Abstract. A stored and inherited relation (SIR) is a stored relation (SR) with additional inherited attributes (IAs), calculated like in a view. SIRs can make common queries less procedural than to the SRs only, without impacting the normal forms. A query may become partly or fully free of logical navigation or of selected value expressions. A dedicated view can provide for the same queries. However, we extend SQL to SIRs so that adding IAs to Create Table can be always less procedural than Create View for any such view. View maintenance can be also more procedural. Finally, our extensions are backward compatible with and generalize the popular tools also avoiding some value expressions to queries through additional attributes in Create Table, being similarly less procedural to declare than Create View for the same purpose. We motivate our proposals through the biblical Supplier-Part DB. We extend the Normal Forms, the Heath’s and Fagin’s theorems to SIRs. Extended Heath’s theorem produces lossless and logical navigation free decomposition. Extended Fagin’s theorem decomposition is lossless as well, but logical navigation over projections remains occasionally necessary. We show how to implement SIRs with negligible operational overhead. We postulate SIRs and our extensions as standard on every SQL DBS and we discuss further research.

1. Introduction

Universally applied Codd’s (relational) model for a Database (Management) System (DBS), [C69] & [C70] has two constructs: a stored relation and a view. Both are named finite relations with atomic attributes only, in 1st Normal Form (1NF) thus. A Stored Relation, (SR), called also a base one, or simply relation or a (relational) table, has stored (base) attributes (columns) only. A view, also called Inherited Relation (IR), has only the inherited attributes. These get only values calculated on-the-fly from SRs or from other views through a stored statement of some data definition language (DDL), usually an SQL Select query. In 1992, we proposed an additional construct, [LKR92]. It was also a 1NF relation, but mixing the stored and the inherited attributes. Examples showed the construct attractive. No one followed however, to the best of our knowledge.

Below, we refine our proposal, especially for SQL DBs. We call our construct Stored and Inherited Relation, (SIR), Figure 1. For every SIR R, we suppose every stored attribute (SA) of R defined as usual. For every tuple of R, the formula we call inheritance expression (IE) defines furthermore every IA value. Accordingly, it completes every stored sub-tuple of R, towards the entire R-tuple. Sometimes, IE may find no value to inherit. In every such case, IE preserves the sub-tuple and completes it with a null instead.

We refer to Codd’s relational model and DB, i.e., with two constructs only, as to Stored Relation or View (relational) model, (SRV-model) and SRV DB. We believe the reader familiar with the SRV-model and SQL in particular. We recall nevertheless that for every SRV DB, its conceptual scheme consists exclusively of some SRs. These are possibly as few as possible without normalization anomalies. Relational views, inheriting from the conceptual scheme or other views present then this scheme differently to different clients. We qualify of SIR-model and of SIR DB the relational model and a DB supporting SIRs. The conceptual scheme of a SIR DB may consist of SRs or SIRs. A view may inherit also from SIRs.

We show that a SIR may have the conceptual scheme more faithful to the reality than SR it expands. The rationale is that IAs may model conceptual properties inconvenient as SAs. The latter could adversely impact the normal form of the SR or lead to impractically frequent updates. By addressing SAs and IAs in the same SIR, an SQL query may totally or partly avoid the logical navigation, necessary within every equivalent query to the SRs only. We recall that such navigation in an SQL query occurs when the query has to refer to several relations with, usually, equijoins among those. Likewise, a query to a SIR may avoid selected value expressions. Altogether, SQL queries should end up usually less procedural (simpler, more usable…) than their equivalents to SRs only, by the basic measure of the number of characters to type-in.

---

1 Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, 75016 Paris, France. witold.litwin@dauphine.fr
2 LAMSADE Research E-Report. Latest update May 2, 2018
On the other hand, it is easy to see that for every SIR R, there is always at least one view R in SRV-model, defining mathematically the same SQL relation, i.e., where the attribute values order matters, not only the value set. View R may also have, for every attribute, the same proper name as in SIR. View R is equivalent in this sense to SIR R, differing only in implementation. For every stored value in SIR R, view R provides indeed the same value, but calculated through inheritance from some SR. View R provides for the same simpler queries as SIR R. Actually, since decades, such views are notorious “escape route” for clients unhappy with the procedularity of most queries to the normalized SRs only. Universal views, providing all the attributes and, possibly, all the values of the DB in one relation, were particularly studied, [MUV84].

We propose clauses for Create Table and for Alter Table declaring or altering an IE. We show that our clauses are such for every SIR R that our declaration of the IE within Create Table R can be less procedural than any equivalent Create View R. Every SA in our Create Table R remains also declared as usual, we recall. SIR R expanding some SR R_B may thus provide for simpler queries to R_B at lower procedural data definition cost than that of any equivalent view R. View R may be more procedural to alter than the IE as well. We show finally how to implement SIRs on a popular DBS with negligible storage and processing overhead. The need for faithful conceptual modelling and non-procedural queries is universal. We therefore postulate SIRs as new standard capability of every relational DBS.

We do it especially since some popular DBSs provide in fact already for limited SIRs for decades. Unknowingly of course, as it will appear. These are SRs possibly carrying also so-called virtual attributes (VAs) or computed, generated… columns. We recall that one defines every VA as a named value expression in Create Table through a dedicated clause. Queries avoid the expression through simple reference to VA name. The advantage of the whole capability is that whatever is the number of VAs in Create Table, their declarations are altogether always less procedural than any Create View otherwise needed. The advantage extends to all the other SQL DDL statements concerned by VAs. Our clauses for SQL aim precisely at the same goal. But we generalized it to every SIR. The IE can consequently support value expressions not possible as VAs at present, e.g., with aggregate functions. For an IA that could be a VA, the IE may also avoid the logical navigation, or reduce it at least. Our clauses are in particular designed for backward compatibility with current ones for VAs, abstraction made of minor syntactical differences between current SQL dialects.

Next section discusses the basic concepts of SIR-model. It also introduces our clauses for an IE in Create Table. We then motivate our proposal by applying SIR-model to the “holy” for SRV-model Supplier-Parts DB. We finally complete the introduction of our SQL extensions; discuss the utility of SIR-model and the related work.

Section 3 extends the basic relational design rules to SIRs. We generalize the NFs other than 1NF. We restate Heath’s and Fagin’s theorems, [H71], [F77]. The restated theorems decompose SRs or SIRs. They create lossless or presumed to be so decompositions into two projections at each step, as do their originals. For an SR, our both projections have the original SAs, but at least one is a SIR. For the restated Heath’s theorem, the latter projection has all the attributes of the former as IAs. A query to SAs and IAs there is free of the logical navigation over the original projections.

The property extends to the decomposition of SIRs as well. The end result is that, if the conceptual scheme of a SIR DB results from successive restated Heath’s decompositions only, it is logical navigation free. One projection must be the universal SIR, with all the attributes in the DB and all the data values. As for a universal view over the projections at present, this freedom, however, lasts only until a dangling tuple appears in a projection. The decomposition ceases to be lossless then. A query selecting a dangling tuple and a tuple elsewhere may require the logical navigation. Even so, SIRs may reduce the latter with respect to the original one, as we will show.

In contrast, for the restated Fagin’s decomposition, the need of occasional navigation over the projections always remains. Altogether, it will appear that SIR DB produced by the restated theorems
should always have the same SRs as these produced by the original theorems. But, some SRs would become parts of SIRs. The new design should typically bring then the discussed advantages. A motivating example in the section backs up our claims.

Section 4 discusses the implementation of SIR-model over an existing DBS. We specify an algorithm mapping SIRs into SRs and views there. We show how to process queries to SIRs. We also show that the storage and processing overhead of a SIR implemented as proposed is negligible. Section 5 concludes that SIRs improve the usability of the relational DBs and overviews the future work.

2. **SIR-Model**

2.1 **Overview**

As Figure 1 illustrates, every SIR is a 1NF relation, i.e., a finite subset of a Cartesian product of atomic attributes (columns) over some domains, subject to every algebraic or predicative operation and aggregate or scalar function applying to 1NF relations. As usual for SRV-model, we also basically consider an IA value immaterial. Likewise, every SIR has a name and scheme defining all its SAs and IAs. Every SA scheme is as usual as well, e.g., as defined by an SQL dialect. The already mentioned IE defines all the IAs of a SIR. Every IA inherits from SRs, views or SIRs. The IE selects IAs and calculates tuples of IA values through some relational or value expression like in a view. The expression must be so that every IA is functionally dependent on the key of the SR constituted from the SAs of the SIR.

Figure 2 illustrates the resulting possible structure of a SIR as an SQL relation, i.e., where the attribute order matters. Each grey rectangle represents a stored sub-tuple. For every SIR R, all these sub-tuples form a stored sub-relation we qualify of base of R. The base has its proper default name R_B. The green and white labelled Null rectangles represent the sub-tuples forming altogether the SQL view constituted from and only IAs. We recall that unlike a relational algebra view, an SQL one may contain duplicates and null tuples. We name this view R_V. For every stored (base) tuple, R_V defines one and only one inherited tuple. Together with R_B-tuple this sub-tuple forms R-tuple. Each green rectangle represents a valued R_V sub-tuple. Some of these sub-tuples can be duplicates. Alternatively, an R_V sub-tuple may have no inherited values, i.e., can be a null sub-tuple. Each line of Null rectangles at the figure symbolizes a null sub-tuple.
Merging these two definitions least procedurally into a single statement is in fact our goal in what follows.

On the other hand, we call every view R with property (i) *conceptually equal* to SIR R, CE-view R in short. Observe that for every CE-view R, for every IA sourced in R_B, not only the proper names of the source and the inherited attribute are the same, but the full source name of the IA has R_B as qualifying prefix. SIR R has the same property, what motivates actually our name choice for such a view. This property is not characterizing every view R, as it will appear. More precisely for every SIR R there is CE-view R. Under some restrictive conditions on the DB, a SIR R may nevertheless also have view R, where the IAs inherited from R_B by CE-view R, are inherited from other relations. We will discuss more these views later.

We define every SIR DB scheme as usual through some DDL (Data Definition Language). As we hinted to already, operationally, we consider an SQL-like DDL. We suppose the statements of this DDL extending some existing SQL dialect with clauses for IEs. We call that dialect the *kernel* for the resulting SIR-model dialect. We presume the kernel to be the dialect of some popular DBS.

We respect to the statement creating a SIR through the above discussed merge of the definitions of R_B and of R_V for SIR R, one can choose to extend Create Table R of the kernel with clauses declaring the IAs, i.e., R_V. Vice versa, one could extend Create View R with clauses materializing the SAs of the SIR, i.e., declaring R_B. Finally, one could keep the semantics of both statements as is, while creating a wholly new statement, e.g., Create SIR R. In every case, SAs and IAs could intermix in the result, as at the figure or R_V could precede R_B. Using Create Table is nevertheless the only choice potentially backward compatible with the current SQL declarations of the already mentioned VAs. Those are indeed declared through Create Table with dedicated clauses. We feel this reason compelling for the same choice for SIRs. Actually, even without R_B materialization clauses, the semantics of Create View naturally extends then to SIRs as well. A source relation may indeed become a SIR, as already mentioned.

In a word, Create Table may thus create an SR or a SIR. Every SIR scheme may furthermore get altered through an Alter Table statement, we suppose extended to SIRs as well. We discuss those extensions soon, as well as those to the other SQL DDL statements.

We recall that, unlike the relational algebra, SQL orders the attributes of every relation. For an SR, the attribute order is that of the usual reading of Create Table, i.e. top-down and left-to-right. An Alter Table can reshuffle this order. Likewise, every SQL view has the attributes following the left-to-right order in its Create View statement. We therefore presume SAs and IAs in every SIR R, to follow the attribute order in Create Table R or the altered one accordingly. In both cases, for every SQL relation R, Select R.* query should display the attributes in order as well. We say “should”, as some popular DBSs apparently do not guarantee it, e.g., SQL Server. We basically presume nevertheless that rule applying to the SIRs as well.

For every SIR R, it should already be clear that it is the IE that defines R_V and, for every tuple t of R_B, it determines the single tuple t’ of R_V expanding t, either with some attribute value(s) or with a null R_V-tuple, as at Figure 2. Observe that IE generalizes in this way a Select clause subquery in a view or query to possibly several attributes. The latter, we recall, is always limited to a single attribute with, for every tuple outside the subquery, at most one additional value.

