rewrite rules will process the Alter into the explicit one, [1], visibly more procedural by far:

(34) Alter Table SP IE (S#, SNAM, S.CITY, STATUS, P#, PNAME, COLOR, WEIGHT, P.CITY, QTY From SP Left Join S On (SP.S# = S.S#) LEFT JOIN P On (SP.P# = P.P#));

New queries to SP may now be LN.F ones. Every existing one will provide for the same outcome, except for queries with: ‘**’ or ‘SP’. The alteration will not affect any existing S-P.SP (stored) content, the one at Fig. 1 especially.@

Observe finally, that, as already discussed for the creation of SIRs with FKs, for every existing SR R with PKN FKs, one should perform IE (i), only if (i) every R’ preexists and, (ii) if R’ also has PKN FKs, then (i) was already applied to R’ etc. E.g., if all the relations of Ex. 13 are preexisting SRS, one should alter CG first, S and P after and SP, at last. Otherwise for a declared FK, an error could appear, while a natural FK could silently miss an IA. R would provide then for fewer LN.F queries, obviously.

V RELATIONAL DESIGN FOR SIR DBS

The relational design has for goal the “best” collection of base table schemes for a DB. Usually, it means the smallest possible number of 4NF SRs. About always in practice then, each of these is also in 5NF and, even in less popular 6NF, [10]. Several methods for are known. Whatever is one’s favor, let us refer to the result as S. In practice, every S contains SRs with (declared) FKs. Then, S is accompanied with some scheme, say O, of the base tables creation order, so that for every referencing table R, every referenced one, say R’ as before, exists when R is being created. A run-time error occurs otherwise on every DBS of our knowledge at present.

We call relational design for SIR DBs, any methods similarly aiming at “best” collection of base table schemes that can be SRs or SIRs. “Best” means here for every SIR, the first the procedural schemes for SRs that can be possible. Every SR, “best”

Observe from our example that, e.g., before one adds WEIGHT KG, P is BCNF (and so on). It’s no more after. Indeed, FD: WEIGHT -> WEIGHT KG, makes P in 2NF at best. But, as an IA, WEIGHT KG, does not create any normalization anomalies. Unlike SA WEIGHT KG would do. The lossless decomposition through Heath theorem, [18], making P without IA WEIGHT KG, hence in 3NF again, would be senseless. Hence, P with WEIGHT KG as IA should not lose its BCNF “status”.

Similar situation occurs for SP with WEIGHT T, given FD: (WEIGHT, QTY) -> WEIGHT T there. The issue was already signaled in [4] and, by some clients of VAs, [28].The practical way out is to expand the definition of the normal forms so to take to the account that no IA may introduce a normalization anomaly. Our proposal is thus as follows:

Def. 2. A 1NF relation, SR or SIR, is INF ; i.e. _i_ = 2...6; or in BCNF iff the relation formed by all its SAs is in INF or BCNF.

Then, both P without WEIGHT KG and with WEIGHT KG as an IA are in BCNF. In contrast, P with WEIGHT KG declared as an SA, i.e., without the value expression, but only enumerated for each value of WEIGHT, would not be in 3NF even. Like P would not be also for the usual definition of 3NF and BCNF. Similarly, - for SP and WEIGHT T. Notice that the definition applies to present relations with VAs. We recall that all these are specific SRs. Finally, it is backward compatible for SRs Altogether Def. 3 is long overdue thus.

V.B SIR DB Specific Design Steps

For a SIR DB, some SRs in S may become SIR bases. Let us call S some schemes forming the intended DB possibly with SIRs. One basically seeks S where every SIR scheme is the least procedural possible, i.e., the least procedural implicit one. As it appeared, e.g., for S-P and S-P1, we may typically expect S = S’ with every SIR being a natural one, basic or compound. On the other hand, as already abundantly discussed, some SRs may need to be enlarged with explicit IAs, CAs especially. Also with respect to O, if O’ is the similar scheme for a SIR DB with all SRs and SIR bases in S, one may need O’ ≠ O. Indeed, O’ may need to take care also of the natural FKs. Actually, if the goal is a SIR DB upfront, then it is obviously not useful to define any O, just to perhaps alter it to some O’ anyhow. Altogether, beyond every presented method for SIR, designing a SIR DB may require some of the following SIR-specific steps.

