Natural Stored and Inherited Relations
Witold Litwin
Université Paris Dauphine PSL
witold.litwin@dauphine.fr

ABSTRACT
A stored and inherited relation (SIR) is a 1NF stored relation enlarged with inherited attributes (IAs). We show that one may consider every usual scheme of a stored normalized relation R with foreign keys as implicitly defining a natural SIR R. For every foreign key, all the attributes of the referenced relation other than the referenced key become IAs of SIR R. Each IA inherits the name and every value of the referenced attribute where foreign key value equals a referenced one. IAs cannot introduce any normalization anomalies, while a typical select-project-join query to the stored (only) R at present, becomes select-project-only towards natural SIR R. In popular terms, such queries become logical navigation free (LNF), without any additional data definition work by the database administrator. We show how discussed stored relation schemes define natural SIRs. We show also that making a typical relational DBS SIR-enabled should be simple. Preexisting relations with foreign keys may be easily converted to SIRs, preserving existing applications, while providing for LNF queries for new ones. We conclude that since inception of the model, relational DB schemes have not been read as they should be or, at least, could be, i.e., as defining natural SIRs. Logical navigation within otherwise simple SQL queries bothered generations uselessly. We postulate every major DBS becoming SIR-enabled “better sooner than later”.

1. INTRODUCTION
The relational model as defined by Codd has two 1NF constructs, [C69], [C69a]. A stored relation consists of stored attributes (SAs). One cannot calculate values of these through the relation scheme. A view consists of inherited attributes, (IAs). These are calculated through a stored query, called view scheme, from SAs or previously defined IAs. Originally, one supposed IAs calculated only. Later, it appeared sometimes practical to also store view snapshot, refreshed whenever needed. Such views were named materialized, [HO1], [GL01], [LZ07]. We proposed recently an enhancement to this model and, consequently SQL, with 3rd construct we called stored and inherited relation (SIR) [L9], [L9a], [L9b]. The concept roots in [LKR92], part of the trend to harness inheritance in relational DBs. E.g., see [S96] or Postgres, [P], or [D6] for later proposals. [LKR92], part of the trend to harness inheritance in relational DBs. E.g., see [S96] or Postgres, [P], or [D6] for later proposals.

Recall that the stored relations of an SQL DB suffice for every view where every attribute has the same proper names as in C-view, but may have a different prefix, a so-called Q-view in [L9]. When it exists, a Q-view may suffice for SQL queries in practice, while its Create View may be less procedural than of C-view [L9]. Likewise, any C-view or Q-view is more procedural to maintain.

The general advantage of SIRs over views is that for every IE, there is always a way to be less procedural than Create View for its C-view, as well as any other Create View that would define mathematically the same relation, [L9]. This remains true even for every view where every attribute has the same proper names as in C-view, but may have a different prefix, a so-called Q-view in [L9]. When it exists, a Q-view may suffice for SQL queries in practice, while its Create View may be less procedural than of C-view [L9]. Likewise, any C-view or Q-view is more procedural to maintain.

Recall that the stored relations of an SQL DB suffice for every SQL query to the DB. DBA may nevertheless elect to create additional views to make less procedural queries to some relations. Such a view typically presents the stored relation to queries as if it had additional attributes. This is actually what every C-view does. The additional attributes serve the goal basically in two ways:

(i) Queries to the view become typically logical navigation (LN) free (LNF). In other terms, the additional attributes make queries typically select-project only, instead of select-project-join ones, over different stored relations. The latter are sources of the additional IAs in the view, making their logical copies. A typical query to the view can then be LNF, projecting simply the view on values selected in some IAs sourced in the relation views. We suppose view R to inherit all these attributes and their values in particular from the stored relation R enlarged to SIR R. In SQL, a view cannot share the name of a stored relation. We suppose therefore that for every view R, one renames stored R to R_ by default. View R as defined always exists. We termed it C-view, for conceptually equal (to SIR R) view, [L9]. Every scheme for IAs of SIR R, called inheritance expression (IE) is basically that of the projection of C-view R on all and only IAs of SIR R. In SQL augmented as we proposed for SIRs, [L9], one may define every IE as it would be in Create View R for C-view R. We termed every such IE explicit.