Next, we consider that notorious SQL naming rules apply to SAs and IAs in SIRs. The additional rule is that for every SIR R, every SA A of R can be qualified not only as R.A, but also as R_B.A. For CE-view R instead of SIR R, these choices mean for an SQL statement either the direct selection of A from its original SR, i.e., select R_B.A, or the selection of A inherited by view R from R_B, i.e., select R.A. For both SIR R and CE-view R, in some situations qualification with R_B may be the only choice. The reason is avoidance of a *circular reference* between CE-views and between SIRs consequently. Such a reference between views occurs when view R1 attempts to inherit from view R2, while view R2 already inherits from view R1. The cycle can be transitive. Every popular DBS we are aware of forbids
the circular referencing. We do not permit for circular referencing between SIRs neither. Every R_B inherits from nothing. Hence, no referencing to it can reveal circular.

Figure 2. SIR structure as an SQL relation

An IA in SIR R may in particular be sourced in other IAs in R, e.g., for an IA defined through a value expression (VE). We prescribe any circular referencing among such IAs as well. Again, today, for VEs in views, any circular referencing among such IAs is similarly prohibited.

SIRs being 1NF relations no SQL DML (Data Manipulation Language) statement needs new SIR specific clauses. We suppose simply that the kernel SQL Select, Update, Insert or Delete statements simply apply to SIRs as well. For the update queries, i.e., the latter three statements, in particular, we suppose that for every SIR R, an update query to R is valid only if it does not attempt to modify an IA defined by a value expression or a Select clause subquery, directly or transitively. In other words, the query may only modify an SA or an IA that R inherits as equal to some SA, directly or transitively. The expected result is as if the modification concerned the SA directly. The rationale is that at present, no popular DBS allows to modify an IA defined by a value expression or a subquery, even if it is theoretically possible, [LV86].

An insert into a SIR R, creates therefore the tuple(s) in R with every SA instantiated as in the query. The primary key of R_B must be among these. The insert may instantiate an IA. The result is the instantiation of the source SA. That one is then inherited back by the IA, and by every IA for which SA is the source. If an insert does not instantiate an IA, then this one gets the value defined by the IE, or becomes null as it was discussed. Likewise, an update to existing tuples proceeds as usual for any SA and an update to an IA, provided valid, propagates to the source SA. Next, a delete removes from R the tuples specified in the Where clause of the Delete statement, regardless whether this one concerns SAs or IAs. Finally, as usual, any of these operations gets blocked if it violates the referential integrity or usual check constraints etc.

An IE can be, so-called, explicit IE. We denote it as E and may index it with the name of the SIR, e.g., E_R. An IE may have also a shorter representation that we call implicit IE, denoted I or I_R. We first introduce the explicit IEs.

2.2 Explicit IE

In the nutshell, an explicit IE $E^*$ for SIR R ($A_1,\ldots,A_n$) is (i) the part of Create View statement defining CE-view R after the prefix: ‘Create View R As (Select’, with (ii) Select clause attribute list $A_1,\ldots,A_n$ reduced to the attributes of R_V only, e.g., symbolized with green columns in Figure 2. In other words, these attributes are all and only not SAs in SIR R, i.e., in view R, they are all and only not inherited with their proper names and values from some SR R_B. The latter in SRV-model is not a part of SIR R of course, but a distinct SR, equal to that part. Since CE-view must inherit from R_B, in pseudo-SQL, the generic form of Create View for CE-view R could be:

(1) Create View R As (Select $A_1,\ldots,A_n$ From R_B[...]);
Then, Create View $R_V$ As (Select $E_R$), where $E_R$ denotes its actual expression, (extension), would create view $R_V$ for SIR $R$. Figure 2 shows, attributes of $E_R$ may intermix in SIR $R$ with the SA's. Let us denote $E_R$ every attribute $A$ that is inherited from $R_B$. Let us also denote as $I$ every $A$ that is an IA in SIR $R$, i.e., is also an attribute of $R_V$, as defined by $E_R$. Then, in pseudo-SQL, Create Table $R$ defining SIR $R$ through some $E_R$, e.g. defining SIR $R$ at the figure, would have the form:

Create Table $R$ $B_1$...$B_j$, $I_j$...$I_i$, $B_{j+1}...B_{i+1}$...$B_{i+2}$...$B_{i+3}$... $I_{i+3}$...$I_n$ From $R_B$ [...]

The Select clause attribute list for $E_R$ would be constituted from every $I$ only. The scheme of every $B$ would be declared as usual in Create Table $R_B$. Thus, every other usual clause, e.g., related to the primary or foreign keys there could be in above Create Table $R$ as well.

We will show several seemingly practical examples of $E''$s. In general, (1) has to enforce all the conditions we have discussed for every SIR $R$, CE-view $R$ and every $R_V$, obviously. One, likely frequent, possibility for From clause of $E''$ is then as follows. Suppose view $V$ conform to the following requirements: (a) some SR named $R_B$ contains an attribute with default name $A$, $A$ being either a simple attribute or a composite one, i.e., $A = A_1...A_m$, (b) some relation $X$ is source of some IA in $V$ and has key attribute $K$, equally perhaps composite as well, $K = K_1...K_m$ and (c) $R_B$.A and $X$.K share the domain. Suppose then that (i) for simple $A$, From clause is: From $R_B$ Left Join $X$ On $X$.K = $R$.A and (ii) for composite $A$, the clause is: From $R_B$ Left Join $X$ On ($X$.K1 = $R_B$.A1 And...And $X$.Km = $R_B$.Am). It follows that $V$ respects all the requirement for $R_V$ given $R_B$, i.e., we may consider $R_B$ as base of some SIR $R$, $V$ as $R_V$ of $R$ and the part of the scheme of $V$ after Select keyword as $E_R$. Likewise view $R$ constructed from $R_B$ and from $V$ through Create View $R$ defined by (1) above, is then CE-view $R$.

Observe also that in SIR $R$, each join above contributes to define $R$, while it refers to a part of $R$. We call it therefore for a SIR recursive join. Actually, a recursive join may also be a $\theta$‐join. One can expect recursive joins to be the most frequent tool for IEs. Motivating examples will justify all these statements.

### 2.3 Implicit IEs

It is sometimes possible in Create Table $R$ to declare an implicit IE $I_R$ instead of $E_R$. We denote then the replaced $E_R$ as $E''_R$, $I_R$ is less procedural than $E_R$ and any equivalent $E_R$ (defining the same SIR $R$). $I_R$ may also provide the backward compatibility of an IA resulting from specific VE with the industrial notation for the aforementioned virtual attributes (VAs), [LV86]. With slight syntactical differences, some, but not all, popular DBSs provide for VAs, also named computed or generated columns, we recall. E.g., MySql does it, with parentheses around every VE. SQL Server provides for VAs as well, but without parentheses around VEs. In contrast, MsAccess does not provides for VAs at all. If for some $E_R$, every IA could be a VA, then every such $E_R$ would be more procedural than the current notation of these VAs. In contrast, for every such $E_R$, there is a backward compatible $I_R$ as we define it below. The compatibility means here that it could be made identical to the present notation, with or without parentheses, at wish.

The reduced procedurality of an $I_R$ usually results from implicit From clause, together with every optional usual follow up clause. The specific generic character we denote as ‘$H$’ within or even instead of $A_1$,...,$A_n$. An list of $E_R$ may also reduce the procedurality, whether From clause is present or not. Likewise, an IA $A$ defined through a VE with a subquery or with Group By clause in $E_R$ may have a shorten definition in $I_R$. That one may be as simple as: ‘$VE$ As $A$’ or ‘$A$ As $VE$’. All these capabilities apply only iff $R_B$ and all the other source relations of $E_R$ fulfill specific conditions. The DBS rewrites then $I_R$ to specific $E_R$ that we denote as $E''_R$. It does it as the pre-processing of Create Table $R$ or Alter Table $R$ processing we specify later for every SIR $R$, supposed defined through some $E_R$ for this processing. The rewrite rules appear open-ended. We define now only some for $I_R$ capabilities we just announced. They appear sufficient for most applications. We leave the study of eventual additional rules for the future. If some rules do not apply to $R$, then the IE must be an $E_R$. 

- 6 -
Rule 1. Every IA in R inherits only \textit{locally} that is from R_B only, through a VE without an aggregate function or a subquery. Then, the implicit From clause in I_R, i.e., the one of E'_R, is simply: From R. No other clauses follow that one in E'_R. The list A1...An is the same for I_R and E'_R, unless modified through Rule 5 below. The list contains thus, in order, all the IAs and SAs defined by Created Table R and every Alter Table R prior to E'_R calculation.

Rule 2. Only some IAs inherit according to Rule 1. Every other IA A inherits from IAs in source relations other than R_B, aliased R_B included, say X1,...,Xm. Suppose that the latter numbering follows the order of IAs in R. Every A either inherits as is from some source attribute S or A results from VE as in rule 1, but involving one or more S and, perhaps, some SAs in R_B. Also, for every I = 1...m, let Xi.Ki be the primary key of Xi, perhaps composed as: Xi_1, Xi_2.... Suppose furthermore that for every Xi.Ki, there is some R_B.Ki as well. The latter attributes are not necessarily disjoint. Then, the implicit From clause is:

\[
\text{(2) From R_B Left Join X1 On R_B.K1_1 = X1.K1_1 And R_B.K1_2 = X1.K1_2... And Left Join X2 On R_B.K2_1 = X2.K2_1 And... Left Join Xm On R_B.Km_1 = Xm.Km_1...;}
\]

The list A1...An in I_R is as for Rule 1 or shorted through Rule 6 below.

3. Only some but not all IAs of R inherit accordingly to rule 2 or even none. Suppose the former case for the time being. Let A'1, A'2... denote these IAs in their order in Create Table. Let then X'1, X'2... denotes the respective source relations. Suppose every other IA, say, Ai1, Ai2..., numbered again in the order of these attributes in Create Table ; i1,i2... \in (1,2...n) ;. Next, suppose every Ai inheriting respectively through some VE V1, V2,..., where each V contains an aggregate function not within a subquery, if VE is followed by Group By VE. Furthermore, suppose every AiK ; k = 1...n ; inheriting from some Xi.K_B, where every Xi.K_B has key Kk with the property common to every X in Rule 2. To simplify, suppose that every K1, K2... is atomic, as it should be usually. Suppose finally that arbitrarily for the time being only every A precedes every A' in Create Table. Then, the implicit From clause, as well as the explicit From clause for E'_R thus, results from the following pseudo SQL expression:

\[
\text{(3) From (SELECT R_B.*, (Select V1 From Xi1_B Where Xi1_B.K1 = K1) As Ai1 From Xi1_B, (Select V2 From Xi2_B Where Xi2_B.K2 = K2) AS Ai2 From Xi2_B...), A'1, A'2... FROM R_B Left Join X'1 On R_B.K1_1 = X'1.K1_1 And... ;}
\]

The final 'And...' continues for X' as it continued for X in Rule 2. The whole Left join clause reduces to From R_B only if none of the IAs conforms to Rule 2. Expression (3) complicates in the way easy to see for composed K1 or K2... Also, if IAs A and A' intermix, then (3) changes adequately. Finally, the A1,...,An list defining the I_R is as for Rule 2.

4. Everything stated in rule 3 holds, except that some Aj1, Aj2... among A1, A2... of I_R inherit each from some Ai1, Ai2... resulting from Rule 3, through a subquery Sj1, Sj2..., each perhaps with an aggregate function and each defining respectively Aj1, Aj2.... Also, let S' be S where name R_1 replaces R in every reference to R. Again, suppose arbitrarily every Aj following every other A in Create Table. Then, the implicit From clause results from the pseudo SQL expression:

\[
\text{(4) FROM (Select R_1.*, S'j1,S'j2... From (Select R_B.*, R_B.A'1, R_B.A'2... (Select V1 From Xi1_B Where Xi1_B.K1 = K1) As Ai1 From Xi1_B, (Select V2 From Xi2_B Where Xi2_B.K2 = K2) AS Ai2 From Xi2_B...), A'1, A'2... FROM R_B Left Join X'1 On R_B.K1_1 = X'1.K1_1 And...) As R_1;}
\]

Here, the 1^{st} subquery defines the relation with all the attributes of the relation produced by its own nested subquery, and with every Aj1, Aj2... The nested subquery defines the relation termed R_1 with all the attributes resulting from Rule 3. Also, as for Rule (3), if some Aj intermix with other IAs differently from the above, expression (4) changes adequately. Finally, A1,...,An list of I_R is again as for Rule 2.
5. Every IA A of R inherited through a VE V, has to be declared in $E_R$ as: V As A. An $I_R$ can alternatively declare every such A as: A As V. When $E_R$ is computed then according to every other relevant rule, the latter notation is reversed to the former one.