(i) For every R in S, with PKN FKs, (a) rename every such FK that should not be in S’ , if there is any.

(ii) For every R in S, for every explicit IA A that should enlarge R, alter R scheme by adding A scheme, including “its” explicit LN, if the latter is not in in the explicit From clause.

(iii) Choose O’ such that for every R (i) for every natural FK, R’ preexists Create Table R’, (ii) for every IA in R, all of its source relations preexist Create Table R as well.

Step (i) may occur for some popular design methods. E.g. it would be the case if one aims at SIR DB from Ex. 18 (ii), while obtaining S of S-P at Fig. 1 as the result of the lossless decompositions of the universal relation, [24], through Heath Theorem, [18], until every base table is in BCNF at least. Step (ii) would actually be needed for as well. Step (iv) is in contrast mandatory for every SIR DB, obviously. Recall finally that for every DB, hence for SIR DBs as well if the universal relation for S has non-trivial MVDs, the decomposition should start with the Fagin’s Theorem, [14]. The Heath Theorem applies then to the resulting intermediate SRs with FDs only. More details would be out of scope here.

Besides, our prior various examples abundantly motivate the above steps. E.g., for S-P and S-P1 DBs, we have S = S’ as at Fig. 2 and only step (iv) applies. O’ could be O’ = S, P, SP or O’ = P, S SP. The final result for SP was the natural basic SIR at Fig. 3, we recall. Likewise, to design S-P and S-P1 variants in Ex. 13, again S = S’ and only step (iv) apply, O’ is either O’ = CG, S, P, SP or O’ = CG, P, S, SP, obviously. Whatever the choice is, in the SIR DB, CG would remain an SR, S and P would be basic natural SRs and SP would end up a compound natural one, we recall.

Next, in Ex. 18 (ii) again, although SP.PW is the only PKN FN, one still needs O = S, P, SP or O = P, S SP. The rationale is the LN towards S in From clause of SP, making S the source of SP.SNAM and of SP.CITY. Thus all three steps apply to this DB. Finally, the design of the SIR DB variants with WEIGHT KG, PERCENTAGE and T WEIGHT will apply any favorite DBA’s design method, followed by steps (ii) and (iii) only.

VI IMPLEMENTING SIRs

The canonical implementation we referred to above consists of a new layer, called SIR-layer, interfacing all clients and DBA, [1]. Internally, SIR-layer calls DB services of some major relational DBS, referred to as the kernel DBS. In this way, it makes the kernel SIR-enabled, in our terminology. Above SIR-layer, the relational constructs for any clients and DBA are: stored relations, SIRs and views. Underneath, i.e., for the kernel, there are necessarily only two constructs: (i) SRS, perhaps with VAs and (ii) views. Every SR or SIR with every IA declared as a VA and every view are the same above and below SIR-layer. SIR-layer simply forwards every query to these constructs only to the kernel. Every SIR with some or all IAs declared otherwise than VAs would be, is created in contrast within the kernel atomically as couple: stored relation R, enlarged with all the VAs if there is any, as well as with any table options
declared and C-view R. In particular, one may create the latter with only the proper attribute names in conflict being qualified (prefixed), e.g., as in (3), those of SAs being prefixed if needed with R . Alternatively, one may qualify all the names. The latter choice is visibly simpler to implement, while with no consequences for the queries to C-view, at any kernel we are aware of.

In particular for R , for every declared FK in R if R’ is a SIR, then SIR-layer replaces the reference to R’ with the one to R’_. The reason is that in SQL at present, every referenced relation must be an SR. As one could see, whether a relation is an SR or a SIR is easy to find out from SYSTABLES and SYSVIEWS.

SIR-layer easily extracts Create Table R_ and Create View R from the explicit Create Table R. The former contains every attribute scheme outside {} and every table option. Some ‘{’ may get replaced with ‘,’. Create View R in turn, copies the name of every attribute in Create Table R outside {} and the schemes of every IA as well as From clause terminating the IE, necessarily within {}. The last ‘}’ ends up the IE, also necessarily. To appreciate how easy the described parsing should be, e.g., practice our examples.

