Relations with Inherited Attributes Revisited

Witold Litwin¹
(March 2016)²

Abstract. The universally applied Codd’s relational model has two main constructs: a base relation, with base attributes only and a view, only with inherited ones. In 1992, we have proposed a single generic construct with possibly both types of attributes. We termed it relation with inherited attributes (RIA). Examples showed the new capability potentially attractive. No one jumped however on the idea. We now revisit our proposal in depth. We show that the relational conceptual schemes with RIAs should be usually more accurate. The ER-model becomes rather useless. RIAs may spare to queries the customary logical navigation through join clauses. Likewise, queries may be free of complex value expressions. Both annoyances are unavoidable at present, unless hidden behind dedicated views, cumbersome then also. RIAs appear the first generally practical solution to this decades’ old dilemma. Better late than never, existing DBMSs should easily accommodate RIAs, with almost no storage and processing overhead.

1. Introduction

The universally known Codd’s (relational) model, [C69] & [C70], uses two basic constructs. Both are relations with atomic attributes only, in 1NF thus. A base relation (table) has only stored attributes (columns). A view (relation) has only the inherited ones. These are defined from base relations or view through some data manipulation language (DML) expression, e.g., an SQL one. Codd’s constructs are thus dichotomic. The rationale for this dichotomy seems unknown. We proposed in 1992 a single generic construct, called Base Relation with Inherited Attributes, RIA in short, [LKR92]. As the name hinted, RIAs were not necessarily dichotomic. They could be base tables or views. But, they could also mix both types of attributes. Examples showed this capability potentially attractive to relational DBs. The idea seemed promising also for OODBs, à la mode then.

Apparently no one followed the lead, however. Below, we revisit our proposal in depth. We call it from now on the RIA-model in short. We refer to Codd’s one as to BRV-model (Either Base Relation or View model). The former clearly includes the latter. We believe the reader familiar with the BRV-model and SQL in particular. We show implications for a relational scheme design we did not originally address. We discuss the implementation of RIAs over existing major DBMSs. We show it potentially easy and practically without storage or processing overhead. We hope the model enters the practice, finally. As one said: better late than never.

Next section details the RIA-model. We discuss the basic concepts and extensions to the current SQL dialects. Then, we show the utility of RIA-model for relational DBs. The conceptual schemes may more accurate model the reality, making the ER-model less useful. The queries become also often free from the customary logical navigation through equijoins along primary-foreign key paths. We stress that this makes RIAs the first generally practical solution to this decades’ old annoyance. Likewise, the queries may become free from complex aggregate expressions, another old exasperation. Both trouble-makes are unavoidable with the current constructs, unless hidden behind dedicated views. These are cumbersome then in turn as well, even by their sole existence.

Section 3 introduces consequently the RIA-specific schema design rules. We restate the normal forms and the Heath’s and Fagin’s theorems. Section 4 discusses an implementation of RIAs over existing DBMSs. Section 5 draws the conclusions and overviews the future work.

---

¹ Université Paris Dauphine - PSL, LAMSADE, Paris, France, witold.litwin@dauphine.fr
² Latest update April 4, 2016
2. RIA Model

2.1 Schema Definition

In the nutshell, RIA-model is the BRV-model plus relations mixing base and inherited attributes, BAs and IAs in short from now on. Thus, like every base relation or view, any RIA is a finite subset of a Cartesian product of atomic attributes (columns) over some domains. All RIAs are also, in this way in 1NF (1st Normal Form) at least. In contrast, one has to adapt the meaning of 2NF etc. to RIAs, what we do later on. Likewise, RIAs support any relational operations base or views already do. These are therefore algebraic or predicative operations and aggregate or scalar functions applying to 1NF relations. By the same token, like for any view already, any IA of an RIA results from a formula, called from now, inheritance expression (IE), we detail soon. Alternatively, a BA of an RIA has a base value only. A base value is usually stored, but some may be not. An inherited value is basically not stored. However, as known, it also might, becoming then a materialized one.