6. Suppose that for some relation X as in Rule 2 above, $E_R$ selects nominally each attribute of X except for X.K. Then, for every such X, $I_R$ may replace this enumeration with the single element X.#. Also, in every A1…An list in $I_R$, # alone in this list stands for X1.#, X2.#,…Xm.#, each as in Rule 2 and as follows. First, that list can be all such relations within explicit From clause of $I_R$. Otherwise, it results from all the SAs in R_B. For every A there, A is Ki as for Rule 2, iff there is exactly one relation X in the DB with key X.Ki. Every such X enters then From clause in $E_R$. If for some A, there are more relations with A as key, then # alone is inapplicable as element of $I_R$.

7. An IA does not fit the assumptions of rule 1 or of rule 2, or ...of rule 4. Then, there is no $I_R$ for R.

Rule 1 is quite obvious for SQL fans. Rule 2 basically avoids the logical navigation to $I_R$, hence to the DBA. $E_R$ would contain the From clause with the recursive left equijoins as in (2) above. A DBA manually defining equivalent $E_R$ could have different preference, e.g. towards right joins only or even towards some or all inner equijoins, under some restrictive conditions. A dialect may require furthermore parentheses in (2), imbricated so to impose the left to the right order on joins, e.g., MsAccess. For only a single implicit inner equijoin and mono-attribute primary key, MsAccess has in fact a completion rule for QBE queries similar to Rule 2. MsAccess calls these joins automatic. SRV-model does not consider implicit joins, although there were proposals for adding these, [LA86], [LSW91].

Rules 3 and 4 avoid the pain of subqueries nested in A1…An or in From clause of $E_R$. Notoriously, a DBA having to specify these for $E_R$ or an equivalent $E_R$ or CE-view R, could find them stressful procedurally. We guess the reader to share the feeling, till the examples in Section 2.4 right below. To understand the rationale for expression (3), observe that every subquery realizes, for each Ai, the calculation that could alternatively result from nested From clause with Group By Xi1_B.A… without Having or Order By in $E_R$. In Rule 4, the level of nesting is two. We have not seen any really practical needs for more levels. Observe finally that both From clauses preserve every tuple of R_B as they should.

Rule 5 allows for, as we call it, secondary form for the declaration in Create Table R or Alter Table R of any IAs under consideration. Unlike the basic form in every $E_R$ rule 5 refers to, the latter is also the one of a VA, abstraction made of dialect-dependent syntactical details, e.g., of the parentheses we spoke about. The convenience of Rule 5 is thus that Create Table R for SIR R with IAs that could be VAs only at some DBS, is then backward compatible with Create Table R there, e.g., at MySQL. In particular thus, as we aforementioned, every table R with VAs at every DBS with that capability is in fact a specific SIR R, limited to the local inheritance only, without being recognized so. Create Table R for such SIR with $I_R$ declared through Rule 1 with Rule 5, but also the one with $I_R$ declared through Rule 1 without Rule 5, are each equally procedural as Create Table R with the VAs instead at present. All are also less procedural than CE-view R would be. The rationale for proposing tables with VAs was thus already visibly the same with respect to the latter feature as ours for SIRs more generally.

Next, Rule 6 provides for every qualifying SIR R, for an IE visibly less procedural than every (equivalent) $E_R$. The need for such SIRs appears very common one in fact, as we will show. Also, there may be view R where for some SA in SIR R, say with proper name A, view R inherits the IA A from another relation than SR R_B, unlike for every CE-view R. The IA has then a different full source name than R_B.A. The view is no more conceptually equal to SIR R. Nevertheless, under some restriction on the DB, it can still provide for the same simpler query. We qualify it of query equal view R, QE-view R. QE-view R may have Create View R substantially simpler than CE-view R. It may even be simpler than every $E_R$. For every QE-view R, Rule 6 allows then for $I_R$ (again) less procedural than Create View R. We recall we stated the existence of such IE for every Create View R of CE-view R or of QE-view R. A motivating example will illustrate all this discussion soon.
Finally, Rule 7 defines when $I_R$ does not exist, given all the other rules stated for the implicit From. We said nevertheless that our rules were open-ended. Hence, future work may restrict this rule.

2.4 Motivating Example

We reuse the biblical Codds’s Supplier-Part relational DB. Its variants motivated the original proposals, [C69], [C70]. These settled the relational (conceptual schema) design rules of SRV-model, based on NFs as known. Through those rules, Supplier-Part DB molded about any practical DB created since. The variant we picked up is probably the most known, [D4]. It is often named S-P in short. We refer to it as S-P1. We restate S-P1 into variants with SIRs. We call these S-P2, SP3...

Figure 3 shows the conceptual scheme of our basic SIR-model DB we call S-P2. Like S-P1, S-P2 consists of three relations. It will appear that S-P2 is the optimal DB for the relational design under SIR-model, for the enterprise of S-P1. S-P2 scheme was in fact also our motivating example in [LKR92], with different syntax for IEs.

The figure illustrates also graphically how the original S-P1 conceptual scheme relates to S-P2 conceptual scheme with SIR SP defined in the upper part of the figure and how both relate to S-P1 scheme with CE-view SP as an external scheme. The colors symbolize SAs and IAs as in Figure 2. We discuss all these schemes in detail below. One may observe nevertheless already that view SP appears redundant in part with respect to the SR SP. It is indeed, as, by definition, it redefines all the (stored) attributes of SP_B as IAs. The redefinition must always cost some procedurality, necessarily adversely affecting CE-view SP compared to SIR SP. The motivating example will now illustrate this point in detail, as well as other properties of SIRs we hinted to.

Example 1. As hinted above, S-P1 models an enterprise with some suppliers, parts and supplies. A supply contains some quantity of a part shipped by some supplier. A supplier may supply nothing for the time being. Likewise, a part may be not supplied. S-P1 conceptual schema of the enterprise consists of three stored relations: S for suppliers, P for parts and SP for supplies. The definitions of SAs are self-explaining. We underline the primary key attributes, as usual. As known, this scheme is optimal one under the SRV-model relational design rules, i.e., contains the fewest relations free of normalization anomalies.

<table>
<thead>
<tr>
<th>S-P2 Scheme</th>
<th>Table S</th>
<th>Table P</th>
<th>Table SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>S# Char,</td>
<td>P# Char,</td>
<td>S# Char,</td>
<td></td>
</tr>
<tr>
<td>SNAME Char,</td>
<td>PNAME Char,</td>
<td>P# Char,</td>
<td></td>
</tr>
<tr>
<td>STATUS Char,</td>
<td>COLOR Char,</td>
<td>QTY Int</td>
<td></td>
</tr>
<tr>
<td>CITY Char;</td>
<td>WEIGHT Char,</td>
<td>S.#, /* Every, but S#, attribute of S.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CITY Char;</td>
<td>PNAME, COLOR, WEIGHT, P.CITY AS PCITY; /* IAs from P.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3 S-P1 DB and S-P2 DB schemes.

Figure 4 shows example extensions of S-P2 relations, i.e., all the tuples. S-P2.S, and S-P2.P have the original S-P1 schemes and tuples, hence have SAs only. S-P2.SP is, in contrast, a SIR. It keeps the original SAs with their schemes i.e., ($\#$, $\#$, Qty) and tuples. These SAs and tuples form the base SP_B. The SP_B key is also the original one, i.e., ($\#$, $\#$). It will appear soon that it is also the key for
the entire S-P2.SP. Observe that this is a general property of every SIR R, i.e., every key of R_B is also the key of SIR R. As required for every SIR as well, S-P2.SP scheme defines also the IE. That one is implicit, since without From clause. According to our naming conventions, we call it I_sp. The generic S.# element of I_sp stands for all but S# attribute of S with their default names. These are SNAME, STATUS, S.CITY as in Figure 4. We applied it to S, since S, as P besides, fulfills conditions for Rule 6. All the other IAs of I_sp at the figures are from P. We could not apply ‘#’ to these. P.CITY has indeed the alias PCITY in SP, since clients often do not adore prefixed names. The alias is unique in SP, as it should, and becomes the default name for P.CITY there. Prefixed names indeed often annoy the clients. The ‘;’ terminates I_sp and the whole scheme of SIR SP, as usual for SQL.

The SQL order of S-P2.SP attributes, for Select SP.* in particular, is the usual one, i.e., the top-down and left-to-right, after the expansion of S.# and of P.#. It is thus: (S#, P#, QTY, SNAME...S.CITY, PNAME...PCITY). Every IA name and value is italics at the figures. Select * From SP will display the attribute names as above and in the same order. The choice of IAs means that the DBA considers every conceptual property of every supplier and every conceptual property of every part, also the conceptual ones of every supply. We examine the rationale for later. Through, e.g., SP.SNAME, DBA models thus the conceptual property of every supply stated in natural language as: name of the supplier of the supply. And so on for every other IA in SP.

### Table S

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

### Table P

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

### Table SP

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

Figure 4 The S-P2 content. IA (proper) names and values are in Italics.

The IAs of I_sp determine their source relations as S and P. In each, the key attribute has the proper name of an SA in. Through Rule 2, E_sp is thus:

\[(IE1) \ E'_sp = \text{SNAME,... P.CITY} \text{ As PCITY From SP Left Join S On SP.S# = S.S# Left Join P On SP.P# = P.P#}\]

The left joins commute. Hence, one could specify manually E_sp equivalent to E'_sp using P first. Each clause defines a recursive join. Since S.S# and P.P# are keys, for every value of SP.S# and of SP.P#, there can be at most one matching tuple in S, as well as in P. Hence, for every tuple of S-P2.SP, E'_sp can produce at most one sub-tuple with values from its sources S and P. Recall that we required such property from every E_r.
For instance, for the first tuple of SP_B at Figure 4, i.e., with SAs S# = S1 and P# = P1, the recursive join clauses match the sub-tuple (Smith, 20, London, Nut, Red, 12, London), inherited from S and from P. The reason is S.S# = S1 for the source tuple in S and P.P# = P1 for that in P and no match for any other tuples in S or P. Similarly for SAs S# = S1 and P# = P1 etc. Observe that as the overall result, key (S#, P#) of SP_B, remains that of SP, as we required for every SIR R with respect to R_B.

2. We first illustrate our claim about the lower procedurality of the IE than of Create View for the CE-view and for QE-view. We recall that in a relational DB every relation must have a unique name. Suppose therefore that the DBA of S-P1 wished to create CE-view SP upfront, hence S-1.SP was created as SP_B instead. Since $E'_SP$ follows in Create Table SP all the attributes of SP_B, the least procedural Create View, i.e., the shortest, would be:

\[
(V_{SP}) \text{Create View SP As (Select SP_B.*, SNAME,…, CITY As PCITY From (SP_B Left Join S On SP_B.S# = S.S#) Left Join P On SP_B.P# = P.P#)};
\]

\[(V_{SP})\] is the shortest Create View SP, because of ‘*’ instead of the explicit enumeration of the IAs from S. Nevertheless, it still remains more procedural than $E'_SP$. Furthermore, it is relatively even more procedural than $I_{SP}$, several times in fact. If the referential integrity between S, P and SP is enforced, then CE-view creation could be slightly simpler:

\[
(V_{SP1}) \text{Create View SP As (Select SP_B.*, SNAME, STATUS, CITY, PNAME, COLOR, WEIGHT, CITY As PCITY From SP,S,P Where SP.S# = S.S# And SP.P# = P.P#)};
\]

But the change makes, by definition (1), $E_{SP}$ equally simpler. Hence CE-view remains more procedural analogously.