We define furthermore every SIR R, through Create Table R for all and only SAs intended for SIR R, augmented with explicit IE. The name R refers then to the stored relation formed by the SAs of the SIR R itself, called base of SIR R. Every SQL C-view R scheme fixes in particular the attribute order. It imposes in this way the same one for the IAs and SAs of SIR R. The entire IE, i.e., including From clause of C-view R, should in particular precede all, so-called usually, table options of Create Table R. We recall that the latter define for every stored R, the primary or foreign keys in particular. Every such Create Table is for a SIR-enabled DBS only of course, i.e., for a (yet hypothetical) DBS supporting SIRs, [L9].

For some explicit IEs, additional rules in [L9], specific to SIRs may make the explicit IE partly implicit. Somehow like * may render implicit attribute names in an SQL statement. A less procedural, i.e., requiring fewer keystrokes, implicit IE results from. We suppose as usual, e.g., as for a statement with **, that the knowledge of an implicit IE implies also that of the explicit one. Every implicit IE is transparently processed into an explicit one, while one issues Create Table or Alter Table with an implicit IE.

being extended and in some additional IAs. Without the view, the equivalent query needs to access all the sources instead, through some LN towards the latter IAs thus.

(ii) The additional attributes avoid some value expressions in the queries. These expressions can be arithmetic with scalar functions. They can be also with aggregate functions, perhaps defined through sub-queries. In both cases, the view defines the additional IAs accordingly. Queries may simply invoke their names.

The additional attributes would be typically inconvenient to manage if they were within the stored relation. The normalization anomalies would always appear and possibly the need for frequent updates. Both problems do not occur for any views with IAs calculated only.

Every C-view defines the IE of the SIR with the same SAs as those that C-view inherits and expands. These SIRs clearly also may attain goals (i) or (ii). As said already, the IE can be then always less procedural than the C-view. This is our main rationale for SIRs. Notice that some SIRs are actually already there for that reason since decades. These are SIRs providing for the 1st case of goal (ii) above. The IE defines there every IA so that it could be a computed (dynamic, virtual…) attribute on a present DBS with that capability, e.g., MySQL, Oracle, starting from Sybase in eighties. Every such IE may be made implicit so that it is no more procedural than the present schemes of those attributes or simply may be the latter schemes.

For more on computed attributes and the rationale for, see countless tutorials or early related research, e.g., [LV86]. Of course no one realized in this material that whatever is the present DBS, all relation with computed attributes possible there are in fact specific SIRs.

In [L9b] we pointed out that one may attain (i) in particular for SIRs with foreign keys. The IAs should inherit then all or at least some of the SAs or IAs in the referenced relations. It appeared also that for some SIRs, one should be even able to infer the explicit IEs only from the present stored relation schemes. In other words, the implicit IE is then empty, i.e., has zero procedurality. Consequently, not only queries become typically LNF, but, also, to define the IE does not bother the database administrator (DBA) whatsoever.

We now analyze such SIRs in depth. We qualified them of natural. We term natural every IA of a natural SIR as well. A referenced attribute can be a natural IA itself, etc. The exceptions are IAs that would result from a circular, i.e., recursive, referencing.

In what follows, we first revisit the concept of foreign key and adapt it to SIRs. Next, we define natural SIRs, as well as the rules for generating an explicit scheme for every implicit one of a natural SIR. We recall that we assimilate every usual scheme of a stored relation with foreign keys to the latter. Next, we show more in depth how natural SIRs avoid LN without any work for DBA. We then discuss altering natural SIRs. Finally, we show that making any a major DBS SIR-enabled, including the capabilities proposed here, should be easy. We conclude that every popular DBS should become SIR-enabled “better sooner than later”.