Furthermore, let R (B, V) be some RIA with B being all its BA(s), and V all its IAs. We may have B ≠ ∅ and V = ∅, i.e., no IAs. Then, we call R a base RIA (only). Alternatively, we may have B = ∅ and V ≠ ∅. Then we call R a view RIA. We qualify of full an RIA with both BAs and IAs. Context wise, we may call it simply RIA. Otherwise, when we refer simply to RIA, it may be of any kind. We neither name nor consider further the case of both sets of attributes simultaneously empty. Observe immediately that any relation possible under BRV-model is an RIA, base or view. The inverse is of course not true. Including for views, as a view may inherit now from a full RIA. The RIA-model is thus a strict super-model of the BRV-model.

Next, let R (B, V) be some full RIA. We require the primary key of the projection R[B] to be also the primary key of R. We thus always have card (R) = card R[B]. Also, any key of R[B] is also a key of R. V’s scheme may result from one or several IEs. A single IE defines all the IAs of V through a formula that could be technically of some view, say V’. V’ would be a projection of V on all its IAs, i.e., without perhaps duplicated tuples. Multiple IEs may be expected often more convenient, as we’ll show. Each IE defines then some but not IAs for R, hence for every inherited tuple it creates. Any such tuple augments some tuples already in R. It may happen that a tuple in R does not inherit any value from an IE. This is like if it inherited a null value, i.e., like if it got preserved by a half outer join. For multiple IEs, every IE defines a view, say I, with IAs being a strict subset of IAs of V. We require the sets of IAs of any two I’s being disjoint. The attributes of V should be furthermore the union of all those of I’s. Next, I’s should not present any name conflicts. All IA names they produce should be thus different, original names being perhaps renamed as usual.

Below, we define any IE as an SQL select query. As for views, this seems the most convenient at present. In particular, although each IE contributes to define R, we require any IE, identified to “its” I from now on, to refer to R in From clause of I, say as R’. The reference serves the operational specification of how each inherited tuple augments these already in R. For this purpose, we consider R’ as R with all the BAs and IAs and their values defined by B and all the IEs supposed evaluated prior to I. All R’ tuples are in this way in the Cartesian product formed by From clause of I. Let R’I denote the relation formed from all and only super-tuples of this product. Next, let I’ denote the final projection of I on attributes defined in the Select clause of I. The attributes of R’I are thus R’.* and I’.*. Then, we form R as Select R’.*, I’.* From R’ Left Join R’I. Here, it is the natural equijoin, i.e., the equijoin on all the common attributes, hence these of R’. The Select clause means further that for the join attributes that are all these of R’ besides, the final projection includes only all the tuples of R’, omitting their eventual duplicates in R’I.

To respect the condition of always having card (R) = card R[B], we finally request from every IE that the key of R’ remains the key of R. In this way, as wished, we preserve for R every tuple t’ of R’, while forming from t’, also a single tuple t in R. As wished too, to form t, we augment t’ either with the matching I tuple, also necessarily a single such tuple or a null value for each I attribute otherwise.
We call well-formed an IE or a collection of these for V if the result fulfils all the above requirements. Otherwise, the IE or the collection is invalid. We show the examples soon. We leave the exhaustive validity analysis for the future.

We consider finally that the DDL (Data Definition Language) and DML statements for RIA-model use as the kernel some favorite BRV-model SQL dialect. Full RIAs specific clauses expand the kernel. Especially, this concerns the Create Table statement that also has to define IAs. Any BAs are then defined as usual by the (kernel) dialect. The IAs use Select expression(s) of the dialect that could be in Create View under BRV-model, i.e. if all the relations referred to were base ones or views. For a full RIA, the expression however must also refer to R itself, as already discussed. Actually this reference may be sometimes implicit for convenience, as we’ll show. Next, as mentioned, an RIA may have IAs only. The expanded Create Table simply should not define then any BA. The Create View statement becomes accordingly useless in a DDL for RIAs, except for the backward compatibility. We discuss RIA specific clauses that appear for the other usual SQL DDL statements later on.