Next, providing the referential integrity between S and SP, and P and SP, the only choices for a QE-view of SP here are S.S# instead of SP.S# or P.P# instead of SP.P#. Each choice provides for the same default attribute names in the view, hence for the same queries as to CE-view SP and to SIR SP. None of these choices reduces nevertheless the procedurality with respect to CE-view for SP with the order of attributes as in Figure 3. Consider however the following variant of S-P2.SP. Suppose first that one does not rename P.CITY, hence its default name in SP stays P.CITY. Suppose next that Create Table SP orders the attributes as (S#, SNAME, STATUS..., P#, PNAME..., QTY) and that the referential integrity is again enforced. The new order does not reduce the procedurality of CE-view SP. But, the following QE-view becomes possible:

\[
(V_{SP2}) \text{Create View SP (Select S.*, P.*, QTY From S, P, SP Where SP.S# = S.S# And SP.P# = P.P#)};
\]

CE-view SP here is forced to enumerate all the attributes. Profiting from ‘*’, makes QE-view SP about twice less procedural than CE-view SP. This QE-view is also clearly the least procedural form of QE-view for any variant of SIR SP differing only by the attribute order. Easy to see that it becomes then also less procedural by about 1/3 than any revised $E_{SP}$. Nevertheless, one can use now for SP: $I_{SP} = S.#, P.#$. This one is less procedural than QE-view, by far even. Its non-procedurality is out of reach for every QE-view SP, obviously. Actually, Rule 6 allows even to declare $I_{SP}$ as $I_{SP} = $. There is indeed only one relation in S-P1 other than SP that has S# as key & same holds for P#. One can see thus our rationale for Rule 6. Finally, observe that the same holds for every SIR R supporting QE-view R and $I_{R}$ as above, while involving more source relations X1,X2... allowing for X1.*, ... in CE-view.

3. We now provide an example of $I_{p}$ for an IA defined by a VE and of utility of Rule 5. Consider that WEIGHT in SP2.P expresses the weight of a part in pounds, while in the conceptual model every part should also have its weight in KGs with precision of 1G. The DBA can model this property as the IA named WEIGHT_KG, following WEIGHT in P. To take care of, one could create instead of SR S-P2.P at Figure 3, SIR P with the same base, although named P_B, and with the following $I_{P}$, placed after WEIGHT in Create Table P:

\[
(W1) \ I_{p} = \text{WEIGHT\_KG AS (Round (WEIGHT * 0.454,3))}
\]
WEIGHT_KG is here in the secondary notation. An \( l_p \) using the basic notation instead would be the other way around, as for any SQL queries to \( P \), defining also WEIGHT_KG. \( l_p \) in (W1) is also locally inheriting. Through Rule 1, \( E_p \) is:

\[
E_p = \text{WEIGHT}_\text{KG} \text{ AS (Round (WEIGHT * 0.454,3)) From P_B}
\]

\( E_p \) is again clearly less procedural than CE-view \( P \). \( l_p \) is even more, about three times, as one may easily observe. Besides, WEIGHT_KG defined by (W1) could be a VA. \( E_p \) would be then slightly more procedural than the VA. One thus needs \( l_p \) to match the non-procedurality of the VA. Also, relation \( P \) with VA WEIGHT_KG would be a SIR with \( l_p \) above, without being explicitly considered as such. This conclusion obviously extends to every relation with VAs at present. Finally, the lower procedurality of VAs with respect to the CE-views is well-known rationale for that offering, often advertised as a view-saver. The same as ours for every \( l_p \) with respect to \( E_p \) more generally.

If declared as VA, WEIGHT_KG would not be updatable at any present DBS, as every VA besides. Decades old research showed however that one could change this state-of-the-art rather easily, [LV86]. In the meantime, IA WEIGHT_KG would not be updatable under our assumptions in S-P2 as well.

3. We now illustrate our statements about possibly greater procedurality of maintenance operation for CE-view \( R \) or QE-view \( R \) than for SIR \( R \). Essentially, both views may require the propagation of an alteration of \( R_B \). As perhaps the simplest case, suppose for both S-P1 and S-P2 that \( SP \) is an SR and that DBA did not anticipate the need for IAs of S-P2.SP. S-P2.SP can become SIR SP to accommodate these. A single Alter SP statement will then do (see the DDL for SIRs discussion below). The DBA of S-P1, supposed under SRV-model only we recall, needs to create a CE-view \( SP \) or a QE-view \( SP \). The DBA has to first rename \( SP \) into, say, \( SP_B \) again, as Figure 3 shows, using one Alter statement. Then, the DBA has to define the view. Both Alter and Create View statements must constitute an atomic transaction, to avoid any run-time errors for applications. SQL Begin Transaction and Commit brackets are thus necessary as well, together with the SQL Error Code tests for the eventual Rollback after each statement. All this leads to several (how many?) statements and quite a headache for the DBA, comparatively.

Similar reasoning with even greater difference to procedurality applies to every SA name change or addition or deletion. E.g., work out example of the DBA shortening QTY to Q, while either S-P2.SP or its CE-view \( SP \) already exists. In practice today, such alterations often create a run-time error. The rationale is that DBA often manages only the conceptual scheme. Views and applications are private to clients. The clients appear often not aware in real-time of alterations the DBA performs. In turn, the DBA is often not aware of all the views, CE-views included. A a run-time error of an application may then sometimes appear weeks or months after the alteration. This makes usually the debugging anything but obvious.

As perhaps an even more instructive example, suppose that DBA has to drop the existing default referential integrity. Suppose also that S-P1 DBA previously created CE-view \( SP \) with the joins in the WHERE clause we have shown and that S-P2 DBA used simply \( l_p \). Once there is no more referential integrity, S-P2.SP._B or S-P1.SP may get, e.g., the tuple (S10, P10, 50) with S10 and P10 not in S and P, respectively. To enforce the new requirement, S-P2 DBA has nothing to do, besides dropping the Foreign Key clauses for \( SP \). The S-P1 DBA needs to do the same for \( SP \). In addition however, s/he needs to alter CE-view \( SP \) to one with outer joins preserving \( SP \). For QE-view \( SP \), one has to alter it even more extensively, modifying the list of the attributes to that of CE-view \( SP \). Otherwise, for both views, the output of a query could be silently errored. Some queries would silently miss the above tuple, e.g., Select * From SP. Silent errors are of even greater concern than the run-time ones. However, for our purpose what matters mainly here is that for S-P1, unlike for S-P2, even if one adequately alters the CE-view or QE-view, each procedure requires again an atomic transaction with several statements.
4. We now illustrate our claim that SIR-model may provide for more faithful conceptual modelling than the SRV-model, while conform to all Codd’s postulates for a relational DB, except of having the conceptual scheme with SRs only. We recall that Codd’s original relational DB design rules, [C69], [C70], taught today to undergrads, propose the conceptual schema of a relational DB to be the (i) the smallest collection of 1NF SRs that satisfies the largest community of the DB clients, called *commonwealth* by Codd, as the reality model. For this purpose, Codd postulated also that (ii) every relation in this scheme should be free of, so-called in general, *storage and update anomalies* that could result from stored redundant values or an update, insert or delete. Initially, Codd termed these anomalies in fact *strong redundancies*. To fulfil requirement (i), we use n-ary relations with as many attributes as possible per relation, instead of, e.g., the binary relations only. We recall that the latter were the basis of once popular *semantic data model*, [A74]. Requirement (ii) boils down in practice today to BCNF for every relation with functional dependencies (FDs) only and to 4NF for those that present multivalued dependencies (MVDs). Relations in 4NF that would not be in 5NF are almost unheard of. For a community with specific needs, not met by the common scheme, views should produce the DB view. We recall that, for the relational model, this was the way to respect the ANSI-SPARC reference architecture, recommended in its time and till today, as mandatory for every data collection pretending to be a DB.

It’s truism to say that Codd’s principles became universal for the relational DBs. Codd used S-P1 to illustrate those. S and P relations did not lead to major controversies. This was immediately not the case of SP. Many found that the minimal three attribute model of SP is not faithful to any practical conceptual one. An actual supply always presents at least some other properties of the supplier and of the part supplied. These are the names at least or even all the properties of each ultimately. Adding any of the attributes modeling these properties to S-P1.SP would create the anomalies however. Hence, according to Codd, as known for decades, they should not be there, [D4]. Especially, since a query or a view can present to the client any of the “missing” attributes as IAs using the foreign keys of the minimal model whenever the need occurs. Many in the conceptual modelling community remained however unconvinced by this postulate, as widely known. The popular ER model resulted from, [C76]. It postulated instead that a tuple of SP in fact does not model an actual supply, but rather a *relationship* between two entities that were a supplier and a part. This approach introduced however its own shortcomings, never solved. E.g., considering a box of supplies modelled by a tuple as a relationship only and not as an entity, may moderately convince folks in charge of lifting it. Also, whether a marriage is an entity or a relationship never got a clear answer.

In S-P2 we expand S-P1.SP into a SIR with the base $SP_B = S$−P1.SP, and with all the others attributes of S and of P, as in Figure 3, we recall. Hence SP models now, in the greatest contrast to Codd’s scheme, not the minimum, but the maximum of the conceptual properties of S and of P that could characterize SP. Thus, SP scheme is now not only more faithful to the reality, but, even, cannot be more faithful with respect to S and P properties the actual supply could present (in the limits of 1NF relation however). All these are the IAs however hence avoid any storage anomalies. Likewise, no updates to $SP_B$ may create an update anomaly. Next, according to our general principles for SIRs, an update to any IAs of S-P2.SP applies to the sources in S or P that are always SAs. A view update to the CE-view of S-P2.SP would do the same, indeed. As known and one may easily verify, e.g., on MsAccess, this prevents any update anomaly for the view, hence for S-P2.SP. Likewise, S-P2.SP is free from the insert and the delete anomalies. Hence, S-P2.SP can serve as a conceptual scheme using the same best normalized SRs as S-P1. In contrast, the latter with S-P1.SP expanded with the SAs formally equal to IAs of S-P2.SP, would not be the one required for S-P1 under the SRV-model. Summing up, S-P2.SP respects all Codd’s postulates for a relational DB, except that by itself it is not an SR (only) and was even not among Codd’s constructs for the relational model.

CE-view SP of S-P2.SP added to S-P1, Figure 3, avoids the storage and update anomalies as well, of course. As a view however, on the theoretical side, it is not an element of a conceptual scheme, we recall. Besides, it is comparatively useless in practice. For the same modelling and operational
properties as SIR SP, its definition indeed has to be more procedural than every IE we have shown or equivalent.

5. Now we show on the example of S-P2.SP that a SIR may avoid or reduce the logical navigation in many queries to it, with respect to the equivalent queries addressing the conceptual scheme of the base-equal SRV DB. The term designates (i) every SIR X and SR Y where X_B = Y as SQL relations. Every SIR R is, in particular, base equal to its R_B. By extension, the term designates also (ii) an SRV DB D1 and SIR DB D2 where for every SR D1.X there is some base-equal SIR D2.Y or equal SR D2.Y and vice versa. Thus, S-P1 and S-P2 are base-equal. If we add to S-P1 either CE-view SP or QE-view SP, it would provide for the same queries, without the logical navigation or with a reduced one, as SIR SP. But, with the drawbacks of their creation and maintenance just discussed.

Consider thus the client needing P#, PNAME and QTY of every supply by Smith. For S-P2, the SQL query could be:

(Q1) Select P#, PNAME, QTY From SP Where SNAME = ‘Smith’;

An equivalent query to S-P1 could be:

(Q2) SELECT PName, SP.p#, SP.s# FROM S INNER JOIN (P RIGHT JOIN SP ON P.P# = SP.P#) ON S.S# = SP.S# WHERE S.SName ="smith";

Q1 has no logical navigation, as it selects all its data from a single relation. Q2 does it, as it must address all three S-P1 relations. The result is visibly by far (more than two times by our crude measure) more procedural than Q1. This stresses the practical importance of avoiding the logical navigation. A truism today, since the need was identified decades ago already, [MUV84].

In fact, Q1 avoids the logical navigation since it selects all its tuples in SP only. A query to S-P2 may however select a dangling tuple in S or P, i.e., a tuple that is not a projection of a tuple in SP. In practice, it would be a supplier not supplying any parts at present or, similarly, a non-supplied part. The logical navigation may be unavoidable. S-P2.SP may make it shorter however, compared to the equivalent S-P1 query. Even worse, a popular DBS, e.g., MsAccess, may be unable to execute the latter for an actual S-P1.

