SIR SQL for Usually Almost Natural Select-From-Where Queries to Base Tables

Witold Litwin[1]

One usually declares SIR SQL base tables as one would do for SQL ones with the same base attributes. However, only the former provides for the logical navigation free (LNF) queries, i.e., queries free of equijoins on Codd's foreign and referenced keys. SIR SQL base tables may also have calculated attributes (CAs), definable in SQL within queries or views only, involving multiple tables, aggregate functions, sub-queries…. Queries to base tables are then free of value expressions defining these CAs, i.e., are CAF, unlike could be queries to SQL base tables. The overall result should usually be almost natural LNF and CAF queries to base tables, substantially less procedural than the same outcome SQL queries to base tables. SIR SQL should accordingly boost the productivity of 7M+ clients of the most used data science language/ It should attract also numerous new clients, confortable with simplicity of LNF and CAF queries, while refraining of SQL otherwise.

Keywords: SQL, Relational data mode, Stored and Inherited Relations

Motto: The king is dead, long live the king!

1. Introduction

SIR SQL generalizes the standard SQL and every known SQL dialect to base tables, called *Stored and Inherited Relations* (SIRs), [Litwin, W. (2019)], [Litwin. W. (2025a)]. Standard SQL or any SQL dialect becomes the *kernel* of SIR SQL dialect generalizing it. E.g., SQLite3 dialect generalizes to SIR SQLite3 dialect with the former as the kernel. The name SIR SQL alone refers by default to the standard SQL as the kernel.

As for SQL, every SIR SQL dialect provides for Data Definition Language (DDL) for DB Administrator (DBA) and Data Manipulation Language (DML) for *clients*. The latter is simply the one of the kernel, although the internal processing differs. In contrast, SQL Create Table and Alter Table generalize so that DBA can create or alter SIRs. Every SIR SQL Create Table R defines some *SIR SQL base table* R. Every R has *base attributes* (BAs) with optional table constraints or options. These are definable in any way they could be within the kernel SQL Create Table. E.g., SIR SQLite3 base table may have any BA, table constraints or options definable in SQLite3 Create Table.

SIR SQL Create Table R let also DBA to define *inherited* attributes, (IAs). R is SIR iff it contains BAs and IAs, we recall. The definition of all the IAs is called *Inheritance Expression* (IA). An IE is *actual* iff one defines every IA as one could do through Create View. Recall that SQL Create Table does not provide for any such attributes. Every SIR SQL base table R has accordingly the *actual* Create Table R (scheme), defining both BAs and the actual IE, whenever there is any. Clients use the actual schemes for updates and queries. Finally, SQL shorthand '*' standing for all the attributes of R in "select R.*…", stands for SIR SQL for all BAs and, if there are any, for all the IAs in the actual R scheme.

We qualify SIR SQL Create Table formulated by DBA of *declared* one, possibly with *declared* IE. Declared Create Table without IE may create a base table with BAs only, i.e., *SQL* base table. Declared IE may be the actual one, but usually it should be shorthand for only. The latter may even consist of specific BAs only, making the declared IE *empty*. Shorthands let DBAs to declare the actual IE less procedurally, most often substantially, especially for empty IEs. SIR SQL interface rewrites every declared shorthand IE to the actual one, hence making the declared Create Table the actual one, for any further processing.

Every SIR R has primary key (PK) that must be BA acceptable as PK for the kernel SQL, e.g., it must be a stored attribute (SA) for SQLite3. The projection on all and only BAs constitutes the *base* of R, named implicitly R_. The name R_ may qualify BAs. Also, updates may use it. Next, as IAs in SQL views, IAs in SIRs, are by default calculable only. They cannot imply any normalization anomalies. Unlike if they were SAs, necessarily for standard SQL base tables.

As for SQL, SIR SQL BA can be a *foreign key* (FK) in some *referencing* base table R. As for SQL again, SIR SQL FK *references* then some *referenced key* (RK). RK must be a key of some (uniquely) *referenced* base table R',

[1] Witold.litwin@dauphine.fr . Latest update: July 15, 2026. Intended originally for plenary talk at 2nd Intl. Conf. on Applied Science, Engineering & Technology (ICAET 2026), Florence.

Usually, R ≠ R' and RK = PK. but some kerdnels let RK be a candidate key. Next, recall that every SQL FK must be declared through SQL *Foreign Key constraint* (noted FK constraint below). By definition, every FK enforces then the referential integrity (RI). As we'll recall as well, SIR SQL FK in contrast may or may not be declared through SIR SQL (specific) FK constraint. As SQL FK constraint, the latter may serve to enforce RI only. However, except for that case, every SIR SQL FK follows the Codd's original FK idea, i.e., is the *logical pointer*, (LP), [Codd, E., F. (1970)]. It is accordingly by default, unless the constraint states otherwise, as we detail later, shorthand for every non-RK attribute of R', say A, considered as a conceptual attribute of R as well. For every R tuple, A has then the value defined through the equijoin on LP = RK or A is null for every tuple such that there is no RK equal to LP value in the tuple. The equijoin is thus, basically, left or right (outer) join preserving every tuple of R. None of the referenced attributes could actually be declared for R in Codd's model. They would necessarily be SAs, implying normalization anomalies, banned by the model. In SIR SQL in contrast, for every such R'.A, there may be IA R.A with the values or nulls defined through the equijoin, Not leading to any normalization anomalies in R, as just stressed.

In the literature, one qualifies an equijoin on R.LP = R'.RK, of *logical navigation* (LN) or of LN join, *through* LP, *from* R *to* R', or of LN clause. These terms transitively apply to any eventual LN from R' to some R'' etc. If SIR R has IAs with names and values defined through LPs, these provide for LN free (LNF) queries, while SQL queries to base tables providing for the same results require LN. Recall that LNF queries have usual SQL format, but From clause and any eventual Where clause, are without any LN. DB literature abundantly shows that SQL queries typically address attributes of some base table R and of some tables referenced by R, requiring LN from R to the latter. SIR SQL provides usually instead for LNF queries to these attributes. The reason is that, as we'll recall, SIR SQL rewrites Create Table R declared with LPs so that the actual IE usually contains all IAs and every LN these LPs imply. SIR SQL query providing for the same outcome as SQL query with LN may then address BAs and IAs of R only, becoming trivially LNF in consequence.

Literature and the motivating examples below show that SIR SQL LNF queries should usually be substantially less procedural than the same outcome SQL ones. The savings on LN should be, at least, in dozens of characters to type-in & debug. Select list of SIR SQL LNF query should usually consist of the names of the selected BAs or IAs only. Then, usual Where clause, if the query has any, should be a Boolean expression on SQL comparisons of attribute names to constants, e.g., A =10 And B < 20…). Finally, From clause should usually formulate as From R only. SIR SQL LNF query should thus usually have almost natural SQL Select-From-Where formulation.

Besides being defined through LN as just overviewed, SIR SQL IAs can be *calculated* attributes, (CAs). One defines CAs through kernel's Create View capabilities for value expressions. These expressions can be arithmetic or with SQL scalar functions, perhaps addressing several tables. They can also contain SQL aggregates, perhaps within sub-queries or can simply define aliases. Recall that no SQL Create Table makes definable CAs. With progress of data science, CA schemes complexify to often dozens if not hundreds of characters to type-in and to dozens of minutes of debugging. CAs in SIR SQL base tables provide for *CA free*, (CAF) queries to these tables. Such queries only name CAs, without defining the value expressions. Unlike can do the same output queries to SQL base tables with the same BAs, but without any CAs necessarily.

As we will show, the overall practical result should be, first, that, as long as there is no CAs, DBA will most often declare SIR SQL base table R as one would declare SQL table R with the same BAs, constraints and options. I.e., it will usually suffice to declare R_ only. Queries to SIR SQL base tables will usually be LNF, unlike to the SQL ones declared through the same statements. CAs will usually cost DBA the procedurality of the value expressions only. Queries addressing CAs, to base tables with, will usually be LNF and CAF. As already mentioned, while keeping the Select-From-Where form, SIR SQL queries should usually be then substantially less procedural than SQL counterparts and almost natural ones. They should thus be also substantially faster to declare and debug. In addition, LN in queries requires learning curve that can be awkward, [Date, C., J., Darwen, H., (1992)]. Same is true for value expressions with sub-queries or aggregates. For casual clients, all this usually makes SQL unusable. Unlike it should be for SIR SQL.