The following example illustrates all the discussed points with the biblical Codd’s Supplier-Part relational DB. It originally illustrated the Codd’s proposal. We refer below to its probably most known, slightly simplified version, often named S-P in short, popularized by C.J. Date, [D4]. The example restates S-P using RIAs. We refer to the original S-P as to S-P1. We call S-P2 its definition with RIAs.

Example 1. S-P1 models an enterprise with some Suppliers, Parts and Supplies. A supply contains some quantity of a part shipped by some supplier. Besides, some suppliers may be supplying anything currently, as well as parts may be not supplied at present. S-P1 conceptual schema consists from three base relations named S, P, SP. This scheme is optimal for the so-called relational design rules, using NFs. SP respects also the referential integrity by default. There cannot be thus a supply declared by a tuple in SP, but with the supplier or the part not yet declared in S and P. However, in practice, the referential integrity of SP would be today enforced only if defined by optional Foreign Key clauses. An application may indeed prefer not to enforce it, or only partly.

1. Figure 1, shows on this basis S-P2 scheme. This scheme is also optimal, as it will appear. The referential integrity is again the default. Figure 2 shows example extensions of S-P2 relations, i.e., a possible content of each. The schemes and the contents of S and P relations are the original ones. We use self-explaining statements to define every relation. We underline the key attributes, as usual. The S,P relations have BAs, but not IAs, i.e. V is empty. They were base relations in S-P1 that are thus now base RIAs. SP keeps the original BAs and keys as well. These form B here. SP also carries IAs. It is thus a full RIA. The choice of IAs means that the DBA considers every property of a supplier or of a part, ipso facto, of any supply. We examine the rationale behind later. The SQL Select statement in Create Table statement expresses a single IE defining all these IAs. Any IE has a name, I_SP here. IE names serve the Alter statement for an RIA that we detail later.

   Here, V consists from all and only IAs in S and P, except the key attributes S.# and P.#. V attributes are all and only created by I_SP. I_SP in particular renames the two CITY attributes. When I_SP is being evaluated, SP referred to in From clause, R’ above, supposedly contains all and only the values of its BAs, i.e., of (S.#, P.#, QTY). Here, X = R’V has all and every tuple t selected by I_SP, but augmented with the R’ values contained in the tuple of R’ x V from which t resulted. E.g. augmented with (S1, P1, 200) for v ∈ V ‘; v = (Smith, 20…) that 1st line in SP in Figure 2 inherits. The final SP in Figure 2, i.e., R above, is SP := (S#, P#, QTY) left join (S#, P#, QTY, SNAME...PCITY) AS X on SP.S# = X.S# and SP.P# = X.P# and SP.QTY [SP.S#, SP.P#, SP.QTY, SNAME...PCITY). The novelty of SP, making it a full RIA here, is that it has BAs and IAs brought by I_SP. The final BAs are all and only also in SP as R’. Also, given the current content of S, P and R’, the result has no tuples augmented with null values.

2. I-SP preserves the key of SP as R’ for the resulting SP as R, as required. Indeed both relations have (S#, P#) as the key, while S# and P# are respectively the keys of S and P. The From clause produces conceptually the Cartesian product (SP.S#, SP.P#,QTY, S.S#..., P.P#...). For every SP.S# and SP.P# value
then, the product may contain at most one tuple matching the join clauses in I_SP, i.e., SP.S# = S.S# and SP.P# = P.P#. The Where clause selects this only tuple. The final join may produce then at most one tuple from that one, given, again, that (S#, P#) is the key of SP as R’. The values of these attributes continue therefore to identify each tuple created. Hence, they remain the key of the resulting SP.