Consider indeed that the referential integrity exists at least between P and SP. Suppose also that that S-P2 client wishes the data in Q3 below for every supplier in S, even if there is no related supply for the time being, e.g., for S4 in Figure 4. Recall that every latter supplier would be modelled as a dangling tuple. Any SQL query to S-P2 expressing this wish requires the logical navigation between S and SP. The least procedural is that through half outer join:

(Q3) SELECT S.S#, S.SName, SP.p#, SP.PName FROM S LEFT JOIN SP ON S.S# = SP.S#;

For S-P1, the equivalent query would need the logical navigation also between P and SP. The following query could do:

(Q4) SELECT S.S#, S.SName, SP.p#, SP.pqy, P.PName FROM S LEFT JOIN (P inner JOIN SP ON P.P# = SP.P#) ON SP.S# = S.S#;

Q4 is almost two times more procedural than Q3. It is enough to require a typical client to think for a couple of minutes at least how to formulate it. Unlike it is for Q3. In particular, since the left and the inner join do not commute in Q4, illustrating the rationale for the current SQL standard we spoke about. More generally, because of this and other annoying properties, [DD91], the navigation through outer joins gained the well-earned reputation of being even more awkward than through the inner ones only. Perhaps, that is why, e.g., the MsAccess outer join processing remains bugged.

---

3 Perhaps surprisingly, MsAccess is the most popular relational DBS by number of licensees, allegedly in hundreds of millions.
since its earliest version. As one result, if S-P2 was implemented over MsAccess as we discuss in Section 4, Q3 would work. Q4 in contrast, cherry on the cake, would have its execution refused. A message would explain that it contains a non-supported join expression. That is clearly a bug, as Q4 conforms to SQL standard. As one may easily experiment, the bug would be independent of our implementation.

Query Q1 was a specific one. As final point here, observe however that S-P2.SP contains all the attributes of S-P2, although a few are renamed. We call it a universal SIR, as a new type of the universal relation [MUV84]. S-P2 avoids therefore the logical navigation through S-P1 not only for Q2 and its variants, but for every query addressing several relations in S-P1, provided its S-P2 equivalent selects all the relevant tuples in S-P2.SP only. Stated differently, the S-P1 query selecting tuples in SP and S or P should select the projections of some S-P2.SP tuples only. Still in other terms, the S-P1 query should not select dangling tuples in S or P. Most queries to S-P1 stated in the literature as representative of practical ones are of this kind, e.g., in [DD91] & later editions. We recall that dangling tuples in S model indeed suppliers without any supply and in P model not-supplied parts, clearly less usual cases than otherwise.

6. We now show examples of the two types of IAs we discussed, namely inheriting from sources in other relations and through an aggregate function, perhaps in a subquery of a scalar function. No such IA can be a VA presently. The IEs we show are implicit ones with the usual procedurality of a VA if they could be so. As for a VA, such an IA may shield the client from declaring it alternatively within an equivalent query to the base equal relations. Even more, it may again make executable a query whose equivalent to the base equal relations is perhaps not at a popular DBS. Alternatively, as for a VA again, it may save to the DBA the drawbacks of a CE-view or even bigger here as it will appear.

For first IA, consider STATUS stored attribute in S-P2. Imagine that, behind the scene, at one point in time, DBA decides to calculate it, e.g., as the total quantity delivered by a supplier divided by hundred and rounded to its integer part. Thus, the supplier of 100 - 199 parts will have status 1 etc. The status of the supplier not supplying any parts for the time being should be null. Continuing having such STATUS as an SA is clearly not the most practical choice. In S-P1, likely the most practical approach would be to either calculate STATUS value in every query needing it or (i) to drop STATUS from S and rename it as, say S_0, (ii) to Create the view S with S.* and STATUS calculated as above. This view would be in fact the CE-view S of SIR S with STATUS as an IA. For this purpose, since S-P2 is a SIR DB, the DBA may alter S, using the Alter statement for SIRs we present later. The DBA may then (a) drop STATUS, (b) rename S as S_B since it plans to create SIR S and (c) recreate STATUS as the I_S:

\[ I_S = \text{STATUS As Round (SUM (QTY) / 100)} \]

STATUS applies an aggregate function. According to Rule 3, \( E'_S \) is:

\[ E'_S = \text{STATUS FROM (SELECT S_B.*, (Select round (SUM (QTY) / 100) From SP_B Where S_B.S# = [S#]) AS STATUS FROM S_B)}; \]

\( E'_S \) refers to SP_B since QTY is an SA, hence this choice avoids the circular referencing. The latter would occur, including among the CE-views of S with STATUS and CE-view SP, if SP was used instead as the source relation in \( E'_S \). Next, \( I_S \) almost minimizes the procedurality of any STATUS scheme. It contains only the proper name and the VE. \( I_S \) is consequently here several times less procedural than Create View for CE-view S. Besides, since the VE employs an aggregate function and refers to attributes beyond S, STATUS cannot be a VA at present. Hence, having S-P1 on a DBS with VAs would not help to avoid either declaring STATUS in every query needing it or creating and maintaining CE-view S.

The result for SIR S with calculated STATUS would be values for suppliers S1...S4. STATUS would be null in contrast for S5. Indeed, this supplier does not supply anything at present. The client of SIR S may get S, through the simplest SQL query, namely:
(Q3) Select * From S;

In contrast, if S was a base equal SR, i.e., S without STATUS, the least procedural equivalent query, also serving eventually as the scheme of the CE-view of our SIR S and (partly) as $E_S$ alternative to $E'_S$, would be:

(Q3.1) Select S#, SNAME, CITY, (Select Int (SUM (QTY) / 100) From SP Where S.S# = S#) As STATUS From S;

Q3.1 is clearly more procedural than (Q3) by far. We recall also that the non-procedurality of queries like Q3 charmed in its time the DB community to SEQUEL, former name for SQL, we recall. In contrast, clients usually dislike Select clause sub-queries, (as well as From clause sub-queries as in $E'_S$). Many have hard time to even figure out the semantics of these. Some could rather equivalently apply the popular Group By instead. The result is however even more procedural.

Cherry on the cake, Q3.1 queries may not work on a popular DBS, e.g., MsAccess, if one adds Order By STATUS clause. Creation of CE-view S followed by Q3 with Order By is mandatory. At the procedural cost obviously higher than of every IE above.

The second example we’ll now discuss shows the IA of S defined through a subquery in $I_S$. Besides, it shows that a recursive join may be not an equijoin. Consider that DBA decides to declare STATUS as above and another IA named RANK. For each supplier $s$, if STATUS ($s$) is not null, then RANK ($s$) is one plus the number of Suppliers with STATUS higher than that of $s$. Otherwise, RANK ($s$) is null. The following clause of $I_S$, declared in Create Table S right after the above clause STATUS scheme, fulfils our requirements and has the secondary form like a VA:

\[
RANK \text{ As IIF (status is not null, (select count(*) +1 from S X where x.status > s.status), null)}
\]

$I_S$ consists now of both clauses. IIF is the scalar function, e.g., of MsAccess SQL dialect. The clause contains a subquery, but it still defines RANK through a VE, perhaps surprisingly to some. The join within the subquery, is recursive and not an equijoin. The VE is locally-inheriting, i.e., inherits values from S only. Also, subquery refers to S and not to S_B since Status is an IA. Through Rule 4, we now have:

\[
E'_S = STATUS, RANK \text{ FROM (SELECT S_1.*, IIf(status Is Not Null,(select count(*) +1 from S_1 X where x.status > s_1.status), null) AS Rank FROM (SELECT S_B.*, (Select round ( SUM (QTY) / 100) From SP_B Where S_B.S# = S#) AS STATUS FROM S_B) AS S_1)};
\]

Like STATUS, RANK also cannot be a VA at present at any popular DBS we are aware of. Finally, to get the feeling of utility of SIRs in this case, consider the simple query of obvious interest:

(Q4) Select * From S;

As it was for the similar query for STATUS, to the best of our knowledge, because of the already outlined limitation of SQL, at least some popular DBSs do not allow for a single equivalent query to the base equivalent relation, i.e., to S (S#, SNAME, CITY). The only solution is again to rename S, e.g., to S_B and create at least CE-view of S without RANK, i.e., extending S_B with STATUS only, defining then RANK in the query. As always, the procedurality of Create View S with STATUS only, is then greater than that of $E_S$ and, to even larger extent, greater than that of $I_S$. However, CE-view S integrating STATUS would not suffice again if one attempts to add Order By RANK clause to RANK defined in the query. CE-view of S with STATUS and RANK followed by Q4 with Order By RANK is necessary. As this one has to result from two Create View statements, it is, comparatively to only one view for CE-view S with STATUS alone, even more procedural than the $I_S$. The maintenance is also more expensive and error prone. E.g., suppose that one renames one day STATUS to, say, C_STATUS. For SIR S, a single Alter Table S suffices, renaming every occurrence of STATUS in $I_S$. For views, one has to alter both views. If one overlooks the one for RANK, an execution error of any queries to view S or any applications using it will result. As often in practice, in possibly the most inconvenient moment.
2.5 DDL Statements for SIR-model

Create Table for SIRs defines SAs and IAs basically as above illustrated. We suppose the syntax for SAs is that of the kernel Create Table. Also, as mentioned for S-P1, we suppose the Foreign Key clauses possible for SIRs as well. Furthermore, Create Table for SIR-model may in particular define a stored (only) table. More generally, every DDL statement for SRV-model enters the SIR-model DDL by definition. As one wisely said, who can more, can less, Figure 1.

For views, in particular, we suppose the reuse is of the kernel’s Create View DDL statement. The rationale is the absence of the recursive join(s) in a view. Reuse of Create View seems then easier to implement than some view scheme only specific syntax in Create Table for SIR-model. We come back to this issue in Section 4.

The other usual SQL DDL statements are, we recall, Alter Table, Drop Table, Drop View and Create Index. With respect to the generalization to SIRs of Alter Table, we suppose that it keeps its kernel dialect clauses Add, Alter or Drop with all their capabilities, to operate on any SAs. We add the extension specific to SIRs, namely that Alter applies also to IAs. Such extensions exist besides already for VAs, differing somehow however between DBSs. The alteration may add an IA, after any specified SA or IA or before the first attribute, or after every existing attribute. The latter is the default. It may also change or drop an existing IA. However dropping an IA requires caution. It may indeed render From and its follow up clauses invalid. Therefore, for every SIR, we allow Alter Table R to contain Alter View clause, defining entirely new IE. More precisely, we suppose the clause acting implicitly on all the IAs, as clause Alter View R_V in the kernel SQL would act on view R_V. In particular, except for the implicit naming, we suppose the entire syntax of the latter applying to the former.

For ADD clause defining an IA through VE, Alter Table accepts the basic or the secondary notations. The secondary notation there is backward compatible with Alter Table of every VA-supporting DBS at present. Besides, some popular DBSs have limitations on ADD clause, with respect to above semantics. E.g., SQL Server allows for adding a VA only as the last attribute.

Next, Alter Table R may drop all IAs. SIR R becomes then SR R. ADD clause may, inversely, add an IA, including $E_R$ to SR R, making it SIR R. As usual for an SR, we prohibit however every Alter Table R to drop every SA of SIR R, i.e., possibly transforming then SIR R into view R. If the need occurs, the extended Drop Table R simply drops as usual the scheme of R, IE included, and, eventually, the content of R. The operation should not of course violate the referential integrity. It may thus, as usual for SRs, trigger a cascade to other SIRs or a refusal of the statement if a violation results. Furthermore, for every SIR DB, if one should alter some view R, one may do it through Drop View R followed by Create View R with the new scheme or Alter View, if the kernel DBS supports that statement. In addition, for SIRs specifically, an alteration of any view R should also have the potential to evolve it into SIR R. At present, we suppose Drop View R followed by Create Table R. This is obviously the simplest to put into practice procedure. Finally, we consider that Create Index statement for a SIR reuses the syntax of the kernel. That one may prohibit or allow indexes on IAs.

Example 2.

1. DBA adds to S-P1.P the already discussed WEIGHT_KG. S/he also adds WEIGHT_T converting WEIGHT_KG further to tons. For application dependent reasons, WEIGHT_T should precede in the scheme WEIGHT_KG.

   Alter Table P Add After WEIGHT WEIGHT_T As WEIGHT_KG / 1000,
   WEIGHT_KG As Round (WEIGHT * 0.454) ;

   Both IA schemes are in secondary notation and could be VAs. As the result, Alter modifies SR P into SIR P with $I_P$ constituted from both IA schemes.

2. We change SA STATUS in S-P2.S to IA STATUS we discussed.

   Alter Table S Modify STATUS As Select Int (SUM(QTY)/100);
The Alter Table syntax here is that of MySQL, except that the latter does not support our Select clause. The intended result is that SR S becomes SIR S with obvious $I_s$. If $E_S$ is necessary instead, Modify allows for From clause as well. In every case, the stored values of STATUS get dropped. In practice, a warning should therefore precede the actual execution of the statement. Below, we refer to S-P2 with P and S altered as discussed and with another attribute introduced later on and named RANK, as to S-P3 DB. Notice that S-P3 scheme has no more SRs, only SIRs.