It was also shown that to implement a SIR SQL dialect, one can reuse existing DBS without any changes. It suffices to add a front-end, termed *SIR Layer*. A proof-of-concept prototype in Python for SQLite3 [Litwin, W., (2023)], required two months only. The timing for other popular kernels should be similar.

Below, we first recall the related work. Then, motivating examples recall SIR SQL advantages. Next, we discuss SIR SQL Create Table features not addressed in these examples, the options specific to SIR SQL FK constraint and not published yet, especially. Afterwards we discuss new SIR SQL Alter Table capabilities. Next, we discuss the design principles of SIR SQL DBs and show that they are mainly as for SQL DBs. We then show how to upgrade existing SQL DBs to SIR SQL ones providing for the LNF queries. Finally, we present SIR Layer and its prototype implementation. We conclude that every popular SQL DBS should provide for SIR SQL 'better sooner than later". This should significantly boost the productivity of 7M+ clients, worldwide, of the most used data science language, [Stonebraker, Pavlo, 2023], [Sobolevskiy, M. (2015)]. SIR SQL should also attract numerous new clients, confortable with LNF and CAF queries, while rebutting SQL otherwise.

2. Related Work

With respect to the related work, the problematic of LNF queries to relational base tables is anything but new. The seminal work on universal relation, already in eighties by Maier & Ullman, proposed the, so-called, *universal relation* as a solution for LNF queries. However, despite its popularity, the concept did not prove practical as yet. For SQL DBs specifically, we have shown in [Litwin, W. (2019)] that for every SIR SQL DB, there is always an SQL DB providing for the same LNF and CAF queries. The latter has to have the base tables with the same BAs as SIR bases in SIR SQL DB. In addition, for every SIR SQL base table R with LP FKs, SQL DBA has to declare Create View R defining a specific so-called *canonical* view R, C-view R in short. Every C-view R defines the same actual IE as SIR R, but should also redefine as IAs all BAs of SIR R. The latter should have the same names and always the same values as in every SIR R tuple. *In fine*, both Create Table R with actual IE for SIR R and Create View R for C-view R should have (a) the same attribute name and value expressions list and (b) the same From clause, as well as Where one, if there were any. However, it easily appears that every such Create View R has to be more procedural than the declared IE in SIR SQL Create Table R, usually substantially,. Besides, SQL DBA has to alter every C-view R anytime the base table gets altered, both alterations constituting an atomic SQL transaction. Unlike would usually do SIR SQL DBA for SIR R. Altogether, doubling SQL base tables with C-views apparently appears generally cumbersome enough, to be about unheard of. In particular, we haven't seen the idea even discussed in any of the SQL DB design textbooks, tutorials, discussion groups… *A fortiori* used for actual DBs. Also, queries discussed in the literature or actual ones are about exclusively to base tables. In particular, all popular benchmarks we are aware of, measure queries to base tables only.

Literature shows furthermore that natural LPs are often used and that SQL FKs are usually also LPs, and that actual queries mostly involve LN. Examples in next section will illustrate all this. Next, it shows that the problem of CAF queries to base tables is anything but new as well. Already in eighties, Sybase proposed base attributes that are so-called *virtual* (dynamic, computed, generated…), (VAs). Every VA was calculable through predefined in Create Table value expression using only arithmetic operations and scalar functions over attributes from the base table being created. This is far from the above mentioned capabilities of CAs in views, hence in SIRs. Nevertheless, despite different syntax of value expressions in VAs and in equivalent CAs, VAs still save DBAs from the procedurality of Create View for CAF queries. This is why VAs remain popular since four decades, spreading to several SQL DBSs.

Notice that for any SIR SQL dialects with the kernel providing for VAs, any latter is by definition base attribute and not a CA. Besides, one defines every VA differently from CA with the same values, while for every VA there is such CA. Actually, for SIR SQL base tables, kernel's capability of VAs should in practice be superfluous.

Our own previous work on SIR SQL was published in several papers, listed in [Litwin, W., (2025)]. With respect to that work novelties are, in the nutshell, as follows. Before, declaring Create Table R for SIR R providing for LNF queries, as if R was SQL base table, i.e., without declaring any IAs, as we mentioned, was possible for so-called primary key named FKs only. Now, it is possible also for FKs named differently from RKs and for RKs being candidate keys. The outcome is that declaring SIR R providing for LNF queries, most often should not require any

procedurality overhead with respect to the creation of SQL base table R through the same statement and not providing for such queries. LNF queries become in this sense a free for DBA bonus to SIR SQL clients.

Then, for likely rare cases, when one cannot declare SIR R as outlined, SIR SQL FK constraint provides now the already mentioned options. The procedurality of an option should typically be fewer than ten characters, substantially less than for the previously proposed solution. Then, SIR SQL Alter Table presented below should be also usually substantially less procedural to declare for SIRs than the previously proposed one. Finally, for the first time as well, we describe the SQL DB upgrading to SIR SQL one.

3. Motivating Examples

We now suppose SQLite3 as the kernel, since it is the one of the proof-of-concept prototype.

Ex. 1. The famous Supplier-Part DB, termed S_P below, was the 1st ever relational DB, proposed by Codd and discussed since in many textbooks, [Codd, E., F. (1970)], [Date, C., J. (2006)], Date, [C., J., (2021)]…. It is the mold for any relational DB since. Let the following sequence of statements be defining S_P for SQLite3:

(1.1) Create Table S (S# Text, SNAME Text, CITY Text, STATUS Int Primary Key (S#));
(1.2) Create Table P (P# TEXT, PNAME TEXT, COLOR TEXT, WEIGHT INT, CITY TEXT Primary Key (P#));
(1.3) Create Table SP (S# TEXT, P# TEXT, QTY INT Primary Key (S#, P#));

Fig. 1 presents S_P that SQLite kernel would create at present, populated with some original sample data. If S_P was real, as the literature abundantly shows, SQL query addressing attributes of SP would usually address also attributes of S or of P. E.g., a client searching for every SP.S# and SP.P# values and QTY associated with would usually need also the associated SNAME and PNAME. If SNAME or PNAME happened to be unknown for some S# or P# values in SP, the unknown outcomes should appear null, as usual. The following query could do, regardless of SQL dialect used:

(Q1) Select S#, SNAME, P#, PNAME, QTY From SP Left Join S On SP.S#=S.S# Left Join P On SP.P#=P.P#;

Q1 has to contain LN from SP to S and from SP to P. The reason is that whatever SP tuple Q1 selects, nothing in S_P scheme indicates the SNAME & PNAME values that Q1 should provide for any selected values of S# and P#. Q1 defines therefore these values itself, whenever they exist or outputs nulls otherwise. The latter option is necessary since Q1 has to work for any SP tuples it could possibly encounter. Since there is no SQL referential integrity constraints declared for SP, Q1 could encounter, e.g., (S6, P1, 20) and (S1, P7, 30). Supposing S and P as at Fig. 1, there is no tuples S.(S6…) and P.(P7…).. Q1 outputs nulls then as it should, through its left outer joins.

More generally, as Codd details for S_P, [Codd, E., F. (1970)], Q1 makes sense, since both SP.S# and SP.P# are LP FKs. Notice that LN through inner joins in Q1 would do only if S# and P# were also SQL FKs. They would be then SQL and LP FKs. Becoming in particular the mold for most of SQL FKs, as the literature shows abundantly. Actually, if SP.S# and SP.P# were SQL FKs, left joins in Q1 would do as well.

For SQLite3 SIR SQL dialect, declaring a DB, say S_P1, through the same statements as for S_P above, (1.1) and (1.2) would reveal the actual ones. Each would create an SQL base table, the same as for S_P. Hence, after the same insert into S_P1.S and S_P1.P as into S_P.S and S_P.P at Fig. 1, the S_P1 tuples in these tables would be the same. Not for SIR SQL SP. The processing would identify SP.S# and SP.P# in (1.3) as two natural SIR SQL FKs we mentioned. Formally, an atomic or composite SA A is a natural FK, iff A (i) shares the proper name and the domain of PK of some R' ≠ R, uniquely named among PKs in the DB, (ii) is not a part of a composite FK of R and (iii) is not subject of SIR SQL FK constraint. Every natural FK is LP only, referred to sometimes as *natural* LP. Finally, the original Codd's LPs we mentioned seem the natural ones. Except that LP and PK shared the domain name. The concept of attribute for the relational model appeared later.