Likewise, one can easily see how SIR-layer should process every Alter Table and Drop Table, [1]. Finally, SIR-layer simply forwards every Create View R to the kernel, whether over SRs, SIRs or views, e.g., it would do so for view S-P with MVD above.

Next, we supposed the canonical implementation to transparently forward every Select query addressing SIR R to C-view R. Kernel DBS may internally optimize the execution time of such queries, e.g., by materializing some IAs of view R for faster joins, [16], [17], [27]. SIR-layer also forwards unchanged to the kernel, every updating query, i.e., Insert, Update, or Delete. The correctness of an updating query depends on kernel’s view update capabilities, [11]. E.g., the LNF query to S-P1, say, Q1: Delete From SP Where SNAME = ‘Smith’; would be directed thus by every kernel towards view SP. It then would be correct for MS Access and MySQL. Both provide indeed for updates of outer join views, of view SP thus. In contrast Q2 would fail on SQL server kernel and SQLite. None provides indeed that capability, forcing the updating of SRs instead. Instead of Q1, the correct query Q2 could then be: Delete From SP Join S On SP.SI/S = S.SI Where SNAME = ‘Smith’; Notice that Q2 would be correct for MS Access and SQL Server as well. The visible drawback with respect to Q1 is the LN.

With respect to all these capabilities, the only one new to consider for implementation is the above definition of FKs by SIR-layer. It appears simple to add to the canonical one. As stated within Section II.C, to find whether an attribute is a natural FK, kernel’s meta-tables, e.g. SYSTABLES, should suffice. Same is valid for determining whether a declared FK is a PKN one and for locating for every FK, every SA to become the source for the NI within the basic natural SIR. Likewise, exploiting the meta-table(s) for views, e.g., SYSVIEWS, should suffice for every compound natural SIR. Both meta-tables should suffice consequently for any other SIR with FKs, with CAs in particular. The canonical implementation itself should be also simple to conceive, [1], [5]. Altogether, to reuse typical present schemes of SRs with FKs as the implicit schemes of natural SIRs, becoming view-savers for LNF queries and CAF queries for every CA declared as a VA would be, we recall, appears simple, perhaps surprisingly simple. It appears also simple to extend this processing to the implicit schemes on SIRs with FKs and explicit IAs, CAs other than VAs especially. Recall that SIRs with the latter become SIRs and view-savers for CAF queries involving such CAs as well.

VII CONCLUSION

On a SIR-enabled DBS, a typical present scheme of a stored relation R with FKs dealt with as proposed, defines a natural SIR R. LNF queries to base tables at no cost whatever for the DBA are the bonus. Also, SIRs with only some PKN FKs or with CAs, still provide for LNF or CAF queries, through Create Table R substantially less procedural than possible till now. This is also a bonus for the DBA.

Next, it appears easy to extend the main relational DB design method to SIRs. An overhaul of the Normal Forms emerges then. The decades old practice of VAs makes this overhaul long overdue. DB textbooks authors should take notice.

Finally, it looks simple to add the proposed processing of FKs to the SIR-layer implemented as proposed till now. Altogether, one may conclude that typical relational DB schemes with FKs were not read as they should be from the very inception of the relational model. LN with its often felt dreadful, joins between base tables, within otherwise simple queries uselessly bothered generations. Likewise did the need for views to alternatively offset this shortcoming. Same for the present limitations of CAs in base tables either restrained to VAs or requiring views as well. Major DBs should become SIR-enabled “better sooner than later”. Making LNF & CAF queries to the base tables the standard, at last. It will be a long overdue service to every SQL client, likely in millions these days.

ACKNOWLEDGMENTS

We are grateful to Ron Fagin for invitation to present this material at IBM Almaden Research Cntr., March 2020. We thank also Berthold Reinwald and C. Mohan for helpful comments. Likewise, we thank Darrell Long for his March 2020 invitation to talk about at UCSC Eng. as well and to Sheldon Finkelstein for inspiring remarks.

REFERENCES


[18] Heath, I., J. 1971. Unacceptable file operations in a relational data model. LN with its often felt dreadful, joins between base tables, substantially less procedural than possible till now. This is also a bonus for the DBA.


[22] Litwin, W. Vigier, Ph., 1986. Dynamic attributes in the multidatabase model. LN with its often felt dreadful, joins between base tables, substantially less procedural than possible till now. This is also a bonus for the DBA.