### S-P2 Scheme

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>SNAME Char,</td>
<td>PNAME Char,</td>
<td>P# Char,</td>
</tr>
<tr>
<td>STATUS Char,</td>
<td>COLOR Char,</td>
<td>QTY Int,</td>
</tr>
<tr>
<td>CITY Char;</td>
<td>WEIGHT Char,</td>
<td>I_SP (Select SNAME, STATUS, S.CITY As SCITY,</td>
</tr>
<tr>
<td></td>
<td>CITY Char;</td>
<td>PNAME, COLOR, WEIGHT, P.CITY As PCITY From S, P,SP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Where SP.S# = S.S# And SP.P# = P.P#);</td>
</tr>
</tbody>
</table>

Figure 1 S-P2 scheme.

### S-P2 Content

<table>
<thead>
<tr>
<th>Table S</th>
<th>Table P</th>
</tr>
</thead>
<tbody>
<tr>
<td>S# SNAME STATUS CITY</td>
<td>P# PNAME COLOR WEIGHT CITY</td>
</tr>
<tr>
<td>S1 Smith 20 London</td>
<td>P1 Nut Red 12 London</td>
</tr>
<tr>
<td>S2 Jones 10 Paris</td>
<td>P2 Bolt Green 17 Paris</td>
</tr>
<tr>
<td>S3 Blake 30 Paris</td>
<td>P3 Screw Blue 17 Oslo</td>
</tr>
<tr>
<td>S4 Clark 20 London</td>
<td>P4 Screw Red 14 London</td>
</tr>
<tr>
<td>S5 Adams 30 Athens</td>
<td>P5 Cam Blue 12 Paris</td>
</tr>
<tr>
<td></td>
<td>P6 Cog Red 19 London</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table SP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S# P# QTY</td>
<td>SNAME STATUS SCITY</td>
</tr>
<tr>
<td>S1 P1 300 Smith 20 London</td>
<td>P1 Nut Red 12 London</td>
</tr>
<tr>
<td>S1 P2 200 Smith 20 London</td>
<td>P1 Nut Red 12 London</td>
</tr>
<tr>
<td></td>
<td>P2 Bolt Green 17 Paris</td>
</tr>
<tr>
<td></td>
<td>P3 Screw Blue 17 Oslo</td>
</tr>
<tr>
<td></td>
<td>P4 Screw Red 14 London</td>
</tr>
<tr>
<td></td>
<td>P5 Cam Blue 12 Paris</td>
</tr>
<tr>
<td></td>
<td>P6 Cog Red 19 London</td>
</tr>
<tr>
<td>S2 P1 300 Jones 10 Paris</td>
<td>P1 Nut Red 12 London</td>
</tr>
<tr>
<td>S2 P2 400 Jones 10 Paris</td>
<td>P1 Nut Red 12 London</td>
</tr>
<tr>
<td></td>
<td>P2 Bolt Green 17 Paris</td>
</tr>
<tr>
<td></td>
<td>P3 Screw Blue 17 Oslo</td>
</tr>
<tr>
<td></td>
<td>P4 Screw Red 14 London</td>
</tr>
<tr>
<td></td>
<td>P5 Cam Blue 12 Paris</td>
</tr>
<tr>
<td></td>
<td>P6 Cog Red 19 London</td>
</tr>
</tbody>
</table>

Figure 2 S-P2 content. IAs are in Italics.

For instance, for the SP tuple with key (S1, P1) in Figure 2 both values being here BAs, the product has only one the tuple (S1, P1, 300, S1, Smith, 20, London, P1, Nut, Red, 12, London). The join clauses select this one and only this one. The projection removes the duplication of S1, P1 forming a tuple of V’. The final join produces the SP tuple (S1,P1,300..London). The first three values are the base ones, all the others are inherited. This is the 1st SP tuple in Figure 2. The similar evaluation produces all the other tuples there. If there were in addition some base tuples in SP not matching on join clauses any tuple in the product, these would be concatenated with the adequate null values. Finally, I_SP reveals a well-formed IE.
3. Suppose now that the referential integrity is not desired for SP. In addition to the S-P2 content at Figure 2, one could wish in SP the tuple showing, e.g., that supplier S7 we have the supplies part P1 in quantity 200, without yet S# = S7 in S. The insert of the BAs into SP, would make I_SP to produce the tuple (S7, P1, 200, null as SNAME...null as PNAME...null as P.CITY). This, despite the tuple with P1 in SP. One may find this result an undesirable side-effect, as well as vice versa for a hypothetic part P7 supplied by S1. If so, the following IE could do:

I_SP1 (Select SNAME, STATUS, C.SITY, PNAME, COLOR, WEIGHT, P.CITY From (SP Left Join S On SP.S# = S.S#) Left Join P On SP.P# = P.P#)

Here, the 1st left join, on SP and S, produces the relation, say X, inheriting all the values in S when join attributes SP.S# and S.S# match, but also preserving all the SP base tuples, if any, without matching S.S#. The follow up left join preserves now all the tuples of X, but expands each of them either with the tuple of P with the matching P.P# value or with nulls. Hence we may have in final SP values inherited only from S, or only from P, or both. Notice that we could commute the left joins.

4. Consider that we have in fact in SP two sources of inheritance: S and P. An IE per source may seem more appropriate, perhaps also simpler, than a single IE for both. Surely, the result is easier to imagine in customary graphical form where IEs would be arcs. The following two IEs could do:

I_S (Select SNAME, STATUS, C.SITY C.SITY As SCITY from S, SP WHERE SP.S# = S.S#) and I_P (Select PNAME, COLOR, WEIGHT, P.CITY As PCITY from P, SP WHERE SP.P# = P.P#).

Each IE is less procedural than I_SP and I_SP1, especially. At least, since each is shorter. It is also likely from the example that most users should rather prefer more, but simple IEs than less, but complex ones. Observe that I_SP and I_SP1 commute. Hence, one may evaluate them in any order. Also, easy to see that each is well-formed as well as the set they form. We leave as easy exercise to find out the single equivalent IE.

5. The STATUS attribute of S is a base one, simply an integer. Imagine however that behind, it is in fact somehow calculated, e.g., as the total quantity delivered by a supplier divided by hundred and rounded. If S is an RIA, one may rather define STATUS as an IA, through the following IE. We allow that one, as any IE creating a single attribute, to bear implicitly the attribute name. Here, the implicit name would be STATUS.

(Select Int(SUM(QTY)/100) AS STATUS FROM S, SP GROUP BY S# WHERE S.S# = SP.S#).

Under BRV-model, S can only be a base table. Hence, the stored value would need to be updated from time to time, being possibly inaccurate in the meantime. All this does not sound practical. Rather, STATUS should not be then in S, but should be dynamically calculated in a query or should “emigrate” to a dedicated view. The simplest one could be:

Create View Status (S#, Int(SUM(QTY)/100) AS STATUS FROM SP GROUP BY S# ).

However, the immediate practical advantage of S as the RIA is that the simplest query Select * from S would continue to deliver STATUS. With the Status view, the join clause on S and Status would be necessary. Alternatively, Status should be more complex, inheriting not only S# and QTY, but also all the other attributes of S and including the same join clause as our IE. Each solution brings some havoc with respect to single full RIA S. RIAs clearly alleviate here a fundamental limitation of the “traditional” relational model. We come back to this important issue in the next section.

6. Consider as conceptual property of every supplier, wished an attribute of S so, the list of the supplies it provides, with P#, PNAME and QTY for each, in descending order on QTY. The following IE in S scheme defines the related inheritance from SP. The LIST aggregate function casts for each supplier all the selected tuples into a single Char type value (a string thus), [L3].

I_Supplies (Select LIST (P#, PNAME, QTY) From S, SP where S.# = SP.S# Group By SP.S# Order By Qty Dsc, PNAME Asc)
The result for S1 would be S tuple:
(S1, Smith...London, (P3, Screw, 400 ; P1, Nut, 300 ; P6, Cog, 100)).

Notice that without the aggregation, I_Supplies would be invalid. It would possibly inherit more than one tuple for some S# values, e.g., for S1. Also, it would not let S# to remain the key of S, hence of its base part neither.