2. FOREIGN KEYS REVISITED

Despite being fundamental to the relational model, the concept of a foreign key seems surprisingly imprecise, [D9]. For our purpose, we call foreign key an attribute, perhaps composed, such that every of its values could be that of some uniquely chosen (stored) primary key, unless the Foreign Key constraint references a (stored) candidate key. The former is the definition by Codd, maintained in all his papers of our knowledge. The latter is a possibility on some popular DBSs, e.g., MySQL. It follows that both, the foreign and, basically, the primary key, share a domain, perhaps composed as well. The latter key is usually called referenced key. The foreign key can implicitly share the proper name with the referenced one. This is typical for atomic keys, less usual for composite ones. For the former, we speak about natural foreign and referenced keys. Composite foreign keys seem less “natural”. They are by far less frequent and, when they are the only choice with existing data, many prefer to add an atomic primary key, often called surrogate. Alternatively to implicit name sharing, one can explicitly designate the referenced key. This is necessary if more than one primary key with the proper name of the foreign key exists or the referenced key has a different proper name. We talk then about declared key. For the above discussed reasons, we also consider that every composite foreign key or not referencing a primary key has to be a declared one. Finally, a foreign key, as just defined, does not imply the referential integrity. Neither did the original relational model, claiming it definable only, [C69a]. But for SQL, hence generally at present, every foreign key must be a declared one, through Foreign Key clause of course. This one guarantees then also the referential integrity somehow.

Behind the formal definition of foreign keys, the idea is the conceptual inheritance of all the attributes of the referenced relation by the referencing one. The typical inheritance of the name of the referenced key by the foreign one symbolizes it. Codd proposed the referential integrity to be only definable somehow. SQL Foreign Key clause made it nevertheless the characteristic property of every foreign key. As it will appear, some DBSs, e.g., MS Access, let it even to be circular. The inheritance cannot be so within the relational model. The recursion that would result from is not in the model. In particular, all popular relational DBSs prohibit circular views, i.e., cyclically referencing each other.

For SIRs, we therefore consider that, first, the referenced and the foreign key may be natural ones. It was so for the original relational model, we recall, see e.g., the “biblical” S-P DB at Figure 1. The SQL Foreign Key clause, extended to SIRs, is required for a foreign key only when the referenced key has a different name or the keys are composite or referential integrity is a must. Technically, as mentioned already, we express the conceptual inheritance through a foreign key, whether natural or declared, of every attribute of the referenced relation beyond the referenced key, by creating for each such attribute, a specific IA within the referencing relation. We discuss these IAs more below.

To deal specifically with inheritance through foreign keys, we introduce an additional option for Foreign Key clause in the implicit SIR scheme, i.e., Create Table in practice. We call it Inheritance and suppose it can have two values. The 1st one written: Inheritance Only, implies no referential integrity. Recall that the relational model considered that it may be unwelcome, as we stressed already. The 2nd value, written: Inheritance No Action, implies the referential integrity only. This choice may in particular prevent the circular inheritance just mentioned. Without Inheritance option, we consider that Foreign Key clause implies by default for SIRs both: inheritance and integrity, as at present.

Example 1. Consider the “biblical” S-P DB at Figure 1, with suppliers and parts in M:N relationship, allegedly the original DB of the relational model, [D04]. S-P is our motivation for most of examples. Here, SP.S# and SP.P# have the same proper names as S.S# and P.P# respectively. The original verbal description of S-P supposes also that each pair shares the domain. Each pair, i.e., (SP.S#, SP.P#) and (S.S#, P.P#), constitutes thus natural foreign and referenced keys. Both pair
provides therefore for inheritance by SP that we detail below. At the figure, both pairs respect also the referential integrity. However, the constraint is not declared. Hence, e.g., an insertion of (S1, P7 200) destroying that integrity for P and SP remains valid. Also, P6 may change to P60 in P if a need occurs, without impacting or being impacted by use of P6 in SP. Otherwise, e.g., if such a change required or implied same change to SP, a client searching for QTY of P6 supplied by S1, could get heart attack learning that such supply never existed.

If the referential integrity is nevertheless required for a pair, e.g., (SP.S#, S#.S), one should declare in Create Table SP through the usual Foreign Key (S#) References (S#.S), with usual On Delete and On Update options perhaps. The clause would provide then both: inheritance and referential integrity. Likewise, if DBA changes SP.P# to Part#, the usual Foreign Key (Part#) References (P.#) would provide both: inheritance and referential integrity.