The presence of these LPs in SP would mean for SIR SQL that (1.3) is not the actual Create Table SP, but only shorthand for. Since (1.3) defines BAs and natural FKs only, it would in particular also mean that it defines an empty IE. (1.3) would consequently trigger the rewrite. This one would append to BAs in (1.3) all the "missing" IA names; followed by From clause with the LN. This would end up with the actual IE and the actual Create Table SP.

More in depth, for every declared SIR SQL Create Table R, for every natural FK in R referencing some R', the rewrite appends (i) the list of all the *natural* IAs (NIAs), *sourced* in R' or (inherited) *from* R', called *default* IAs and (ii) the *default* LN clause. Every NIA is named upon the proper name of a non-RK attributes of R'. The rewrite

appends default IAs to the current attribute list within Create Table R being processed, in their source order. For (ii), the default LN clause provides for NIAs values for queries to R, calculable ones only, to recall. Consequently, if Create Table R being processed has From clause already, the rewrite appends to it the left LN join. Otherwise, it appends the entire From clause that is From R_ followed by left LN join. Finally, for processing convenience, every component of the declared non-empty IE and of every actual one, should be within { } brackets, replacing the usual SQL separators. Altogether, (1.3) would become:

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

Here, the entire IE is within a single pair of { } brackets. It is not always so, as we illustrate later. IE in (1.4) is obviously an actual one. IAs appended to (1.3) are all NIAs sourced in S or P. The rewrite basically first appended to the current Create Table SP that is simply (1.3), all the NIAs sourced in S and From SP_ followed by LN join on SP_.S#=S.S#. This became the current Create table SP for the continuation of the rewrhen ite, given the presence of LP P#. The processing adds all the NIAs inherited from P. Since now there is From clause already, the rewrite adds LN join on SP_.P#=P.P# only.

Besides being the actual Create Table SP then, (1.4) is also the actual SP scheme At least for SIR SQL with SQLite3 as the kernel. Likewise, the declared Create Table for S and the one for P are also the actual ones and the actual schemes. For SIR SQL, these two are indeed (only) SQL base tables, since they do not contain any LPs or CAs. As said, S_P1 clients formulate queries towards the actual schemes. So, all the BAs and IAs, if there are any, in (1.1), (1.2) and (1.4) here.

Abstraction made of TWEIGHT column we discuss soon, Fig. 2 illustrates the (actual) S_P1 schemes and content for the clients, after the inserts of Fig. 1. (1.4) is substantially more procedural than (1.3). This does not impact S_P1 DBA with respect to S_P one. As the latter, the former declares (1.3) only.

Observe finally that (1.4) is not only the actual scheme of SP for queries, but is also its actual conceptual one. The rationale is the presence within of all the non-PK attribute names and values in S and in P that conceptually characterize every supply in SP as well. E.g., every supply obviously also has conceptually supplier's name, not only the ID only, i.e., S# in (1.3). Same holds for the part names etc. As we mentioned however, none of these attributes could actually be in S_P.SP. They should be SAs and imply normalization anomalies, prohibited by Codd's model. Unlike they do in S_P1.SP, since they are IAs.

As the result, for S_P1 clients, query Q1 becomes simply:

(Q2) Select S#, SNAME, P#, PNAME, QTY From SP;

Q2 is thus effectively LNF reduction of Q1. Like would be any other query to S_P1.SP, addressing not only some SP attributes, but also some other conceptual ones of a supply stored in S or P only, to avoid normalization anomalies in SP. The crucial practical advantage of Q2 over Q1 is that it saves time to type-in at least more than four dozens of characters, 51 characters at least precisely. This represents almost half of Q1. Last but not least, it also saves the debugging time, at least proportional to the query size and likely larger for outer joins, [Date, C., J., Darwen, H., (1992)]. Altogether, Q2 may save a couple of minutes at least to a usual SQL client. Also, a possibly crucial advantage of Q2 over Q1 and of queries alike is that the former is clearly doable by about any office clerks and casual DB clients. Stating the same for Q1 and any other queries with LN would be clearly a big lie. While Q2 appears indeed quasi-natural within its SQL Select-From-Where. mold. Q1 and any queries with LN in fact, are clearly not so.

Finally, it's instructive to see C-view SP that the actual IE in (1.4) implies. Suppose thus that S_P DBA renames SP to SP_, since SQL views cannot share the proper names of base tables. Then, IE in (1.4) would lead to C-view SP as follows:

(1.5) Create View SP As (Select S#, P#, QTY, SNAME, S.CITY, STATUS, PNAME, COLOR, WEIGHT, P.CITY, From SP_ Left Join S On SP_.S#=S.S# Left Join P On SP_.P#=P.P#);

Observe that (1.5) after Select keyword is the actual IE in (1.4) with same-name and same-place IAs instead of BAs there. It is in fact so for every actual IE and C-view implied. (1.5) is visibly substantially more procedural than IE in (1.4). More importantly, to create it, implies useless effort for S_P DBA, with respect to S_P1.DBA, doing nothing for the same service, i.e., LNF queries to SP. (1.5) could be slightly shorter by the use of SP.*. However this

useful for queries shorthand, is anything but recommended for views. Various reasons for it are only a-click-away on the Web.@

Ex. 2 Suppose that in S_P, there should be a base table, say CG (City, GPS) providing for every city within, its GPS coordinates, e.g., of the City Hall, expressed as an integer. If GPS data are unknown for a city, that one isn't in CG, but the city still can be in S or P as above defined. The sequence starting with Create Table CG below, followed by (1.1) to (1.3) would do, for both SQL and SIR SQL DBSs:

(2.1) Create Table CG (CITY Text, GPS Integer, Primary Key (CITY));

For SQL the result would be simply S_P with CG as additional base table. For SIR SQL the resulting DB, say S_P2, would again differ. The reason is that in both S and P, CITY would become the natural FKs. (1.1) to (1.3) would then be rewritten as follows:

(2.2) Create Table S (S# Text, SNAME Text, CITY Text, STATUS Int {GPS From S_ Left Join CG On S_.CITY=CG.CITY} Primary Key (S#));

(2.3) Create Table P (P# TEXT, PNAME TEXT, COLOR TEXT, WEIGHT INT, CITY TEXT {GPS From P_ Left Join CG On P_.CITY=CG.CITY} Primary Key (S#));

(2.4) Create Table SP (S# TEXT, P# TEXT, QTY INT {SNAME, S.CITY, STATUS, S.GPS, PNAME, COLOR, WEIGHT, P.CITY, P.GPS From SP_ Left Join S On SP_.S#=S.S# Left Join P On SP_.P#=P.P#} Primary Key (S#, P#));

Each IE above is an actual one, with NIAs and default LN appended to. Again, the additional procedurality of these statements would not bother S_P2 DBA. For the clients, in contrast, if Q2 requested in addition S.CITY, S.GPS, P.CITY and P.GPS for QTY < 200, it would remain LNF. Being again as simple and natural for even casual clients, as the SQL Select From Where mold provides for a query:

(Q3) Select S#, SNAME, S.CITY, S.GPS, P#, PNAME, P.CITY, P.GPS, QTY From SP Where QTY<200;

In turn SQL Q1 modified accordingly to, say, Q4, would require in addition LN as before through SP.S# and SP.P#, but also through S.CITY and P.CITY to CG. One way for the SQL client to define Q4 could be as follows:

(Q4) Select S#, SNAME, S.CITY, CG.GPS As S_GPS, P#, PNAME, P.CITY, X.GPS As P_GPS, QTY From SP Left Join S On SP.S# = S.S# Left Join CG On S.CITY = CG.CITY Left Join P On SP.P# = P.P# Left Join CG X on P.CITY = X.CITY Where QTY<200;

Q4 is visibly almost three times more procedural than Q3, i.e., requiring almost 140 more characters. Besides, many if not most of SQL clients are not used to chain LN joins. Hence, Q4 would usually require rather long debugging, say dozens of minutes at least. One may easily test it, e.g., in SQL classroom. All the other ways to express Q4 are at least similarly procedural and debugging time consuming. Unconvinced reader is welcome to try some. E.g., with nested From clause.