### 2.6 Data Manipulation

Any SIR is a 1NF relation, by definition. The relational algebra operators of SRV-model operate on 1NF relations as defined by their mathematical model, Figure 1. Whether an attribute is a SA or an IA is immaterial to the operators. Each applies thus as is to SIRs as well. One may project, select or join thus any SIRs. The same holds for any SQL Select statements. Including these with value expressions, scalar and aggregate functions, the special clauses: Top k, Group By, Order By...

In short, SIRs do not require extensions to any current DML statements. For a modification of an SIR, i.e., the SQL Insert, Update or Delete statement, each statement should act as we already discussed in Section 2.1, i.e., as it would act on the CE-view.

Example 3. The simplest for SP SQL Select statement Select * From SP would show all the SP values, of all SAs and of all IAs in Figure 4. Supposing MS Access SQL as the kernel dialect, would make the statement Insert SP (select ‘S4’ as $S\#$, P4 as $P\#$, 100 as QTY); adding the tuple with these stored values and with all the virtual IA values. The statement Update SP set QTY = 250 where $S\#$ = ‘S1’ and $P\#$ = ‘P1’; should normally succeed, updating one stored value in SP. The statement: Update SP set QTY = 250, CITY = ‘Paris’ where $S\#$ = ‘S1’ and $P\#$ = ‘P1’; also succeeds. The change to CITY may propagate to S indeed, as the CE-view of SP would let it to propagate, e.g., under MsAccess. The side-effect that might surprise would be the city change for every other supply by S1. Next, an update of SP.STATUS succeeds provided S.STATUS is an SA. But if S.STATUS is the above defined IA, any update to it in SP or even S must fail. Finally, the statement Delete SP Where $S\#$ = ‘S1’; would erase as usual physically from the DB all the values of the stored attributes in the selected tuples in Figure 4. Formally, it would also erase all the inherited values.

### 2.7 Usability of SIRs & Related Work

Our motivating example illustrated how SIRs may make a relational DB less-procedural, hence more usable by usual meaning of this qualifier. First, with respect to queries to S-P1, the equivalent ones to S-P2 and S-P3 were free of logical navigation or with reduced one, or were freed of selected VE. If S-P1 should provide for the same queries, one would need to create the CE-views S, P and SP we have discussed. But then, every IE in S-P2 and S-P3 was less-procedural than Create View of its CE-view. The views were also more procedural to maintain. Same reasons motivated VAs, already decades ago. The examples illustrated how implicit IEs seamlessly, i.e., with the same usability, integrate that capability and even generalize it to VE impossible to be VAs at present.

SIRs are also as in footsteps of the universal relation idea we already hinted to, also known since decades. That relation should have as attributes all these in the DB as well as all the values so that no query to the DB should need the logical navigation. Through often passionate, although now rather extinct interest in the topic there were various proposals for universal relations, [M4]. None apparently made to the industry, in the sense that it is up to DBA only or to the client to create a universal SR or, rather, a universal view. As we signaled S-P2.SP was actually in the footsteps of this effort, being however, as a SIR, also a new type of a universal relation. The gain is again lesser creation cost, hence higher usability, than that of the (universal) CE-view or VE-view. S-P2.SP made logical navigation free not only for our example Q1 query, but generally for every query to S-P2 etc., not selecting a dangling tuple in S or P. For a latter query, the logical navigation may get reduced at least, as our example showed. We will also show soon that this usability generalizes further, to every DB without a multivalued dependency (MVD) in fact, [F77]. In presence of an MVD, selected queries
not aimed at dangling tuples may nevertheless need the logical navigation. This one can however get reduced as well.

Besides, as we also mentioned, SIR-model is backward compatible with the SRV-model. The latter is indeed a strict sub-model of the former. The trivial condition to stay within the current model with the SIR-model is simply to refrain of IEs. In our example, it one can still stay with S-P1 instead of profiting from S-P2 etc. Switching to the SIR-model is safe in this sense. No loss of any current capabilities of S-P1, when expanded to S-P2 etc., may result from. Every application of it on a popular DBS should continue to run thus, if this DBS gets capabilities to manage SIRs as well.

It is notorious that the ”biblical” S-P1 DB was the mold for most of practical ones. We may thus reasonably expect all the nice behavior discussed extending to most of practical DBs as well.

From the theoretical standpoint, we already recalled through our examples that insufficiency of the conceptual modelling by normalized SRs only were known almost since the inception of the relational model. There is no more such trouble with our example SIRs. These are intended upfront for the conceptual schemes and absent from the SRV-model. No more need to twist the purpose of views to enrich the actual conceptual scheme. Besides, no more need for the data dictionary identifying these views among usually many more. Likewise, there is no more need for the ER modelling, with its own troubles. Finally, one observed perhaps for all SIRs in S-P2 etc., that their bases were normalized as required by the SRV-model, i.e., were as in S-P1. Actually, we transpose these normalization principles to SIR-model formally soon.

Finally, one could observe from the example that the inheritance model for IEs is the original one of the relational model. That is, the foreign key value is the surrogate of the inherited object that is the one with the primary key equal to. This model characterizes also most of popular DBSs. We should mention however that some, so-called, object-relational DBSs proposed different models in in 90ties. The open-source Postgres DBS is the most prominent survivor of this trend, [SM96], [P]. Those models of inheritance should not be confused with that of IEs. E.g., Postgres has a dedicated INHERITS clause in its Create Table, creating a sub-relation (sub-table) from the entire inherited relation etc.

3. SIR-model Relational Schema Design

At the peak of its glory, four decades ago, research on SRV-model issued countless proposals for somehow optimal (conceptual) relational scheme design. In practice, remain basically the NFs, the Heath’s Theorem and the Fagin’s one. SIR-model (conceptual) scheme design rules obviously should preserve that effort for the (stored) bases of SIRs. The new need is the integration of IEs with. The SRV-model basic design rules need to be restated accordingly. Such a proposal is the subject of what follows.

The relational scheme design goal was the removal of the anomalies. We preserve the goal for SIRs. Through IEs, we aim in addition at avoiding the logical navigation and specific VEs to queries. We continue with S-P2 as the motivating example. We first restate the NFs for SIRs. Next, we restate accordingly the Heath’s and Fagin’s theorems. The restated theorems generate the same lossless decompositions, but with SIRs as projections, instead of the present ones. The benefit from the restated Heath’s theorem is the total absence of the logical navigation, otherwise necessarily generated by the (original) Heath’s theorem. The restated Fagin’s theorem avoids in contrast the original logical navigation only partially. Both results take care of the IAs defined through the value expressions.

3.1 Normal Forms

The basic design rule for a relational DB scheme under SRV-model is the respect of the normal forms (NFs). We recall that these are 1-3NF, BCNF, 4-5NF. Any relation in 5NF is in 4NF that is in BCNF etc. Every relation in SRV-model is by default in 1NF we also recall. Next, relations in 4NF that would not be in 5NF are rare, what makes BCNF and 4NF the most useful in practice. E.g., SP (SH, P#, QTY) in S-
P1 is in BCNF, while SP’ (S#, SNAME, P#, QTY) with stored attribute SNAME would not be. We’ll give examples of 4NF later. Each NF eliminates some of anomalies we already signaled. E.g., SP’ would need to store SNAME redundantly. Also, SNAME update could erroneously create two different names for same supplier. This could contradict S, where SNAME is anyhow already. Using SP instead, avoids the trouble.

First, recall now that any SIR is in 1NF by definition. Hence no need to restate this NF. The other forms have to be restated for SIRs. Observe in this context that the above anomalies of SP’ would not exist for a view SP’. We therefore state that an SIR R (B, V) is in $i$NF or BCNF, iff B is in $i$NF or BCNF. Actually, since R can have null values that were not in the original Codd’s model, we implicitly consider as usual today that NFs apply to relations possibly with as well, e.g., as formally in [JS90].

Example 4. SP in S-P2 is in (extended) BCNF and 4NF, as well as in 5NF even. Indeed, the projection SP [S#, P#, QTY] on all and only stored attributes conforms to these NFs. Same happens, trivially, for S and P in S-P2. However, as mentioned, the stored relation SP’ (S#, SNAME, P#, QTY) would not be in BCNF. But, an SIR SP’ with IA SNAME in turn, would be. More generally thus, if, for any reasons, SNAME or any other IA in SP in S-P2 was rather a stored attribute, SP would cease to be in BCNF etc.@

3.2 Sir DB Schema Design

We recall that at present, i.e. for a SRV-model DB, this process aims on a relational DB the (conceptual) scheme with possibly least number of relations free of anomalies. Usually, it means that every relation has to be proven as in 4NF or as at least in BCNF. The former need occurs if a relation presents a (non-trivial) multivalued dependency (MVD). The latter, by far more frequent, characterizes schemes with the functional dependencies (FDs) only. The least number of relations means the grouping of all attributes functionally dependent on the same one(s) into possibly one relation, with the latter as the primary key. Possibly means here the respect of a myriad of other less or more fuzzy criteria, e.g., not “too many” null values for some attributes.

Designing a scheme is furthermore usually a many-steps process. It focus primarily on the logical navigation, avoiding selected VEs is a secondary goal. Ideally, we optimistically start with the attempt of the universal stored relation, say U, for the entire DB. U has the potential of avoiding the logical navigation to every query, as all the attributes are in. Unfortunately, practical chances for U in 4NF are zilch. We usually perform then a decomposition of U into projections, i.e. we suppose that the DB consists of these projections as SRs instead. The decomposition must be lossless, producing the projections whose equijoin equals the decomposed relation. Any projection may end up proven in 4NF or proven in BCNF and free of any MVDs. It is then in 4NF thus as well. Or, a projection may not end up so. We decompose any such projections again. We continue, until every projection is anomaly-free, in which case we create it as an SR. The notorious cost with respect to U is the logical navigation over the projections for most queries.

As known, the Heath’s and Fagin’s decomposition theorems are two the most used. The former may help with annoying FDs. The latter removes MVDs. Each theorem decomposes a relation into two projections. The resulting scheme has the least possible number of normalized relations for the DB, i.e., is of the smallest size and the optimal one in this sense. Actually, as only a few seemingly know, in presence of both MDs and FDs, Fagin’s theorem must serve first. Otherwise a sub-optimal decomposition may result, meaning the scheme with more stored values than otherwise needed in a scheme nevertheless optimal in the sense we just defined. Even otherwise, there may be several decompositions that are all optimal in the discussed sense. So-called independent projections are preferable. Their known advantage is the preservation of the FD-cover. Rissanen’s theorem testing the independence of the chosen projections may help.

We now generalize these principles to the SIR-model, i.e., U and the projections may be SIRs. Such schemes were out of scope of the original methodology, of course. In other words, even U may
contain IAs, e.g. the aggregate ones we showed. For FDs and MVDs used for the decompositions, we nevertheless originally assimilate all these IAs to SAs. We apply to the projections the restated NFs. Then, in contrast, we consider any IA again as is. For the Heath’s and Fagin’s theorems rested for SIRs, our goal is that the decomposition of a SIR, say R again, is not only lossless, but also that at least one of the projections inherits some, possibly all, attributes of R. The result aimed on is that the lossless decomposition possibly does not cost us the logical navigation through the projections, unlike for the original theorems. We leave for the future eventual restatement of multitude of other rules aiming on best schemes, [D12], [F11], [V11], Rissanen’s work included.

The major gain that will appear below is that, for the same stored relations and the same size optimal schemes for a DB altogether, the DB using SIRs effectively spares the discussed logical navigation. More precisely, the optimal SIR scheme will be always as follows:

(a) The SRs whose schemes are projections resulting from original Heath’s and Fagin theorems become bases of SIRs resulting from the restated theorems or remain the same SRs.
(b) In the absence of MVDs, no restated decomposition creates the logical navigation through the projections.
(c) Otherwise, a restated decomposition removing an MVD still spares or at least reduces the logical navigation for some queries addressing the projections, but not for all such queries.
(d) The latter queries in (c) should be uncommon.