Create Table S (
  S# Char 5,
  SNAME Char 30,
  STATUS Int,
  CITY Char 30,
  Primary Key (S#));
Create Table P (  P# Char 5,
  PNAME Char 30,
  COLOR  Char 30,
  WEIGHT Int,
  QTY Int,
  Primary Key (P#));
Create Table SP (  S# Char 5 {SNAME, STATUS, S.CITY}
  P# Char 5 {PNAME, COLOR, WEIGHT, P.CITY}
  QTY Int
  {From (SP_.Left Join S On SP_.S#=S.S#) Left Join P On SP.P#=P.P#});

Fig. 2: S-P1 scheme with explicit SP scheme. The implicit one would be that of S-P.SP, outside the brackets {} here.

Table S
<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
</tr>
<tr>
<td>S3</td>
<td>Blake</td>
<td>30</td>
<td>Paris</td>
</tr>
<tr>
<td>S4</td>
<td>Clark</td>
<td>20</td>
<td>Athens</td>
</tr>
<tr>
<td>S5</td>
<td>Adams</td>
<td>30</td>
<td>Athens</td>
</tr>
</tbody>
</table>

Table P
<table>
<thead>
<tr>
<th>P#</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Nut</td>
<td>Red</td>
<td>12</td>
<td>London</td>
</tr>
<tr>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>17</td>
<td>Paris</td>
</tr>
<tr>
<td>P3</td>
<td>Screw</td>
<td>Blue</td>
<td>17</td>
<td>Rome</td>
</tr>
<tr>
<td>P4</td>
<td>Screw</td>
<td>Red</td>
<td>14</td>
<td>London</td>
</tr>
<tr>
<td>P5</td>
<td>Cam</td>
<td>Blue</td>
<td>12</td>
<td>Paris</td>
</tr>
<tr>
<td>P6</td>
<td>Cog</td>
<td>Red</td>
<td>19</td>
<td>London</td>
</tr>
</tbody>
</table>

Table SP
<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>S.CITY</th>
<th>P#</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
<th>P.CITY</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
<td>P1</td>
<td>Nut</td>
<td>Red</td>
<td>12</td>
<td>London</td>
<td>100</td>
</tr>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>17</td>
<td>Paris</td>
<td>200</td>
</tr>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
<td>P3</td>
<td>Screw</td>
<td>Blue</td>
<td>17</td>
<td>Oslo</td>
<td>400</td>
</tr>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
<td>P4</td>
<td>Screw</td>
<td>Red</td>
<td>14</td>
<td>London</td>
<td>200</td>
</tr>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
<td>P5</td>
<td>Cam</td>
<td>Blue</td>
<td>12</td>
<td>Paris</td>
<td>100</td>
</tr>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
<td>P6</td>
<td>Cog</td>
<td>Red</td>
<td>19</td>
<td>London</td>
<td>100</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
<td>P1</td>
<td>Nut</td>
<td>Red</td>
<td>12</td>
<td>London</td>
<td>300</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>17</td>
<td>Paris</td>
<td>400</td>
</tr>
<tr>
<td>S3</td>
<td>Blake</td>
<td>30</td>
<td>Paris</td>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>17</td>
<td>Paris</td>
<td>200</td>
</tr>
<tr>
<td>S4</td>
<td>Clark</td>
<td>20</td>
<td>London</td>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>17</td>
<td>Paris</td>
<td>200</td>
</tr>
</tbody>
</table>

Fig. 3: S-P1 content. IAs are Italic. S-P1.SP is the natural SIR for S-P.SP.

### Example SQL Commands

```sql
Create Table S (  S# Char 5,
  SNAME Char 30,
  STATUS Int,
  CITY Char 30,
  Primary Key (S#));
Create Table P (  P# Char 5,
  PNAME Char 30,
  COLOR  Char 30,
  WEIGHT Int,
  QTY Int,
  Primary Key (P#));
Create Table SP (  S# Char 5 {SNAME, STATUS, S.CITY}
  P# Char 5 {PNAME, COLOR, WEIGHT, P.CITY}
  QTY Int
  {From (SP_.Left Join S On SP_.S#=S.S#) Left Join P On SP.P#=P.P#});
```