As mentioned for S-P1, and following the BRV-model, we suppose also for RIAs that optional Foreign Key clauses may accompany an RIA Create Table specifying the referential integrity details. Also, analogously to views in BRV-model, IEs basically do not cascade upward the updates to IAs. They might however, with additional usual dialect-dependent clauses. Within the limits already heavily studied for view updates.

The other usual SQL DDL statements are, we recall, Alter Table, Drop Table and Create Index. With respect to the first statement for an RIA, we suppose that it “inherits” from its kernel dialect the clauses Add, Alter or Drop, operating on base attributes, as well as all the related capabilities. The specifically for RIAs supposed extension is then that the Alter clause applies also to the IEs. That alteration consists simply of a new expression. Next, the Drop Table simply drops as usual the definition and the eventual content of an RIA. The operation should not of course violate the referential integrity. It might thus be required as usual to cascade to other RIAs or get aborted if a violation would otherwise result. Finally, Create Index statement for an RIA is the same, except that it naturally concerns the base or materialized attributes only.

Example 2.

1. Suppose WEIGHT expressed implicitly in pounds. Alter P by appending IA WEIGHT_KG converting it to kilograms and IA WEIGHT_T converting it further to tons.

   Alter Table P Add (Select WEIGHT / 2.1 As WEIGHT_KG From P X, P Where X.WEIGHT = P.WEIGHT), (Select WEIGHT_KG / 1000 As WEIGHT_T From P X, P Where X.WEIGHT_KG = P.WEIGHT_KG);

   The example defines RIAs that inherit from themselves. We’ll qualify these naturally of self-inheriting. Each IE again implicitly bears the name of the attribute it creates. Notice that, this time, they are to be evaluated in order. In fact, we used two IEs to illustrate this case. A single IE say I_W could indeed also do:

   Alter Table P Add I_W (Select WEIGHT / 2.1 As WEIGHT_KG, WEIGHT_KG / 1000 As WEIGHT_T From P X, P Where X.WEIGHT_KG = P.WEIGHT_KG);

   Main SQL dialects would in fact allow defining both attributes as so called virtual or computed etc. attributes. We come back to this practice in next section. For compatibility, the following simpler syntax could do for an IE in a self-referencing RIA:

   Alter Table P Add I_W (WEIGHT / 2.1 As WEIGHT_KG, WEIGHT_KG / 1000 As WEIGHT_T);

   For the same reason, the IE(s) could be the other way around: ... WEIGHT_KG WEIGHT / 2.1, WEIGHT_T AS WEIGHT_KG / 1000 ...

2. We alter S by replacing STATUS with the inherited one.

   Alter Table S Drop STATUS Add (Select Int(SUM(QTY)/100) As STATUS FROM SP, S GROUP BY S# WHERE S.S# = SP.S#);

3. We wish SP to implicitly inherit also eventual alterations of S or P, adding or dropping attributes there. I_SP does not do it. We suppose therefore that the usual SQL (Klein’s) operator ‘*’ in the dialect used, supports also the syntax */A or */A1,...,An. Here A designates the attribute(s) that ‘*’ does not generate in its Select list. We alter the SP scheme in Figure 1 as follows.
Alter Table SP Drop I-SP Add I_SP_ALL (Select */S.S#, P.P# From S, P,SP Where SP.S# = S.S# And SP.P# = P.P#);

2.2 Data Manipulation

Any RIA is a 1NF relation, by definition. The relational algebra operators of BRV-model operate on 1NF relations. Whether an attribute involved is a BA or an IA is immaterial to these operators. They apply therefore to RIAs mixing BAs and IAs as well. Hence, one may project, select or join such RIAs as well. 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...