Indeed, first, the Heath’s theorem states, we recall, that for any stored relation ABC (A, B, C) and an FD A → B, the decomposition AB (A, B) and AC (A, C) is lossless. That is: ABC (A, B, C) = AB (A, B) Join AC (A, C). In practice, as known well, we may have several choices for A, B and C. As every decompositions doubles A, for stored relation ABC, it is usually wise to choose A with fewest attributes. Likewise, A should be the primary key of AB. B does not depend then on any proper subset of A so AB is in 2NF at least. Also, for reasons previously invoked, we should hunt for the largest B. We may end up nevertheless with AB not in 3NF at least hence B may get decomposed in turn, etc. With all the discussed principles in mind, we restate the theorem for an SR and a SIR as follows. Let ABCD with ABC being as for the original theorem and D denoting IAs or D = ∅. In the former case, ABCD is a SIR. In the latter case, it denotes the SR ABC. We decompose ABCD into AB (A, B) and ABCD (A, B, C, D). Here, B denotes B inherited through IA = B, generating thus the implicit recursive equijoin clause: On AB.A = ACDB.A for E′ACDB.

As the original decomposition, the restated one is thus also into two schemes and clearly lossless. But, while AC was an SR, ACD can be a SIR and ABCD is a SIR with base AC. This decomposition is possible only for the SIR-model. It produces AC as the stored projection of ABCD, as the original theorem does for ABC. But, it also preserves all the original attributes together, although as A, B, C, D. It avoids thus, as promised, the logical navigation to any queries selecting the original attributes.

Figure 5 illustrates both decompositions. At Figure 5.a, query Q (B,C) selecting B and C, say for each tuple that was in ABC, is logical navigation free for the restated decomposition. It may indeed access only one projection. For the original decomposition however, Figure 5.b, the equivalent query Q’ cannot be. It has to join the projections. Query Q remains possible, but only provided the expense of creating the CE-view ABC from the projections. For the reasons already abundantly discussed, the creation of this view should be always more procedural than IA = B, i.e., IA = B for D = ∅. Observe finally that query Q (B,D) to ABCD, making sense for the restated theorem only thus, would also remain free of the logical navigation after the decomposition.

The decomposition thought the Heath Theorem is usually called lossless. The term appears partly misleading in the light of the restated decomposition. Each original projection loses indeed some attributes of the decomposed relation. Only the recovery through the join is lossless. It is the reason for the logical navigation in every query to attributes in different projections. Also, it is the reason for the join view providing the recovery, necessary in SRV-model to avoid the navigation. Only the
restated decomposition should qualify as lossless, since one of the projections keeps all the original attributes and values.

Figure 5. Restated (a) and original (b) Heath’s decompositions, as well as, (c) restated Fagin’s decomposition.

In practice, we start the decomposition, by Heath’s or Fagin’s theorem, from the optimistic assumption of a universal relation, usually noted U, for the DB. The decomposition of U until the final best normalized schema usually requires several steps. E.g., from U for S-P1, till its final 3-relation scheme, there are obviously two successive Heath’s decompositions. Observe that if all the decompositions are the Heath’s ones only, then the restated theorem always produces one projection that is a universal SIR. Every query to the DB can address that one and is then logical navigation free. Again, the universal view, formally equal to U, would lead to the same result, but at a higher definition cost, as we have seen for S-P2.

As we mentioned in the introduction and seen in the motivating example, finally, the freedom from the logical navigation provided by the restated decomposition(s) holds nevertheless only until, as often, dangling tuples enter a stored projection. The join is no more lossless. It will appear, but is easy to observe already, that the content of S and of SP at Figure 4 illustrates the case, because of the dangling S4 tuple in S, already mentioned in the motivating example. Queries of obvious practical interest may require the logical navigation again. Even so, ABCD may still reduce that one with respect to AC only. Actually, queries Q3 and Q4 already pointed out this property, although we did not refer there to Heath’s theorem.

Next, Fagin’s theorem also states that in presence of MVD A ->> B | C in the presumably stored relation ABC (A, B, C), its decomposition into AB (A, B) and AC (A, C) is lossless. Now, suppose B’ being a (perhaps empty) subset of B such that A -> B’ and let C’ be a (perhaps empty) subset of C, where A -> C’. Actually, we may about always expect either B’ or C’ non-empty, but not both, as in the example that follows. We restate the theorem as follows. Suppose ABCD a stored relation or a SIR as above stated. The restated decomposition creates ABDC’ (A, B, D, C’) and ACDB’ (A, C, D, B’) where \( l_{ABDC'} = C' \) and (ii) \( l_{ACDB'} = B' \). As Figure 5.c illustrates, C’ and B’ avoid the logical navigation for any query to BD and C’ or to B’ and CD in the projections, unlike for the original decomposition. Only a query to B/B’ and C/C’ still needs it. As notorious, the result of such a query should be however typically awkward, as B and C are supposed mutually independent. Such query are therefore unlikely.

Unlike for FDs only and the exclusive use of Heath’s decompositions then, for an MVD in contrast, our decomposition does not avoid completely the logical navigation that the original Fagin’s decomposition may create. If we wanted to, as long as there is no dangling tuples, a full auxiliary view scheme of ABC, as at Figure 5.b, would be the only option for both decompositions. But, without this price, our decomposition do limits the navigation to fewer queries. Furthermore, as it will appear the remaining queries should usually have the logical navigation through the final optimal scheme of the DB partly limited, i.e., with fewer joins. Notice that the restated theorem again conserves each original stored projection as is or as the base of one of the SIR projections.
On these foundations, the generic schema generation algorithm for SIRs is quite analogous to that for the stored relations only. More precisely, U remains the starting point, except that it may have IAs upfront. From there, we perform the same, wisely chosen, successive decompositions eliminating MVDs and “annoying”, i.e., anomaly creating, FDs. However, at each step, we now use a restated theorem instead. If we face both dependencies, the restated Fagin’s theorem works first. We naturally end up with the same stored relations, hence the same size scheme, but also with less logical navigation, as claim (b) states. If there are no MVDs, we remove the discussed logical navigation entirely, as claim (a) states. Finally, the rationale for claim (c) is that in a real-life DB, MVDs are rare with respect to annoying FDs. Also, B’ or C’ usually have several attributes, unlike B/B’ or C/C’. Even for a decomposed MVD, most queries to the projections should be normally logical navigation free as well.

The following example illustrates all the debated points.

Example 5. The biblical S-P1 scheme results from Heath’s theorem only. Similar schemes are countless in practice, as widely known. Our scheme in Example 1 would need the restated Heath’s theorem only. To illustrates also the restated Fagin’s one, we modernize S-P1. Each supplier has now one or more contact email addresses. Each address may serve for any inquiry about the supplies or the supplier itself. Each address is the value of new stored attribute EMAIL. Every address is for one and only one supplier. We redesign the S-P scheme under SIR-model accordingly. We call the result S-P4.

We start optimistically with the universal relation U as the only SR. In short notation we have:

\[ U \text{ (EMAIL, S#, SNAME, STATUS, CITY, P#, PNAME, COLOR, WEIGHT, PCITY, QTY).} \]

Notice the necessarily different names for the supplier and part cities, unlike in S and P of S-P1 or S-P2. U is potentially the optimal SR for S-P4, unless proven otherwise. What’s easy, since EMAIL already introduces the MVD: S# -&gt; EMAIL | (SNAME, CITY, STATUS, P#...QTY). U is not in 4NF thus. Regretfully, U cannot be the optimal S-P1 scheme. We have to decompose it. We have MVDs and obviously FDs. We start with the restated Fagin’s theorem. The decomposition may create two projections, intended as SRs, say SE and SP as follows, with C’ = (SNAME, STATUS, CITY) and B’ = \( \emptyset \):

\[ SE \text{ (S#, EMAIL, SNAME, STATUS, CITY), SP \text{ (S#}, \text{ SNAME, CITY, STATUS, P#...QTY).} \]

SE is now a SIR, with IAs in Italics and \( I_{SE} = \text{SNAME, STATUS, CITY} \). We thus have also:

\[ E_{SE} = I_{SE} \text{ From SE Left Join SP On SE.S# = SP.S#;} \]

with \( I_{SE} \) denoting in short its actual extension. The base SE_B (S#, EMAIL) would be the projection for the original Fagin’s decomposition. SP is the same for both decomposition. SE is in the restated BCNF. It would not be if any of its IAs, e.g., SNAME, was an SR. The IAs of SE spare the logical navigation to every queries to EMAIL and to any of its IAs, unless one inserts a dangling tuple into SE. Otherwise, these queries would necessarily navigate over SE_B and SP. In contrast queries selecting emails and an attribute in SP that was not inherited in SE would still need to navigate through the projections. We come back to these queries later on. We will show that practical queries should require lesser navigation anyway, backing up our earlier claim.

SE has no more MVDs, hence it is also in 4NF. SP has no more MVDs neither. But, is not in (restated) BCNF (hence neither in 4NF). The restated Heath’s theorem applies. For all the already discussed reasons, we choose the following decomposition, with S# as A, in particular since it is a single attribute key:

\[ S \text{ (S#, SNAME, STATUS, CITY), SP \text{ (S#}, \text{ P#, PNAME...PCITY, QTY, SNAME, STATUS, SCITY);} \]

Here, the IAs in SP may result from \( I_{SP} = S.#, \) with the implicit join S.SP# = SP.S#. The projection SP is again an SIR, with \( I_{SP} \) defining B. All the tentative SAs of the decomposed SP remain thus preserved in the projection SP, as the IAs sourced in S. Notice that this does not change anything for SE scheme. S
is as for the original decomposition. It is in BCNF, hence can be definitively made an SR. SP however still isn’t in restated BCNF. Its projection on the stored attributes isn’t in BCNF in SRV-model indeed, given the FD: P# -> PNAME, COLOR, WEIGHT, PCITY. We thus apply the restated Heath’s theorem again. One gets SP decomposed to:

\[
P (P\#, PNAME, COLOR, WEIGHT, PCITY) \text{ and } SP (P\#, S\#, QTY, SNAME, STATUS, SCITY, PNAME, COLOR, WEIGHT, PCITY).
\]

Here, the IAs of SP result now from revised \(I_{SP} = S.\#, P.\#\), with the implicit join SP.P# = P.P#. Now S-P4 has every relation in BCNF, hence in 4NF, as there are no more MVDs. The optimal scheme is as follows, with PCITY supposed (arbitrarily) renamed to simpler CITY, becoming PCITY in SP only. We underlined the primary key stored attributes.

\[
S (S\#, SNAME, STATUS, CITY),
P (P\#, PNAME, COLOR, WEIGHT, CITY),
SE (S\#, EMAIL, SNAME, STATUS, CITY) \quad /* IAs from S
SP (P\#, S\#, QTY, SNAME, STATUS, S.CITY, PNAME, COLOR, WEIGHT, P.CITY As PCITY) \quad /* IAs from S and P.
\]

Notice that SP scheme is that of S-P2 from Example 1. Because of IEs, as for S-P2, most practical queries to S-P4 are now clearly logical navigation free. However the already signaled queries to SE and SP are not. Some of these queries, e.g., select every P# supplied by supplier with given EMAIL, seem of marginal interest only. Clients in practice need also names. Then, the restated decomposition still reduces the logical navigation by two joins otherwise necessary, i.e., SE with S to get SNAME and SP with P to get PNAME. We may thus reasonably expect at least some logical navigation spared for practical queries to SE and SP together and for most of such queries to projections of a decomposed MVD in general.

Also, if we did not start decomposing U with the Fagin’s theorem, but with Heath’s one, the result would be the sub-optimal we spoke about. Indeed, the first decomposition of SPE could use the FD: EMAIL -> S#, leading to:

\[
SE (S\#, EMAIL), SP’ (EMAIL, SNAME...P#... S#),
\]

where \(I_{SE} = S\.\#, \) with the implicit recursive join SE.EMAIL = SP’.EMAIL. SE is again in BCNF. But now, SP’ is also free from any MVD, hence we do not need Fagin’s decomposition for it neither. However, SP’ isn’t (yet) in restated BCNF. Through successive restated Heath’s theorem decompositions, the final scheme for S-P would be:

\[
S’ (EMAIL, SNAME, STATUS, CITY, S#),
SE (S\#, EMAIL), P (P\#, PNAME, COLOR, WEIGHT, PCITY)
SP’ (P\#, EMAIL, QTY, S\#, SNAME, STATUS, CITY, PNAME, COLOR, WEIGHT, PCITY).
\]

Here, \(I_{SP} = S’.\#, P.\#\) with the implicit recursive joins SP’.EMAIL = S’.EMAIL and SP’.P# = P.P# define the IAs in SP’. Now, if a supplier had \(m\) email addresses on the average, \(S’\) and SP’ would have each \(m\) time more stored values on the average than, respectively, S and SP. We have more stored values than before, i.e., a sub-optimal result, as predicted.