Observe finally that if S_P2 was SQL DB and DBA would like to provide for LNF queries, s/he would need C-views S,P and SP. The effort needed would be clearly substantially more important than for S_P DBA declaring (1.5) only. Not to mention no effort whatsoever for SIR SQL S_P2 DBA.@

Ex. 3. Consider the popular variant of S_P, e.g., [Date, C., J. (2006)], p. 225, with the additional base table denoted as SPJ (S#, P#, J#, QTY). Each SPJ tuple models the quantity allotted to some job (project) with J# as PK. This quantity is a part of QTY in SP table, for supply identified in SPJ by (S#, P#). The referential integrity is consequently presumed for SPJ.((S#,P#). Yet another table J with (J#, JNAME, CITY), models all the current jobs. Suppose then that to create J table, DBA follows (1.3) with:

(3.1) Create Table J (J# TEXT, JNAME TEXT Primary Key (J#));

Next, suppose that DBA also creates CG table from Ex. 2. Finally, after the declaration, in order, of J, CG, S, P and SP, DBA declares the usual SPJ table definition:

(3.2) Create Table SPJ (S# TEXT, P# TEXT, J# TEXT, QTY INT Primary Key (S#, P#, J#) Foreign Key (S#,P#) References SP (S#, P#)….);

Here, '…' denotes usual additional, kernel dependent, SQL FK constraint specs. Let us denote SQL DB created so as SP_J1 DB and let SP_J2 be SIR SQL one created by these statements. SP_J1 will have all the S_P2 tables and J, and SPJ tables created by (3.1) and (3.2). Samples of SP_J1 data without CG ones are easily available, e.g., on page 102 of [Date, C., J. (2006)].

For SP_J2 the resulting J table will be as for SP_J1. CG, S, P and SP tables will be as for S_P2. For the declared Create Table SPJ itself, SIR SQL processing will find out the FK constraint. As we discuss in Section 4 below, for it it would be in fact SIR SQL FK constraint that happens to be the SQL one as well. Whenever it is so, the processing rewrites the declared Create Table by appending IAs and LN clause as above defined for natural FKs. Actually, it does so before rewriting the statement as implied by every natural FK. The actual Create Table SPJ, will thus result from appending to (3.2), first all the IAs named upon non-PK attributes of SP, i.e., QTY ONLY; All the IAs sourced similarly in S and in P will follow, including S.GPS and P.GPS. Finally, JNAME will follow and From clause will all LN, i.e., the Left Join clauses, implied. Altogether, the result will be as follows. Again it will not bother DBA by its frightening procedurality:

(3.3) Create Table SPJ (S# TEXT, P# TEXT, J# TEXT, QTY INT {SP.QTY, SNAME, S.CITY, STATUS, S.GPS, PNAME, COLOR, WEIGHT, P.CITY, P.GPS, JNAME, J.CITY, J.GPS From (SPJ_ Left Join SP On SPJ_.S#=SP.S# And SPJ_.P#=SP.P#) Left Join J On SPJ_.J#=J.J#} Primary Key (S#, P#, J#) Foreign Key (S#,P#) Referencing SP (S#, P#)….);

As said it is (3.3), not (3.2) that SPJ clients know for queries. Accordingly, consider query Q5 selecting data that follow for every project with more than 50 parts allocated to. SIR SQL client will express Q5 simply as LNF query:

(Q5) Select S.NAME, S.CITY, S.GPS, SP.QTY, PNAME, P.CITY, P.GPS, QTY, JNAME, J.CITY, J.GPS From SPJ Where QTY > 50;

SQL formulation of Q5, hence with LN would be again substantially more procedural. We leave it as a (dreadful) exercise. Observe again that about any office clerks can issue Q5, while not your result, whatever it is.@

Notice that for all the examples preceding the latter, DBA could alternatively declare the discussed FKs as SQL ones. The same SIR SQL tables would result. The only difference would be the enforcement of the referential integrity.

Ex. 4. Every supply should have for conceptual attribute its total weight, say TWEIGHT, calculated as WEIGHT*QTY. Suppose that TWEIGHT should follow P#. To create S_P1.SP with TWEIGHT, DBA of SIR SQL DB, say of S_P3, should submit:

(4.1) Create Table SP (S# TEXT, P# TEXT, {WEIGHT*QTY As TWEIGHT} QTY INT Primary Key (S#, P#));

The rewrite of (4.1) will add all the IAs pointed to by S# and P# as well as From SP_ followed by LN joins. The result will be the actual Create Table SP, four times more procedural than that in (4.1):

(4.2) Create Table SP (S# TEXT, P# TEXT, {WEIGHT*QTY As TWEIGHT} QTY INT {SNAME, S.CITY, STATUS, PNAME, COLOR, WEIGHT, P.CITY, From SP_ Left Join S On SP.S#=S.S# Left Join P On SP.P#=P.P#} Primary Key (S#, P#));

As said, tor every SIR R, it is the actual Create Table R, hence (4.2) here, that is R scheme for queries. A typical query to S_P3, say Q6: select SNAME, TWEIGHT for every supply with TWEIGHT > 3600, would thus somehow naturally formulate in SIR SQL as:

(Q6) Select SNAME, TWEIGHT From SP Where TWEIGHT > 3600,

In contrast, whatever is popular SQL dialect used, DBA wishing TWEIGHT for SP table could only declare TWEIGHT as SA. TWEIGHT cannot be indeed declared VA in any of these dialects, to our best knowledge. However, to declare it SA would obviously make little sense. In practice, SQL queries to SP1 needing also TWEIGTH would have to include (a) the value expression for TWEIGHT and (2) LN from SP to P. Again a substantially more procedural query, i.e., by at least fifty characters, to type and debug than (Q6) would result. We leave SQL versions of Q6, as instructive exercise.@

4. More on SIR SQL

4.1 Declared Create Table

As the examples illustrated, SIR SQL declared Create Table is intended so that the declared Create Table R for SIR R specifies IE in the least procedural way. In particular, as one could see, IE can be empty in which case DBA declares BAs and table constraint or option only, i.e., actually declares R_ only. This occurs when there are no CAs

and the actual IE. is due (i) to natural FKs as in S_P1 or in S_P2, or (ii) to FKs declared through SIR SQL FK constraint formed as SQL FK constraint, as in (3.2). The rewrite produces the actual IE as the examples showed.

If R has CAs, the rewrite appends From R_ clause followed by every LN join implied by FKs, if there are any such joins. From R_ clause in the actual IE is always necessary for CAs. Recall that, by definition, the actual IE should define any CA as it could be in Create View or any queries to R_. DBA should consequently declare only the remaining parts of every CA actual scheme. In practice, most often it should be the value expressions only, aliasing being assimilated to. E.g., what we did for TWEIGHT in (4.1). Alternatively, it can be a sub-query, as in Ex. 6 below. Finally, if CA scheme involve joins in From clause, the rewrite will append any LN joins to those. It is up to DBA to ensure the SQL correctness of the resulting expression.

Besides, the rewrite preserves every declared SIR SQL constraint formulated as SQL constraint. But, as it will appear soon, one may declare SIR SQL constraints also with some SIR SQL specific options. The rewrite filters then parts of the constraint or filters that one entirely.

The examples have also illustrated that for every declared Create Table defining a SIR, the table defined in From clause of the actual IE is so that for every tuple *t* that could be in R_, there is exactly one super-tuple of *t* in that table. Consequently, for every SIR R, PK of R_ is also PK of R. You might wish to check back this property with the examples.

In our motivating examples furthermore all SIR SQL base tables, especially those providing for LNF queries, except for the one with TWEIGHT CA, were declared as could be SQL base tables with the same base attributes and FK constraints. In particular IE was an empty one and every FK was either a natural one or shared the name of RK that was PK, perhaps composite as in (3.2),. Literature shows that such *primary key named* (PKN) SIR SQL FKs should be the most frequent, every natural FK being PKN by definition. Our examples did not show in contrast SIR SQL FKs declared as SQL ones, but referencing keys named differently from the FKs or being candidate keys, provided the kernel offers such possibility. Such FKs seem also popular, but apparently less than PKN ones. SIR SQL deals with every such FK as with PKN one, i.e., the rewrite simply appends all the NIAs sourced in the referenced table. The actual base table provides accordingly for LNF queries addressing any of these NIAs as well.