For a modification of an RIA, i.e., the SQL Insert, Update and Delete statements, an Insert proceeds as usual for any base attribute. It might insert a value of an IA as well, provided the update propagates to the source BAs. The insert may fail thus, e.g., if an IA is inherited through a value expression or a scalar or aggregate function. Same occurs for Update and Delete statements. The latter deletes as usual physically from the DB all the selected values of the BAs. It also deletes from the RIA the values of all the selected IAs, although conceptually only, of course. As for BRV-model, a modification may cascade upward or downwards along referential integrity paths. A modification of an IA may lead then to a physical deletion elsewhere as well.

Example 3. The simplest for SP SQL Select statement Select * From SP would show all the SP values, of all BAs and of all IAs in Figure 2. The Insert statement of the MS Access SQL dialect, Insert SP (select ‘S4’ as S#, P4 as P#, 100 as QTY); would add the tuple with these base values and, therefore, with all those inherited through I_SP or I_S and I_P etc. The statement Update SP set QTY = 250 where S# = ‘S1’ and P# = ‘P1’; should normally succeed, updating one base value. However, the statement: Update SP set QTY = 250, CITY = ‘Paris’ where S# = ‘S1’ and P# = ‘P1’; may succeed iff a change to CITY value propagates to S. To authorize it requires some thinking. The side-effect that the city would change then also for any other supply by S1 may indeed surprise. Next, an update of STATUS as BA in S may succeed and as an IA in SP as well, provided it propagates upward to S. But if STATUS the IA STATUS above defined, any update to must fail. Finally, the statement Delete SP Where S# = ‘S1’; would erase as usual physically from the DB all the values of the base attributes in the selected tuples in Figure 2. Conceptually, it would also delete then all their inherited ones.

2.3 Utility of RIAs

RIA-model enhances the usability of the relational model. It does it through useful capabilities of full RIAs, while keeping all those of the BRV-model. In the nut-shell, a full RIA adds to a base table the capabilities provided otherwise only by specific views. One avoids then their creation and management hassle and perhaps the navigation through them and the base table. Examples above already hinted to such capabilities. More generally, on the theoretical side, they may avoid the conceptual modelling pitfalls of the BRV-model, well-known since the relational model was proposed, but without a satisfactory solution. Full RIAs may also naturally avoid the usual logical navigation in queries along the primary-foreign key logical access paths, [MUV84]. Full RIAs may finally avoid complex value expressions in queries.

Both the logical access paths and the expressions are often necessary in queries at present, unless hidden behind dedicated views. Managing views is however also a hassle, including logical access path to these in turn. This decades’ old dilemma was basically the proverbial one, of choosing between Scylla and Charybdis, [M4]. RIAs appear the first generally practical solution to it, i.e., avoiding to queries both the logical navigation and complex expressions, without additional dedicated views. The result should be useful. RIAs put indeed under a single umbrella and expand capabilities of two already popular practices. One, called mainly virtual attributes, amounts to self-referencing RIAs with simple arithmetic value expressions. The other, sometimes qualified of implicit joins, amounts to the logical navigation sparing capabilities of RIAs.
To justify our claims, we focus on S-P1. Countless actual DBs using nevertheless S-P1 as template, the benefits we show generalize accordingly. We also mentioned, the BRV-model is a strict sub-model of the RIA-model. The trivial condition to stay with the current model only, while using a hypothetical RIA-model enabled DBMS, is simply to refrain from full RIAs. Switching to the RIA-model is a safe move thus. No loss of any current capabilities of a relational DB may result from.

Example 4. Figure 1 and Figure 2 illustrate that if a DBMS uses the RIA-model and for any reasons, we wish S-P1 DB only, it suffices to generate the scheme of S-P2 without I_E_SP. Then, any queries to S-P1 under RIA-model in this way, amount to these possible under BRV-model only.

We now justify, one after the other, our other claims.

**Conceptual modelling**

For the conceptual modelling, the known basic goal for a relational DB is to possibly (1) model in the DB scheme all the properties of the “real” object wished for the scheme, (2) have them all possibly the same base relation scheme. That is why we use n-ary relations and not, e.g., only the once popular only binary ones [A74]. A base relation is not intended however for the calculated values. Those are supposed in views or dynamically calculated in queries. A view in turn cannot have base attributes. The BRV-model cannot thus fulfill the goal for objects with properties of both types.