Finally, suppose for S-P4.S that we calculate STATUS as in Example 1. The only change to S would be:

\[
S (S\#, SNAME, STATUS, CITY),
\]

where STATUS results from the \(I_s = \text{STATUS INT(SUM(QTY)/100)}\), with the implicit FROM SP_B and the implicit join S.S# = SP_B.S.#.

Notice finally, that the order of attributes in a relation mathematically does not matter. In SQL it does however. If so, one may reorder the IAs produced by the enhanced decomposition.
4. Implementing SIR DBs

4.1 Basic Processing Scheme

As said already, the most practical way towards the SIR-model enabled DBS, seems to transparently manage a SIR DB by an existing (kernel) SQL DBS. One way is to create the SIR-layer managing the SIR DB through calls to the kernel services, Figure 6. For the kernel, SIR-layer appears as any clients. SIR-layer processes every DDL or DML statement for a SIR DB through the internal generation of these for the kernel. It’s obviously useful to have the SQL syntax at the SIR-layer as compatible as possible with the kernel SQL dialect. Below, we presume the total immersion of the kernel syntax in the enhanced one.

In particular, for the Create Table R statement received, SIR-layer should determine the type of the relation to create. For R being an SR, SIR-layer pushes the statement as is down to the kernel. In turn, the processing of R with IEs, being a SIR thus by SIR-level is clearly more involved. First SIRs obviously need dedicated meta-tables for the IEs. The schemes of these are easy enough to skip details. Then, the simplest design seems to represent every SIR R in the kernel by its base R_B and its CE-view R. SIR-layer simply forwards afterwards every query as is to the kernel for execution using view R. SIR-layer avoids the complex burden related to.

Accordingly, we qualify of of basic (processing) scheme, (BPS), the SIR-layer algorithm for creating CE-views within the kernel we propose now. BPS always starts with the conversion of $I^k_R$, if there is one (and only one then, we recall), into $E^k_R$. Next, BPS passes the Create Table R_B statement to the kernel DBS, using for that all and only SAs of Create Table R. Then BPS creates the CE-view as follows. Let $A_1, \ldots, A_m$ contain every SA in R and every IA in $E^k_R$, all the attributes being in the order resulting from Create Table R and every Alter Table R. SIR layer meta-tables should maintain this order. Then, BPS simply issues to the kernel the following statement, with From and Where clauses of $E^k_R$:

\[(V1) \text{Create View R As } (A_1, \ldots, A_m \text{ From...Where...})\]

Example 6. (1) We submit to SIR-layer S-P2 scheme at Figure 3. SIR-layer finds no IEs in Create Table S and Create Table P. It passes each statement as is to the kernel that creates both relations as usual for SRs. SIR-layer in contrast determines that Create Table SP defines $I_{SP}$ that we discussed, hence applies BPS. BPS creates $E_{SP}$ and issues the following statements to the kernel DBS. The actual Create View below contains in fact the extension of $E_{SP}$, defined in IE1 formula in the motivating example.

Create View SP_B... ; /* From all and only stored attributes of SP at Figure 3.
Create View SP As (Select SP_B.*3, $E^k_{SP}$);

(2) Suppose now that DBA creates S-P3. To implement S-P3.S, BPS generates Create Table S_B as just above, then generates $E^k_S$ for STATUS and RANK as in the motivating example and finally, send the following statement, with the extension of $E^k_S$, to the kernel:

Create View S As (Select S#, SNAME, $E^k_S$, CITY);

For relation P, BPS generates $E^k_P$ and send the following statement to the kernel:

Create Table P_B...
Create View P As Select P#, PNAME, COLOR, WEIGHT, WEIGHT_KG/1000 As WEIGHT_T, WEIGHT_KG As Round (WEIGHT * 0.454), CITY From P_B;

Finally, for SP, BPS generates the same statement as for S-P2.SP, extended however with all the additional IAs in S and in SP.@

Figure 6 illustrates Example 6.2. The SIR-layer shows the SIRs as rectangles. SIRs constitute the Conceptual Scheme (CS) of S-P3, in ANSI-SPARC DB Reference Architecture. The sizes are intended to reflect the number of tuples and the number of attribute values per tuple as seen by the client, i.e.,
supposedly as in Figure 4, augmented with the IAs proper to S-P3. The lower part shows under the same convention the SRs and the CE-views. These are the Internal Schema (IS) of S-P3, together with various usual underlying physical data structures. We leave as exercise, the BPS definition of the kernel statements for the IS construct(s) for views at the SIR-layer, i.e., for External Schemes (ESs) of SIR DB, in ANSI-SPARC terms.

One may obviously tailor BPS as we defined it above to specifics of the kernel. E.g., one can take advantage of VAs when available, saving the CE-view when all IAs are VAs. We leave the analysis of such enhancements for the future.

4.2 SIR-Layer DDL & DML Statements Processing

We suppose BPS designed also for all the other DDL statements for SIRs. As for Create Table statement for a SIR, Alter Table and Drop Table also require from BPS more processing than calling their kernel counterparts only. As the result, Alter Table R adding an IE to SR R, as our example Alter adding WEIGHT_KG and WEIGHT_T to SR P did, triggers renaming of R to R_B and creation of the resulting CE-view R. For Alter concerning an SA of SIR R, BPS issues the Alter Table R_B statement. Finally, for alteration of any IAs, BPS first issues Drop View R_1 statement etc., until Drop View R statement to the kernel. Then, BPS issues every new Create View R_1...Create View R statements, reflecting the alteration. For this purpose BPS obviously explores the meta-tables at SIR-layer. We skip easy, but tedious details. Finally, for Drop Table R for SIR R, BPS issues the sequence starting with Drop Table R_B, followed by Drop View for every partial view of R, if any and for the CE-View. Notice also that future work may also tailor these statements, especially to the availability of VAs.

With respect to DML statements, once BPS created the CE-views, SIR-layer sends every SIR-layer query as is to the kernel. For every SR or view the query names, the kernel proceeds as usual. For every SIR in the query, the kernel processes its CE-view instead. For a SIR-layer update query to a SIR in particular, the kernel proceeds accordingly with the CE-view update. The kernel may however be unable to satisfy for some queries the intended semantics of the SIR update queries from Section 2.1. As widely known, view updates in popular DBSs are indeed subject to numerous limitations. Cherry on the cake, every DBS has somehow different limitations. We take therefore the simplest practical stand for the SIR-layer, i.e., that an update query to SIR R is valid iff the kernel processes it. As for view update queries at present, if a SIR update query turns invalid, the client may attempt to rephrase it. Ultimately, one may always address the rephrased update(s) directly to the SRs or to the bases of SIRs involved in the initial query.

Observe that, in our implementation, the kernel DBS always processes a query to a SIR through its CE-view. It is known for decades that the processing of the query through a view may be faster than of the (equivalent) query addressing the source SRs directly, [H1]. The rationale is a possible partial materialization of views or of join clauses through indices, [V87], [GL1], [LZ7]. In our case this may
concern the joins in CE-views, reflecting the recursive and usually outer ones that should be common to IEs, as one could see. Provided the kernel DB has thus view materialization capabilities, queries to SIRs could be accordingly both less procedural and substantially faster than their equivalents to SRs only.

We leave for the future the ambitious goal of enforcing the full semantics of SIR-layer update queries, in spite of the limitations of a specific kernel. E.g., for SQL Server a view is updatable only if it inherits from a single SR, unlike S-P2.SP thus. If SQL Server is the kernel DBS thus, the client has to rephrase every update query to this SIR to update queries to S, P or SP_.B. In contrast, MsAccess and MySQL are less restrictive: they accept update queries to views over multiple tables. However, details differ. E.g., some UPDATE SP queries to S-P2 could be valid for both kernels. Even more uniformly, none of these DBSs would let for any DELETE From S-P2.SP... unfortunately. Strangely, MsAccess would however process some of these deletes if they come through its graphical interface. Finally, an INSERT SP... could be valid for MySQL but would be always invalid for MsAccess, etc.

SIR-layer should be implemented in some host language, obviously calling the Embedded SQL interface of the kernel. This is a future work. In the meantime, [L6] backs up Example 6 with a manual simulation on MsAccess as the kernel. For each simulated SIR, a stored MsAccess table is its base. The MsAccess stored queries simulate the CE-views the BPS would create. The client may appreciate advantages of SIRs, through queries to CE-views. One may also alter and update any views, e.g., to experiment with every SIR definition, manipulation and processing aspect we discussed. As easy bonus, one may experiment the QBE interface for SIRs, generate forms, graphics, etc. In sum, one may play with all nice capabilities of MsAccess that made it so popular, almost as if they were designed for SIRs as well.

4.3 Operational Overhead of SIR-layer

The kernel storage for a SIR is in practice the one for its base. CE-view storage is negligible provided the view is not materialized, as we suppose. The storage for the kernel meta-tables for the IEs should be obviously negligible with respect to the typical one for the DB data. Next, as shown, the optimal scheme with SIRs has the same size and the same stored attributes as the optimal one for SRV-model. Hence, the storage for the values of these SIRs within the kernel is the same. The optimal DB with SIRs should cost thus negligibly more in storage than the optimal DB with the stored relations only for the same application.

For DDL statements, the processing cost of each, including of CREATE Table using BPS, is clearly negligible. For DML, since the SIR-layer passes every query as is to the kernel, its own query evaluation overhead is negligible as well. Within the kernel, the processing of every query to a SIR costs the same as the processing of the same query to the CE-view. Hence, there is no incidence on the query evaluation overhead of SIR-layer. Altogether, perhaps surprisingly, the enticing capabilities of SIRs appear thus practically almost without overhead cost.

5. Conclusion

Stored and inherited relation, (SIR), appears a useful construct for a relational DB. Through the IAs, a SIR may be conceptually richer than an SR with the same SAs, while IAs do not introduce any anomalies. SIRs alleviate in this way the notorious limitation of SRs, the dark side of the normalization. The popular ER model, proposed precisely because of this limitation, appears useless.

The operational gain of a SIR DB with respect to the SRV DB with the same SRs only is typically less-procedural SQL queries. These may be free of logical navigation or with reduced one. They can be free of selected value expressions as well. The cost for a SIR for these advantages with respect to the SR with the same SAs, hence without the latter, is the IE. With respect to the CE-view or QE-view, alternatively providing the same advantages, the IE should be always less procedural than the Create View. Especially, - if the IE is implicit. Likewise, the view maintenance is more procedural than that of
the SIR and may be source of big trouble. Recall that it is lesser procedurality of relational assertions that decisively attracted the users of equivalent navigational queries to CODASYL DBs.

Our extensions to Create Table for SIRs seamlessly integrate VAs, i.e., can be made backward compatible with the current clauses for those. Every relation with VAs is indeed a SIR with specific IAs, defined in Create Table through an implicit IE, we recall. The clients apparently found VAs useful, since they remain in use for decades now. Our extensions to SQL for SIRs only expand this capability, to a larger class of view-saving value expressions and to IAs saving the logical navigation. User should find them thus useful accordingly as well.

SIR-layer appears a higher level interface to SQL DBs. Its implementation over a popular DBS looks easy and with negligible operational overhead. The future work should start with that implementation. Depending on the kernel's capabilities, it may be wise to include enhancements to BPS we have mentioned. But even without these, the result should be a win-win deal. Better sooner than later the existing DBSs should provide for SIRs.

On the theoretical side, the design rules for SIRs based on restated NFs and Heath’s and Fagin’s theorems appear about as easy as the current ones. However, the decompositions based on these two theorems exclusively, are only the tip of the iceberg of known proposals, [D12]. Future work could restate those proposals for SIRs as well, especially the proposals for the lossless decomposition using outer joins, [JS90].

Finally, most of major DBSs are now interoperable, [LA86]. Multidatabase IEs seem thus attractive as well.

Acknowledgements

We thank Peter Scheuermann for helpful comments.

References

[C70] Codd, E., F. A Relational Model of Data for Large Shared Data Banks. CACM, 13,6,1970.
[P] Postgres SQL. https://www.postgresql.org/