The inheritance we mentioned above follows the obvious conceptual one through the natural foreign keys in SP, of every attribute of S and of P other than referenced keys. E.g., for every supply, its supplier identified with SP.S# has also a name, status and city. Which are in fact those it has as supplier within S for S.S# = SP.S#, i.e. the same natural foreign and referenced keys.

We show in next example how this inheritance impacts explicit Create Table SP for natural SIR SP. Similar observations apply to parts. This is what makes sense at present of typical queries to SP and S or P with the LN through S.S# = SP.S# or P.P# = SP.P# equijoins. E.g., consider the query for the name for every supplier of screws.

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>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>1</td>
<td>Smith</td>
<td>London</td>
<td>1</td>
<td>P1</td>
<td>Blue</td>
<td>2000</td>
<td>Paris</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>Paris</td>
<td>2</td>
<td>P2</td>
<td>Red</td>
<td>1500</td>
<td>Paris</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>Clark</td>
<td>London</td>
<td>3</td>
<td>P3</td>
<td>Black</td>
<td>2000</td>
<td>London</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>Adams</td>
<td>Athens</td>
<td>4</td>
<td>P4</td>
<td>White</td>
<td>1200</td>
<td>Athens</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>Davis</td>
<td>Rome</td>
<td>5</td>
<td>P5</td>
<td>Green</td>
<td>450</td>
<td>Rome</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 1. “Biblical” S-P database.

3. NATURAL SIRs

3.1 The Concept

Consider a stored relation R with foreign keys referencing keys in other relations. Suppose also SIR R augmenting R only with IAs sourced in the referenced relations and as follows. First, for every referenced relation R’, for every attribute of R’ other than the referenced key, there is one and only one IA is in SIR R, with the same proper name as in R’. IA name can be qualified if the need occurs. Next, for an SQL DB, where the order of attributes matters, for every R’ with referenced key being atomic and primary, we suppose every IA placed with respect to the foreign key as its source is with respect to the referenced key. Likewise, for every other referenced key, all the IAs in their original order should follow the last attribute of the foreign key. Next, for every foreign key value, we suppose the IA values all these for the same referenced key value. If the latter does not exist, we set to null every corresponding IA in the tuple.

In less technical words, we may say that for our SIR R, we consider every stored foreign key value as an abbreviation of all the values it references. Every value other than that of the referenced key is implicit only in the stored relation R, with a foreign key we expand to a natural SIR R. In the latter, each such value becomes a single IA value. IE précises how every such value should be calculated upon a need. Somehow like we use here the abbreviation LNF for Logical Navigation Free, with every upper case letter serving of foreign key towards the entire word it implicitly references and that we “calculate” upon the needs.

For every natural SIR, we term it also natural for stored R it augments, as well as for base R of SIR R the former becomes. Finally, we recall that we qualify of natural every IA of SIR R and its IE.

It’s easy to see that for every stored R with foreign keys, hence for every base R, as well; we have only one natural SIR R. Next, suppose that for the scheme of some stored relation R with foreign keys we can determine somehow its natural SIR R explicit scheme. We consider then R scheme as the implicit one of SIR R. Notice that it will be also the R sub-scheme in the explicit one, except for the relation name itself. In such an implicit scheme, we consider in particular every foreign key as, also implicitly, representing every natural IA of SIR R it contributes to, as we discussed.

In addition to every scheme that a stored relation with foreign keys could have at present, we recall that the implicit scheme of a SIR can have Foreign Key clauses with above defined Inheritance option. It is easy to see that there is no need in contrast for these options in the explicit scheme. The inference rules we propose below produce the natural explicit Create Table from every implicit one as just defined. We formulate these rules after the next motivating example.