The possibility of LNF queries to base tables with above discussed FKs other than PKN ones, is novel with respect to our previous proposals, [Litwin. W. (2025)]. Previously, the declared IE could not be empty for such FKs. DBA had to specify for each such FK, the choice of NIAs available for LNF queries and their placement within the table. One may expect however that DBA will usually, although not always, as we show below, go along with default IAs anyway. Dealing with non-PKN FKs as above discussed, should usually be therefore substantially less procedural. In fact, LNF queries become a free for DBA bonus to the clients also in this case. Altogether, LNF queries are thus a free for DBA bonus to SIR SQL clients anytime the declared Create Table has empty IE and every FK is either natural or declared so that it could be an SQL one.

Literature shows that FKs as just stated should be predominant. Rarely, DBA may however still need FKs implying IAs or LN different from the default ones. DBA may also sometimes need to waive the referential integrity or in contrast, may need FK to enforce it only, as for every SQL FK. As we hinted to, SIR SQL provides for such needs so-called *options* for the declared FKs, specified besides as for SQL. We present these options for the first time. As it will easily appear, the procedurality of the options should usually be a few characters only at most. Equivalent declarations through our previous proposals should usually be substantially more procedural.

First SIR SQL specific option lets DBA to alias R' in FK constraint. It uses the syntax kernel SQL provides for the aliasing in From clause. The option is necessary for every table R with several FKs referencing the same R'≠R or with FK referencing R. The rewrite produces LN using the aliasing and qualifies the appended IAs with the alias whenever needed. Depending on other options the rewrite may or may not keep FK constraint as declared, but stripped from any options. If it does, the constraint will enforce the referential integrity on FK as usual for SQL.

Then, the option termed LPO, means that FK is LP only, i.e., it is not subject to the referential integrity. DBA can mix it with the aliasing, The option makes sense only for FK and RP differently named (why?). The rewrite removes the declared FK constraint, while it appends LN and NIAs as already described. Alternatively, so-called LNO option, also mixable with aliasing, lets DBA to declare the rewrite appending LN only. The result has the LN clause and does not enforce the referential integrity, while DBA declares IAs. Next, the option named LN, means that the

rewrite should produce LN and SQL FK constraint, while IAs specs are again up to DBA. The result will enforce the referential integrity and provide for the actual IE with LN produced by the rewrite and IAs declared. Finally, RIO option will provide for the referential integrity only. It means that for some reasons, e.g., privacy, no LNF query should address the referencing and referenced tables.

Each option other than the aliasing follows 'For' keyword, as illustrated below. For LNO and LN options, DBA specifies IAs individually or through SQL '*' or through SIR SQL specific shorthand '#', (Litwin. 2019). While R'.* stands for all the attributes of R', we recall that R'.# stands only for all the non-RK ones. This lets DBA to declare the same IAs from R', as by default, but placed elsewhere in R. The rewrite expands every '*' and '#' encountered.

Ex. 5. 1. Suppose that (1.3) declares also SIR SQL FK constraints on SP.S# and SP.P#, without any SIR SQL options. The actual Create Table SP would still be (1.4), but with both constraints, enforcing thus the referential integrity of both FKs.

2. Part_Structure DB, [Date, C., J. (2006)], seems the ancestor of DBs with FKs referencing the same table. It is S_P with additional base table PS (MajorP#, MinorP#, Qty). For LNF queries to PS table, supposed in S_P2 for change, the following declared Create Table would do:

Create Table PS (MajorP# Text, MinorP# Text Qty Text Primary Key (MajorP#, MinorP#) Foreign Key (MajorP#) References P MP (P#), Foreign Key (MinorP#) References P (P#);

There are two SIR SQL FKs, hence LPs here. Both reference P. SQL rules require to alias P for at least one of, to prevent name conflicts on IAs. This is done through MP. The actual Create Table will be (again, substantially more procedural):

Create Table PS (MajorP# Text, MinorP# Text Qty Text {MP.Pname, MP.Color, MP.Weight, MP.City, MP.GPS, P.Pname, P.Color, P.Weight, P.City, P.GPS From PS_ Left Join P MP On PS_. MajorP# = MP.P# Left Join P On PS_MinorP# = P#} Primary Key (MajorP#, MinorP#)) Foreign Key (MajorP#) References P (P#), Foreign Key (MinorP#) References P (P#);

3. Suppose finally that DBA wishes to rather place every non-PK IA from MP after MajorP# and every non-PK IA from P after MinorP#. Plausible rationale may be that such table structure is more convenient for Select * From PS Where …queries. For the sake of the example, suppose also that DBA does not wish the referential integrity on P. Perhaps urgent supplies can concern parts not yet in P. DBA can fulfill all the need as follows:

Create Table PS (MajorP# Text {MP.#} MinorP# Text {P.#} Qty Text Primary Key (MajorP#, MinorP#); Foreign Key (MajorP#) References P(P#) For LN, Foreign Key (MinorP#) References P (P#) For LNO;

It is instructive to compare SIR SQL LNF query: Select * From PS Where MP.Pname = 'Screw'; to its SQL equivalent.@

4.2 SIRs Inheriting From SIRs

An IA in Create Table R usually inherits from BAs or IAs in another SIR R'. However, any circular referencing among SIRs is, at least presently, forbidden. Recall that the same holds for any standard SQL views. I.e., if R references (inherits from) R', R' cannot reference R. However, R' may reference R_. Indeed, SAs in R_ do not reference anything, while every VA declared in R_, if there is any, may reference R_ only. Likewise, a sub-query in Create Table R' cannot reference R, but can R_.

Ex. 6. Consider that STATUS in S_P1 is not BA, but CA that for every S.S# value, is either the integer part of the sum of every QTY supplied, divided by 100 or is null, if the supplier presently does not supply anything. Since SP inherits from S, Create Table S cannot reference SP to get QTY values. It can however inherit QTY from SP_. The following declared Create Table S will do then. Here SP declared as in (1.3) is R above and S below is R':

Create Table S As (S# Text, SNAME Text, CITY Text {(Select Int (SUM (QTY) / 100) From SP_
Where S_.S# = S#) As STATUS} Primary Key (S#));

The rewrite will find out that the declared statement has {} brackets, hence IAs, but does not have From clause nor SIR SQL FKs. The actual IE must have From S_ clause. The rewrite will thus append it after STATUS.@

5. SIR SQL Alter Table

For any base table R, SIR SQL Alter Table R includes every capability of SQL kernel for altering BAs of R. In addition, for every R becoming SIR, say R', after the alteration, SIR SQL Alter Table provides also for eventual alterations of IE. With or without altering a BA, DBA may indeed need to add, drop or redefine in any way IAs. Thus one may need to displace some IAs with respect to BAs or to alter value expressions, or to alter the set of BAs being SIR SQL FK, hence to alter NIAs in the actual IE… SIR SQL Alter Table provides for any IE alterations through *IE option*. DBA declares that one as one would declare IE for Create Table, except that IE option includes also BA names of R' as place holders. IE option is mandatory for R' being SIR, unless Alter Table renames the base table name only. This constraint could be relaxed sometimes at the price however of complicating the implementation, outlined in Section 6. We leave it consequently for future work.

Analogously to IE, the *declared* IE option may be also the *actual* one. Together with the relevant BA data types, table constrains or options already in R scheme or declared in Alter Table for the altered BA, the actual IE option defines R' scheme. However, usually, also analogously to IE, the declared IE option should be substantially less procedural shorthand. As IE, declared IE option may for this purpose use in particular '*' and '#' shorthands. Likewise, BA names within may be those of FKs. The rewrite rules for declared IEs expand analogously the declared IE option. .

DBA declares IE option as: IE (…), where (….) is formed as above outlined. One places it always after all BA altering options, if there are any. Unlike for Create Table with IE, there should not be however any { } brackets in Alter Table, within IE option.

Ex. 7. (a) S_P2.SP DBA alters S.CITY to SCITY. Altered S will not have IAs hence will not be a SIR. No need for IE option, SIR SQL Alter Table is simply the kernel SQL one.

Alter S Rename CITY To SCITY; @

(b) S_P3 DBA decides to rename SP.QTY to Q and keep all the other attribute names the same. The altered SP will have IAs hence IE option is a must. It must name all BAs of altered SP. Then, Q renaming will propagate to TWEIGHT. Altogether, Alter Table SP as follows will do:

Alter Table SP Rename QTY to Q IE (S#, P#, WEIGHT*Q As TWEIGHT, Q);

The rewrite will produce six times more procedural actual IE option for further processing outlined in Section 6:

(5.1) Alter Table SP Rename QTY to Q IE SP (S#, P#, WEIGHT*Q As TWEIGHT, Q, SNAME, STATUS, CITY, PNAME, COLOR, WEIGHT, CITY From SP_ Left Join S On SP_.S# = S.S# Left Join P On SP_.P# = P.P#); @

Ex. 8. S_P1 DBA wants to alter SP name to SUPLY. SUPPLY will be a SIR, hence IE option is necessary:

Alter Table SP Rename To SUPPLY IE (S#, P#, QTY);

S# and P# are natural FKs in SUPLY, they will trigger thus the rewrite as follows towards the actual IE:

Alter SP Rename To SUPPLY IE SUPPLY (S#, P#, QTY, SNAME, STATUS, CITY, PNAME, COLOR, WEIGHT, CITY From SUPPLY_ Left Join S On SUPPLY_.S# = S.S# Left Join P On SUPPLY_.P# = P.P#); @

Ex. 9. S_P3 DBA decides, for privacy reasons, that S_P3 clients allowed to query SP should have LNF access to PNAME only. Except for trusted clients, no others should be aware through IAs in SP of any other P attributes. Unlike it should be for S table. The following declared Alter Table would do:

(5.2) Alter Table SP Rename P# To PID, IE (S#, PID, WEIGHT*QTY As TWEIGHT, QTY, PNAME From SP_ Left Join P On SP_.PID = P.P#):

Here PID replaces P# as LP, hence SIR SQL FK. Again IE option is the must. In particular, DBA needs it to declare IAs inherited through PID. S# remains FK in the altered SP. (5.2) will be rewritten to the actual statement:

Alter Table SP Rename P# To PID, IE (S#, PID, WEIGHT*QTY As TWEIGHT, QTY PNAME, SNAME, STATUS, CITY From SP_ Left Join P On PID = P# Left Join S On SP_.S# = S.S#): @

Ex. 10. S_P3.SP should possibly provide the delivery address for every supply. DBA therefore can first create base table D (D#, SAD, ZIP, CITY), where SAD means street address. Then, DBA may alter SP as follows:

(5.3) Alter Table SP Add D# TEXT IE (S#, P#, WEIGHT*QTY As TWEIGHT, QTY, D#);

Here, SIR SQL applies the usual meaning of Alter Table R Add A that is that A is added after every existing BA. IE declares where A should be. If the kernel Add option provides for the placement anywhere among existing BAs, e.g., as in SQL Server, then SIR SQL would provide for the same capability. We leave the actual Alter Table resulting from (5.3) as motivating exercise.@

Ex. 11. S_P3.SP DBA wishes to rename TWEIGHT to TWEIGHT_P, supposing it means weight in pounds and add TWEIGHT_KG defined as below. The following declared statement will do;
Alter Table SP IE (S#, P#, WEIGHT*QTY As TWEIGHT_PND, Int (0.494 * TWEIGHT_P) As TWEIGHT_KG, QTY);
Again, we leave the resulting actual Alter Table as an instructive exercise.@

From all these examples, it is instructive to sum up how significantly less procedural should be in practice SIR SQL Alter Table R with respect to the general procedure at present if DBA used SQL C-View R instead. Recall that C-View R defines logically the same table for queries as SIR R. The general procedure to alter any SQL view R, even the only procedure on most of popular SQL DBs, e.g., on SQLite3, is to issue Drop View R followed by Create View R…. These statements should be embedded within the SQL (atomic) transaction. If C-view alter is due to a BA one, then Alter Table R_ should also be within as well. The implied procedurality is substantial. Recall that one needs to invoke Begin, Rollback and Commit keywords and test SQI Code using some host language. These invocations require dozens of characters at least. Observe finally that the syntax within ( ) of every actual IE option, is also the one of Create View R statement. As the examples illustrate, the overall difference in favor of SIR SQL should then usually be at least in dozens and more likely in a hundred, characters. The advantage of SIR SQL Alter Table should also hold for popular SQL DBSs providing for Alter View or Create or Replace View statements, less procedural than the general procedure.

6. Implementing SIR SQL

6.1. Canonical Architecture

Let us call SIR (SQL) DBS, any relational DBS providing for SIR SQL. To implement a SIR DBS ‘simply’, i.e. through a couple of months of programming, one can stick to the *canonical architecture*, Fig. 3, [Litwin, W. 2019], Litwin, W (2023). In the nutshell, SIR DBS consists then from a kernel SQL DBS, e.g., SQLite3 and from a front-end, called *SIR SQL layer* or *SIR-layer* in short. SIR-layer reuses the kernel DBS through kernel SQL statements. Redoing in contrast entirely a SIR SQL DBS appears obviously anything but a great idea.

More in depth, SIR-layer takes care of every SIR SQL dialect statement and returns the outcome. Recall that every SIR SQL dialect extends to SIRs some kernel SQL dialect, e.g., SQLite3 dialect. The kernel SQL DBS is the actual storage for every SIR SQL DB. SIR-layer declares every such DB to the kernel with the same name through the kernel statement to create a DB.

6.2 Create Table

Fig. 3 illustrates the result of what follows for S_P1.SP. First, SIR-layer processes every declared Create Table R, to find if there are any { } or natural FKs. It uses kernel meta-tables for the latter goal, e.g. the sqlite3 schema table. If it finds neither, it forwards the statement to the kernel as it was declared. The table to create will thus be the kernel’s base table. If, in contrast, it finds natural FKs or { }, it rewrites the statement as already discussed. In particular, if it finds { } and no From clause, it appends From R_ clause. In every case, SIR-layer produces an SQL atomic transaction for the kernel, with Create Table R_ and Create View R as follows. Create Table R_ contains simply every BA and every SQL table constraint and option in the declared Create Table R. It does not contain any SIR SQL specific constraint above discussed. Create View R follows the former and creates C-view R. Accordingly the attribute list consists of every BA name and of every IA in the actual IE. in the original order of all these attributes. From etc. clause in the actual IE follows. To prevent any name conflicts in C-view, for the latter, every attribute there is qualified with its source table name, including R_ for every BA.

6.3 Alter Table

As before, let R be the base table to alter and R’ the altered one. If there is no IE option, SIR-layer simply forwards the statement to the kernel. It means indeed that R was and will remain an SQL table. Otherwise, SIR-layer rewrites IE, whenever needed, as previously described. Then, it issues the query to kernel meta-tables, seeking in the one of all the base tables, whether it contains base table named R or R_. The actual query depends on the kernel. If there is base table R_ in the DB, then SIR-layer issues in order (i) the declared alteration of a BA, with table name

changed to R_, if there is any such alteration, (ii) Drop View R, (iii) Create View R with the content of the actual IA option. Notice that it suffices to replace IE key word with Create View and to add Select keyword after '(', to define the altered C-view R.

. If in contrast, there is base table R, then SIR-Layer issues to the kernel (a) the declared alteration of a BA, if there is any such alteration in the declared Alter Table, (b) Alter Table renaming R to R_, (c) statement (iii) above. In both cases, the issued statements are within the SQL atomic transaction. Finally, if IE option is 'IE ()', then SIR layer issues to the kernel Drop View R followed by Rename R_ to R.

Ex. 12. The kernel is SQLite3. For Ex. 7,a, SIR-layer will simply forward (4.3) to the kernel. For 7.b, it will parse (5.1) to the atomic transaction with inside:

Alter Table SP_ Rename QTY to Q;
Drop View SP;
Create View SP As Select S#, P#, Q, WEIGHT*Q As TWEIGHT, QTY, SNAME, STATUS, CITY, PNAME, COLOR, WEIGHT, CITY From SP_ Left Join S On SP_.S# = S.S# Left Join P On SP_.P# = P.P# ;

All this, including the transaction statements we overviewed, is what SQL DBA who has created C-view SP would need to type-in, instead of (5.1). To say that it is substantially more work than for the SIR SQL DBA seems an understatement.

6.4 Other SIR SQL Statements

For Drop Table R statement, SIR-layer forwards to the kernel the declared Drop Table R iff it finds in the meta-tables that R is a base table name, i.e., that R is not a SIR. Otherwise, i.e., if it finds that R is a view name in the meta-tables instead, i.e., that R is a SIR, it issues Drop View R followed by Drop Table R_, both within an atomic transaction.