Example 2. In the wake of Example 1, recall that in S-P, each pair: (SP.S#, and S.S#) and (SP.P#, and P.P#) constitutes the natural foreign and referenced keys. The discussed inheritance by SP from S and from P follows. Suppose now that we augment the original SP to SIR SP with the following IAs, designated in Italics:

(1) SP (S#, SNAME, STATUS, S.CITY, P#, PNAME, COLOR, WEIGHT, P.CITY, QTY)

In (1), every IA is uniquely named as some non-key attribute of S or of P. There is also one IA per every such attribute and SP has no other IAs. Furthermore, suppose that for every SP.S# value from S, for every IA sourced in S, the value is as for S.S# = SP.S#. Likewise, consider that for each SP.P# value, every IA value from P, is as for the same P.P# there. If there is no referenced S.S#, then suppose we set all IAs from S to nulls. We do the same for every missing P.P# value. Finally, observe that SIR SP (1) has every IA placed with respect to its foreign key as its source is with respect to the referenced one. All these properties make SIR SP a natural one. It is thus also the natural SIR SP for its base SP, and for the original stored (only) SP. By the same token, the attributes SNAME...S.CITY, PNAME...P.CITY are all natural. Figure 3 shows the content of SP (1), given that of S-P at Figure 1. IAs are Italics.

We now operationally define the explicit SQL scheme of SIR SP (1), i.e., Create Table SP, valid for SIR-enabled DBs only, of course. We follow the general rules we proposed earlier for every Create Table for a SIR, recalled in the Introduction. As said, one may start with Create View SP for C-view SP. Recall that in SQL, no Create View SP can refer to a relation named SP. Hence, suppose that one renames the “biblical” SP to SP.. Then suppose the following statement to define C-view SP:

(2) Create View SP AS (SP_.S#, SNAME, STATUS, S.CITY, SP_.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#));

It is easy to see, from values within Figure 1 in particular, that (2) effectively creates logically the same relation as intended for
SIR SP (1). Next, suppose the following Create Table SP for the original SP:

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

The following Create Table SP is then the explicit scheme for SIR SP (1), in SQL extended for SIRs as proposed:

(4) 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#) Primary Key (S#, P#));

Scheme (4) indeed merges (2) and (3) to get every SA and IA in (1) in order. IE consists of every IA in (1) and of From clause in (2). In what follows, we designate DB with SIR SP (4) as S-P1. Figure 2 shows S-P1 schemes, with the implicit one, i.e., (3), highlighted in bold within the explicit SP. Figure 3 shows S-P1.SP content for S-P.SP at Figure 1.

Notice that as said generally before, (3) became also in (4) SQL scheme of the base SP_ . except for the relation name. We suppose From clause in (4) to refer to that one. The goal of (4) is to augment SP_ after all. In (2), SP_ designates instead “classically” a separate relation, i.e., S-P.SP renamed to SP_. Not any part of the relation being created, unlike in (4). We allow for such referencing in SIRs in general. It avoids indeed sometimes, like in (4) in fact, a (prohibited) circular referencing among SIRs, [L9]. Since R_ is a stored relation, it inherits indeed from nothing, R in particular. Except for this formal difference, both above SP_ schemes define the same relation, including the physical implementation, data types only here.

The From clause in (4) calculates hence the same value or null for every IA augmenting every stored tuple of base SP_ of SIR SP, as the same clause calculates for the same IA in the (only) tuple containing the same sub-tuple (S#, P#, QTY) in view SP (2). Notice also that IE effectively precedes the table options in (3). Those define the primary key only, since both SP_.S# and SP_.P# are natural foreign keys. Observe finally, as an exercise, that all the values calculated through IE (4) are effectively those we intended for SIR SP. Then check with the figures. Observe furthermore that IE in (4) is less procedural than Create View (2), as claimed for every SIR R with respect to C-view R, [L9]. The easy to grab rationale is that (2) had to redefine every IA that remains an SA in S-P1.SP_. More precisely, suppose the procedurality p of (2) measured as the minimal number of characters or keystrokes to type-in. We have then p = 152 for view SP (2) and p = 116 for IE in (4). So C-view SP scheme is 30% more procedural than IE. On a SIR-enabled DBS, SP (4) would provide for LNF queries as we show soon. C-view SP (2) would do as well. At any present DBS, not supporting SIR SP thus, (2) would be the only choice. Time to type is in general at least linear with p. As a “view saver”, our SP (4) would thus save 30% of the time necessary for keystroking Create View SP. In popular words, to create in SQL view SP when SIR SP could exist, would be just a consequent waste of time. As we signaled, the property of lesser SP could exist, would be just a consequent waste of time. As we signaled.