For SIR SQL data manipulation statements, i.e., select or update queries submitted to SIR-layer, the latter simply forwards every query to the kernel. For SIR SQL update queries, safe policy for every kernel and every SIR R is to rather address R_. E.g., Insert To SP_..., Update SP_... and Delete From SP_... for S_P1.SP. An update statement addressing accordingly SIR R directly, e.g., Insert To SP…, may or may not work. It depends on kernel view update capabilities. The kernel would indeed address any such queries to view R. In particular, no kernel provides for any CA updates at present.

7. Creating SIR SQL DBs

7.1 Correctness of Actual IE

The examples have illustrated in particular the following general constraint on every actual IE. Namely, every From clause there should (i) reference R_, (ii) define table, say T, where (i) for every tuple t that could be in R_ as stand-alone table, there is exactly one super-tuple of t and (ii) T does not have any other tuples. That is why, in particular, PK of R is also a key of R_, as required for every SIR.

This property is in fact analogous to the similar one of every C-view R. Observe that it characterize every variant of SP we discussed. Besides, if DBA declares R with FKs and CAs requiring some declared From clause, it is also up to DBA to ensure the correctness of the actual IE.

7.2 Creation Order of Base Tables

The basic principle of SIR SQL DB creation is that for every base table R' referenced by some SIR R, R' should exist prior to R. One reason is that SQL kernels we're aware of already respect this principle, at least by flagging an error whenever the table referenced by FK being declared does not exist. Observe also that the creation of S_P1 and of all its variants respected this rule. The other and main reason in fact, is that if R' is not in the DB when DBA creates R, R attribute that would be natural FK referencing R' otherwise, silently will not end up one. The rewriting will not produce the default IAs and LN hence will not provide for LNF queries to R and R'.

If nevertheless the case happens, DBA can still declare R' after the creation of R so that FK referencing the latter is dealt with as it should be. DBA has to declare then R' and follow with Alter Table R IE (). The rewrite will append

the default IAs from R' and the LN join to the existing actual IE. SIR SQL base table R that will result from will be the same as the one that would be created if R' existed before R. E.g., if S_P1.SP was created before S, after creating the latter and regardless of when P is created, DBA should submit simply Alter Table SP IE ();.

7.3 Designing SIR SQL Base Tables

To design best normalized SIR SQL base tables, one can apply any popular method for the SQL base table design. Since IAs cannot create any normalization anomalies, for every SIR R, one should apply however the method to R_ only. I. e., one should consider that SIR R is in iNF ; i=1..6 ; iff R_ is in iNF. The result of such normalization for SIR SQL DB will be all SQL tables and all the bases of SIRs. The rewrite may then add every default IAs and LN. Also or alternatively, DBA may declare some IAs, NIAs or CAs, in particular if there are FK constraints with LNO or LN options, For CAs, there may be multiple tables where they could be. E.g., if S_PJ2 DBA wants TWEIGHT, SP or SPJ may be the choice. More generally, the case happens whenever DBA declares CA, say A, in some base table R1, while base tables R2, R3… inherit A directly or indirectly from R1. DBA should then choose R1, e.g., SP in our case. LNF or CAF queries may then address A in any of these tables, while the inverse may not be true. E.g., for S_PJ2 with TWEIGHT in SP, CAF queries could address SP or SPJ. Otherwise they could address SPJ only.

7.4 Upgrading SQL DB to SIR SQL DB.

Upgrading SQL DB, say D, created using the kernel DBS prior to availability of SIR-layer upon, means to make D accessible to SIR SQL clients as if it was created as SIR SQL DB. The upgraded D, say D' has then for each base table D.T, the base table D'.T that could be created through Create Table T in D as the declared Create Table T for D' or that one could create through such Create Table T except for SIR SQL FK constraints aliased in addition. Those constraints concern SIR SQL FKs discussed in Section 4.1. The base tables of D' are supposed also as if they were created in an order required for creating SIR SQL DB in Section 7.2. The typical result should be the same LNF queries to D' base tables with SIR SQL FKs, impossible for D clients, to recall again, as if D' was created as SIR SQL DB. D' DBA may also later add CAs, unlike can do DBA of D. D itself with its stored content, remain unaffected by the upgrade.

To upgrade D, DBA has to perform specific SIR SQL Alter Table T for every D.T that as D'.T_ would contain some SIR SQL FKs and only for such D.T. These Alter Table statements should execute in the above discussed order in which DBA could create these base tables for D'. If T has only SIR SQL FKs referencing each a different table, none of them being T, then DBA should declare: Alter Table T IE (). The effect would be the rewrite of the existing D.T scheme to D'.T scheme with IE that would result from the declared SIR SQL Create Table T. Otherwise, if for some D.T, FKs, atomic or composite, say F1…Fn ; n > 1 ; reference the same table or T is also the referenced table, then DBA should declare as follows in Alter Table m >= n - 1 aliases A1,…,Am for all the names of the tables referenced by these FKs, considered in their quality of LPs only:

Alter Table IE (LP (F1) References A1,…, LP (Fm) References Am);

The result will be again the rewrite of D.T scheme to D'.T one. Adding again the default IAs, but also altering as needed each FK constraint concerned. At the implementation level, every discussed Alter Table T renames T to T_ in the adequate kernel's meta-table(s) and creates C-view T.

Example 13.1 Suppose that S_P was created using SQLite3, before SIR-layer was added to. After that SIR SQLite3 clients, i.e., those using SIR-layer, can manipulate S_P as SQLite3 clients can do, but only so. E.g., SIR SQL client can execute Q1 above, but not Q2. To provide for the latter and any other LNF query to S_P, SIR SQL DBA needs to upgrade S_P. DBA can do it by issuing simply: Alter SP IE ();. S_P has indeed only one table with SIR SQL FKs that is SP. These (natural) FKs reference furthermore different tables. After the upgrade, the original SP content will remain untouched. Only LNF queries to S_P, e.g., Q2 will become possible and for SIR SQL clients only. The rationale is of course that the upgrade makes SP for SIR SQL clients as at Fig. 2, except for TWEIGHT. To let also for the latter, SIR SQL DBA may follow up with the adequate Alter. With respect to the canonical implementation, the upgrade will rename SP to SP_ in the SQLite3 meta-table and will add to SQLite3 S_P, C-view SP from Fig. 3.

2. Suppose that SQLite3 S_P also had PS table from Ex. 5.2. To upgrade S_P, DBA may rename P referenced by MajorP# to MP, by issuing:
Alter Table SP IE (); Alter Table PS IE (LP (MajorP#) References MP);
The resulting SIR SQLite3 declared and actual PS table schemes, in the form of Create Table, analogously to SQLite3, we recall, will be as in Ex. 5.2. In consequence, PS will support now LNF queries. PS content will remain as before the upgrade.@

Finally, observe that the upgrading procedure may be automated into a single statement, say Upgrade D with arguments for LP aliasing when needed. We leave this goal for future work.

8. SIR SQLite3 Proof-of-concept Prototype

SIR-layer in Python and SQLite3 as the kernel appeared the best for the goal. The prototype provides also for self-running demo. The overall effort was 2-3 months of makeshift Python's developer, i.e., the effort conform to expectations. The demo creates S_P1, either from the actual SP scheme or from the S_P.SP scheme. The latter is assimilated to SIR SQL declared scheme with empty IE, resulting from natural FKs S# and P#. Then, one manipulates S_P1, through LNF queries or, after adding T_WEIGHT CA, through LNF and CAF queries. Users may easily alter the demo. E.g., to prepare their own SIR DBS reusing another kernel: DB2, SQL Server, PostgreSQL…, you name it. See (Litwin, 2025) for more on the prototype.

9. Conclusion

For five decades i.e., since 1974, when IBM introduced SQL, every SQL DBS requires to specify in queries to base tables, LN and CAs, at least other than equivalent VAs could be, for some dialects. The procedurality of LN and of CAs is usually substantial, i.e., at least dozens of characters to type-in and debug. SIR SQL gets rid of this annoyance, usually reducing the queries to the LNF and CAF ones. SIR SQL base tables supporting LNF queries should also usually be declared as could be SQL base tables with the same base attributes, table constraints and options, not supporting such queries, to recall. As said, in this sense, LNF queries are thus for DBA, a free bonus to SIR SQL clients.