Summing up, stored SP scheme (3) can indeed the implicit scheme of SIR SP, leading to (4) as the explicit one. Like is SP_ scheme within (4) as well, except for its implicit default stored relation name. See again Figure 2.

3.2 Inferring Explicit Natural Schemes

As usual, we suppose every referenced relation already declared when the referencing one is being created. Suppose then that a SIR-enabled DBS gets Create Table R that defines stored attributes only, hence may be an implicit scheme of a natural SIR. DBS processes it towards the explicit natural scheme as follows.

1. Process the declared foreign keys. For every declared foreign key without Inheritance No Action option, if there is any, complete Create Table R with the unique in R name of every IA required for the natural SIR R. Then, remove every Foreign Key clause with Inheritance Only option if there is any. Also, if there is any, remove every Inheritance No Action option, from its Foreign Key clause.

For every declared referenced key, perhaps composite, retrieve from meta-tables, e.g., SYSSESTABLES etc. for DB2, the name of every attribute other than the name of any attribute within the referenced key. Place each of these names, qualified if needed, in Create Table R, following the order on IAs defined for a natural SIR. DBS processes it towards the explicit natural scheme as follows.

Process the natural keys. Retrieve from appropriate meta-tables, e.g., SYSSESTABLES etc. for DB2, the name of every relation in the DB with the atomic primary key, together with the name of the key.

Suppose R is such a relation with primary key C. For every R’, find whether R also has an attribute named C that is not an attribute of a declared foreign key. If so, complete Create Table R with the unique in R names of every IA sourced in R’ and required for the natural SIR R. Follow the ordering of IAs of a natural SIR.

3. Create From clause. To start, after last stored attribute and before the table options, insert: From R_. Then, for every R’ above processed, let K1…Kn; n ≥ 1 ; be the referenced (key) attribute(s) and F1.Fn the referencing ones. For every R’, append to current form of From clause: Left Join R’ On (F1 = K1 And…Fn = Kn). Example 3. We skip the easy but tedious proof of the rules. We only show how they build the explicit Create Table SP (4) from (3). Suppose S-P1.S and S-P1.P already created. There is no declared foreign key in (3), hence rule (1) does not apply. Rule 2 finds S.S# and P.P#. For the former, Rule 2 inserts to (3) SNAME...S.CITY, right after S#. Likewise, it inserts PNAME...P.CITY right after SP.P#. Supposing S.S# found first by Rule 2, Rule 3 builds From clause as in (4) and terminates the build-up. Reverse order for Rule 2 would reverse Left Joins. Result would be mathematically the same, as these joins are commutative.

For DBA, the rules mean simply p(IE) = 0. In other words, e.g., to create the natural SIR SP instead of S-P.SP costs DBA zero additional work and time. As free bonus, DBA saves also 100% of procedural cost of view SP. Simply put, on SIR-enabled DBS, to create view SP instead of the natural SIR SP to gain typically simpler LNF queries for clients, would be for DBA a total waste of time. We now detail these bonuses for both clients and DBA.

3.3 Natural SIRs as LN & View Savers

We detail the bonus for SP only, but the results clearly extend to any natural SIRs. The IAs in (4), are the attributes that S-P.SP conceptually has as well. But, as said, operationally does not. As stored SP, it is indeed constrained to its normal form by the model. The well-known goal is to avoid the (normalization)
anomalies, [D2]. Anomalies in contrast do not occur for an IA, [L9b]. E.g., recall that if SNAME was within stored SP, then, first, it would denormalize it. Indeed, since we have FD S# -> SNAME, SP would no more be in 2NF even. Then, the denormalized SP would require, e.g., six times more storage for SNAME = ‘Smith’ than the normalized S for its single ‘Smith’, Figure 1. Next, to update Smith to, say, Smit, would require I update only in S, but 6 in the denormalized SP. If even one of these failed, FD S# -> SNAME would be violated with obvious drawbacks for queries. If SP.SNAME is in contrast a natural IA, none of these anomalies occurs. Similarly, for every IA sourced in any other non-key attribute of S and P. SP can have thus all the attributes it should, but could not have with the current model.

A typical query to stored SP formulates as a select-project-join query. It searches for attributes in SP, but also for some conceptual attributes that ended up elsewhere. E.g., in real life, most queries about a supply require not only some SP data, but also the name of supplier(s) and those of part(s) concerned. The one only. Every attribute value in S or P possibly needed is now most queries about a supply require not only some SP data, but conceptual attributes that ended up elsewhere. E.g., in real life, indeed also within SP. LN does not only always increase query. It searches for attributes in SP, but also for some even for pros, since decades, [J90], [D91].

provides for the same LNF queries. As shown however (2) is formulation for stored SP would be query Q as follows or an atomic) transaction to the kernel that will (a) rename SP to SP_, and (b) create view SP (2). From now on, SIR-layer will transparently direct every query to SP, towards either SP or SP_, as above discussed. Every preexisting query, say query Q above, will provide the same outcome. The equivalent query by a new application could be LNF in contrast, e.g. could be Q’ above. No SP tuple would ever get affected by the upgrade.

4. ALTERING NATURAL SIRs

One can alter any natural SIR R through every Alter Table R clause for SIRs in [L9]. We add Foreign Key with Inheritance option to all this. Every SIR R or even a stored only R altered with a Foreign Key clause with or without that option, starts referencing a relation not yet referenced. That one was perhaps created after R and should be referenced since. Alternatively, DBA needed the clause as s/he wished to upgrade stored R preexisting on a DBS rendered SIR-enabled. In every case, the clause adds to the existing scheme the IAs determined as above discussed. We recall that, for every preexisting relation altered in this way, free bonus of typically LNF queries follows. The upgrade does not change or move any content neither affects any preexisting applications. We show it in next section.

Example 4. Suppose that DBS managing S-P becomes SIR-enabled. Existing S-P applications may continue to function as before. Suppose nevertheless that S-P clients wish LNF queries to SP for the future ones. The following Alter Table upgrades SP to S-P1.SP (4), without any referential integrity thus:

(5) Alter Table SP (Add Constraint Foreign Key (S#) References S(S#) Inheritance Only, Add Constraint Foreign Key (P#) References P(P#) Inheritance Only);

New queries to SP become now typically LNF. The existing ones continue acting as before. The alteration preserves the existing (stored) SP content, e.g., that at Figure 1. Example 5 below details this point.

5. IMPLEMENTING NATURAL SIRs

[L9] proposed the implementation for SIR-enabled DBS that we term canonical here. In sum, the latter consists of a (canonical) layer above an existing DBS, called SIR-layer, interfacing all clients and DBA while, internally, calling services of the DBS, referred to as kernel (DBS). Above SIR-layer, the relational constructs are: stored relations, SIRs and views. Underneath, i.e., for the kernel, there are necessarily stored relations, perhaps with computed attributes and views only. For every stored relation or a SIR with IAs declared as computed attributes or a view, SIR-layer simply forwards Create statement to the kernel. Then, it also forwards every query to these constructs only. In contrast, for every SIR R other than above, SIR-layer atomically creates within the kernel the couple: stored relation R and C-view R. Create Table R and Create View R are easily extracted by SIR-layer from the explicit Create Table R. SIR-layer transparently forwards then every Select query addressing SIR R to view. Kernel DBS may internally optimize somehow such queries, e.g., by materializing some IAs within view R for faster joins, [GL01], [H01] or [V87]. SIR-layer passes gains to SIR R in this way. Update queries to SIR R go to view R or to base relation R, or are refused, depending on view updating capabilities of kernels, [D2a]. E.g., MySQL provides for updates of outer join views, while SQL Server does not. SQLite flatly refuses every view update, etc.