For CAF queries, one should declare in SIR SQL Create Table, the value expressions defining the CAs. This should be in general substantially less procedural than for the same CAs declared in queries or views. The latter have to indeed include also some LN. Recall also that SQL base tables at present simply do not let for CAs definable with SIR SQL. Nevertheless, one may feel long overdue even our sample CAs TWEIGHT and STATUS. We expect the future SIR SQL DBs to profit from CAs more and more. While DBA cannot avoid the outlined residual procedurality of CAs, LNF queries to base tables with CAs should still in general remain a free for DBA bonus to the clients. Or, if DBA declares SIR SQL specific options for FK constraints, they should cost DBA almost negligible procedurality overhead for every option other than perhaps LNO, expected relatively rarely used.

Finally, the proof-of-concept prototype SIR DBS with SQLite as the kernel proved simple to realize. Although the problem of LNF and CAF queries to base tables is anything but new, our solution is the only practical, to our best knowledge.

In sum, we conclude that if present SQL DBs had SIR-layers, usually quasi-natural LNF and CAF select-from-where queries to base tables would be the routine, while their present equivalents a bad dream. Recall also that lesser procedurality is the Holy Grail for DB science and entire CS actually. Furthermore, to provide a present popular SQL DBS with SIR-layer, should cost no more than a few man-months effort, proven for SQLite3, i.e., - peanuts by industrial standards. We postulate therefore that DB courses and textbooks take notice of SIR SQL, despite the lack of the full implementation as yet. After all, the same happened to relational DBSs for years.

By the same token, we postulate also to upgrade every popular SQL DBSs to SIR SQL, "better sooner than later". As well as to upgrade existing SQL DBs, what can be easy and fast, as it was shown. 7M+ SQL clients worldwide would benefit from. Bear in mind that this number makes SQL the most used DB language, (Gaffney, & al, 2022). (Sobolevskiy, 2015), (Stonebraker, Pavlo, 2023). Recall also that SQL crowd makes 70% of all developers. Who provide for estimated 31B US$ market for SQL DBSs, (ZipDo, 2024). SIR SQL should finally attract numerous new clients, confortable with typical LNF and CAF queries, while rebutting SQL otherwise.

References

Codd, E., F. (1970). A Relational Model of Data for Large Shared Data Banks. CACM, 13, 6.

Date, C., J. (2006). An Introduction to Database Systems, 8th ed. Pearson Education Inc. ISBN 9788177585568, 2006, 968p.

Date, C., J., (2021). E.F. Codd and Relational Theory. Revised Edition. LULU PRESS, INC., 2021.

Date, C., J., Darwen, H., (1992). Watch out for outer join. In Date and Darwen Relational Database Writings 1989-1991. ADDISON-WESLEY,.

Gaffney, K., P., Prammer, M., Brasfield, L., Hipp, D., R., Kennedy,D., Jignesh, M., P., (2022). SQLite: Past, Present and Future. In PVLDB, 15(12):3535-3547.

Sobolevskiy, M. (2015). How Many SQL Developers Is Out There: A JetBrains Report, Dec. 23.

ZipDo, (2024). Essential Sql Statistics in 2024. https://zipdo.co.

Litwin, W. 2019. SQL for Stored and Inherited Relations. 21st Intl. Conf. on Enterprise Information Systems – Vol. 1: ICEIS, 37-48, 2019 , Heraklion, Crete.

Litwin, W (2023). Prototype Front-End for Stored and Inherited Relations On SQLite3 : Demo for Supplier-Part Database. Aug. 2023, https://www.lamsade.dauphine.fr/~litwin/witold.html.

Litwin, W. (2025). Home Page. https://www.lamsade.dauphine.fr/~litwin/witold.html .

Litwin. W. (2025a). SIR SQL for Logical Navigation and Calculated Attribute Free Queries to Base Tables. In Proc. of 27th International Conference on Enterprise Information Systems, ICEIS 2025, Vol. 1, 191-201, Scitepress (publ.).

Stonebraker, M., Pavlo, A., (2023). What Goes Around Comes Around…And Around. SIGMOD Record, June 2024, 53, 2.

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Fig. 1. S_P database.

Table SP

S#	P#	QTY	*TWEIGHT*	*SNAME*	*STATUS*	*S.CITY*	*PNAME*	*COLOR*	*WEIGHT*	*P.CITY*
S1	P1	300	*3600*	*Smith*	*20*	*London*	*Nut*	*Red*	*12*	*London*
S1	P2	200	*3400*	*Smith*	*20*	*London*	*Bolt*	*Green*	*17*	*Paris*
S1	P3	400	*6800*	*Smith*	*20*	*London*	*Screw*	*Blue*	*17*	*Oslo*
S1	P4	200	*2800*	*Smith*	*20*	*London*	*Screw*	*Red*	*14*	*London*
S1	P5	100	*1200*	*Smith*	*20*	*London*	*Cam*	*Blue*	*12*	*Paris*
S1	P6	100	*1900*	*Smith*	*20*	*London*	*Cog*	*Red*	*19*	*London*
S2	P1	300	*3600*	*Jones*	*10*	*Paris*	*Nut*	*Red*	*12*	*London*
S2	P2	400	*6800*	*Jones*	*10*	*Paris*	*Bolt*	*Green*	*17*	*Paris*
S3	P2	200	*3400*	*Blake*	*30*	*Paris*	*Bolt*	*Green*	*17*	*Paris*
S4	P2	200	*3400*	*Clark*	*20*	*London*	*Bolt*	*Green*	*17*	*Paris*
S4	P4	300	*4200*	*Clark*	*20*	*London*	*Screw*	*Red*	*14*	*London*
S4	P5	400	*4800*	*Clark*	*20*	*London*	*Cam*	*Blue*	*12*	*Paris*

Fig. 2. S_P1.SP table with TWEIGHT. IAs are *Italics*. S and P of S_P1 are as at Fig. 1.

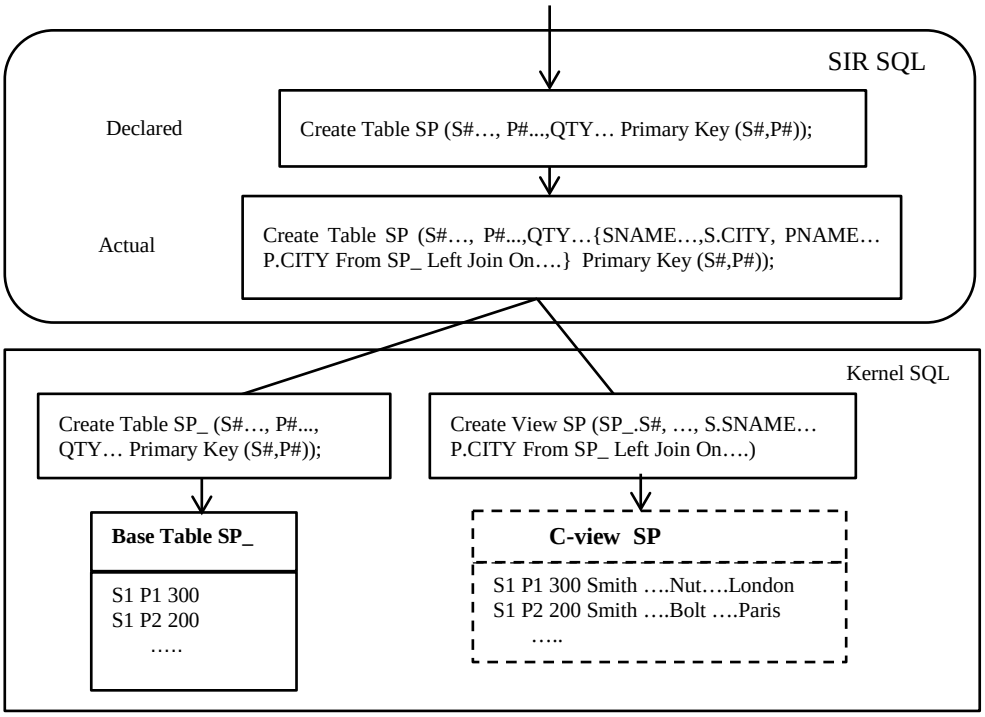

Fig. 3. Canonical architecture for S_P1.SP table. C-view SP content is basically calculable only, as for any SQL views.