With respect to the canonical implementation defined in [L9], the only new capability required from SIR-layer for natural SIRs, is processing of implicit schemes through the proposed rules. This appears simple to add. The canonical implementation itself appears also simple to conceive, [L9]. See also [L6] for a simulated implementation of the latter. Altogether, to reuse present schemes of stored relations with foreign keys for natural SIRs appears simple, perhaps surprisingly simple. The old dream of LNF typical queries comes as welcome bonus.

Example 5. Suppose that S-P as at Figure 1 was created on some DBS becoming later the kernel for some SIR-layer supporting natural SIRs, is processing of implicit schemes through the proposed rules. This appears simple to add. The canonical implementation itself appears also simple to conceive, [L9]. See also [L6] for a simulated implementation of the latter. Altogether, to reuse present schemes of stored relations with foreign keys for natural SIRs appears simple, perhaps surprisingly simple. The old dream of LNF typical queries comes as welcome bonus.

Example 5. Suppose that S-P as at Figure 1 was created on some DBS becoming later the kernel for some SIR-layer supporting natural SIRs, is processing of implicit schemes through the proposed rules. This appears simple to add. The canonical implementation itself appears also simple to conceive, [L9]. See also [L6] for a simulated implementation of the latter. Altogether, to reuse present schemes of stored relations with foreign keys for natural SIRs appears simple, perhaps surprisingly simple. The old dream of LNF typical queries comes as welcome bonus.

6. CONCLUSION

On a SIR-enabled DBS, every present scheme of a stored relation R with foreign keys may define in fact a natural SIR R. No more additional data definition work for DBA to create the latter. Typically LNF queries to base tables are the bonus for clients. By the same token, DBA saves the work on views necessary for the same queries at present. Decades old burden for generations of relational DB clients or DBAs otherwise is over.

To add the proposed processing of foreign keys to the canonical implementation of SIR-layer previously proposed appears
simple. The canonical implementation appeared simple to realize already. Altogether, it appears that, that since the model was introduced, relational DBs schemes have not been read as they should be. No one realized that usual schemes of base relations with foreign keys were abbreviated ones of natural SIRs in fact. Major DBs should become SIR-enabled “better sooner than later”. It would be a fundamental and long overdue service to, likely, millions of clients of SQL DBs of all kinds and flavors at present.

Future research work should first aim on the proof-of-concept implementation of SIR-layer supporting natural SIRs. Besides, one may generalize our rules for implicit schemes of natural SIRs to other SIRs with foreign keys. One should also generalize present relational design to DBs with SIRs. E.g., it is easy to see that even the sheer concept of normal forms should evolve. That need was in fact already observed for relations with computed attributes, [D9a]. Recall that those are specific SIRs (see [L9] for more on it). Early results on these goals are in [L0], together with more material on natural SIRs that restricted size of this paper prohibited to include.

ACKNOWLEDGMENTS.
We are grateful to Ron Fagin for invitation to present this material at IBM Almaden Research Cntr., March 2020. Likewise, we thank Darrell Long for his March 2020 invitation to talk about at UCSC Eng. as well. We thank finally Peter Scheuermann for helpful comments.

REFERENCES

[S-P1 Scheme]
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, 
  CITY Char 30, 
  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#), 
  Primary Key (S#, P#));

Figure 2. S-P1 scheme with explicit SP scheme. The implicit one would be that of S-P.SP, bold for SP here.

<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>Nat</td>
<td>Red</td>
<td>12</td>
<td>London</td>
<td>300</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>Nat</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>P3</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>
<tr>
<td>S4</td>
<td>Clark</td>
<td>20</td>
<td>London</td>
<td>P4</td>
<td>Screw</td>
<td>Red</td>
<td>14</td>
<td>London</td>
<td>300</td>
</tr>
<tr>
<td>S4</td>
<td>Clark</td>
<td>20</td>
<td>London</td>
<td>P5</td>
<td>Cam</td>
<td>Blue</td>
<td>12</td>
<td>Paris</td>
<td>400</td>
</tr>
</tbody>
</table>

Figure 3. S-P1 content. IA names and values are Italics,