Example 5. In S-P1, if STATUS is no more a base attribute, it cannot be in the base relation S scheme anymore. A query or some view scheme, say Status, must provide its values it instead. No such constraint for S in S-P2, benefiting from the IE STATUS above.

Another facet of this trouble concerns the SP relation in S-P1. It is useful to remind that the relational model illustrated with S-P1 was an instant hit. Most folks attracted to, had no problems with acceptance of S and P schemes. One can indeed easily imagine practical use of these data, despite the inherent simplicity of 1NF. In contrast, many could not swallow SP scheme. It’s hard indeed to imagine an actual supply characterized by the three attributes there only. Names at least seem a practical must.

Some could think that instead, the obvious way-out was to simply add all the other attributes of S and P as base ones to SP. Codd & al have pointed out however very early, as widely known, that it would be a bad idea. The **anomalies** would follow, by violations of the NFs. We recall that a possibly important storage overhead could appear. Next, the same value could need many re-inserts, with evident risk of error then in addition. SP as is seemed after all preferable, as widely known. Its conceptual modelling insufficiencies could be compensated somehow then again by additional view(s), the full (universal) one of S, P and of SP in S-P1 in particular. But this constituted then also another facet of the discussed limitation of the BRV-model. Once more, SP as RIA in S-P2 clearly avoids the trouble.

Actually it’s worth recalling that a heated debate followed the SP scheme proposal. Leading to the Entity-Relationship model, (ER), especially, [C76]. The ER diagram was proposed as the actual conceptual scheme. This one was then only modelled further by the relation schemes. For S-P, S & P relations were then modelling entities. SP was modelling in contrast “only” the relationship between these. The three attribute could suffice then. However, to convince real-life folks that the box of parts they face is “only” a relationship seemed less obvious. Likewise, the question whether a marriage is an entity or a relationship never got a unanimous answer. Again, using RIAs, avoids to S-P2 such esoteric troubles. No need for the ER-model neither.

Our next claim has two reasons, occurring in probably any actual DB at present. First, the well-known relational design principles that we also recall in next section, often lead a query to address several relations. The mandatory logical navigation through inter-relational join clauses, sometimes called natural, results from. This annoys most users, as known for decades, [MUV84]. Likewise, a query may need results of a complex value expression. Possibly with aggregate functions, GROUP BY, subqueries in Select list or in the Where clause etc. Many users have hard time with or are simply
unable to formulate complex queries. The well-known and only possible fix to both issues at present consists of additional views, shielding the logical navigation and complex calculus. Managing additional schemes is however another hassle. Full RIAs avoid both facets of the dilemma.

Example 6. Some S-P1 users could clearly find the value expression for the STATUS too complex for their taste. Under BRV-model, DBMA must provide a view hiding that one. The full RIA S in SP-2 avoids this trouble. Next, consider a stereotype request, say Q, to either DB, selecting in every supply, the IDs and names of the supplier and part involved, together with the quantity supplied. Let query Q₁ express then Q for S-P1 and query Q₂ for S-P2. These could be basically as follows:

(Q₁) Select S#, SNAME, P#, PNAME, QTY from S,P,SP Where S.S# = SP.S# and P.P# = SP.P#;
(Q₂) Select S#, SNAME, P#, PNAME, QTY from SP;

The joins in Q₁ are unavoidable for any equivalent query to S-P1. They are due to the necessity of the logical navigation through S, P and SP. In contrast, neither Q₂ as one sees, nor its equivalencies need the navigation, hence all spare the join clauses to the user.@

Complex Value Expressions

Our next claim was that full RIAs generalize in fact some popular practices already beyond BRV-model, as originally defined. One is a view-saver usually called computed, dynamic or virtual attributes or columns. The concept appeared in 80ties. Major DBMSs, Sybase first, picked it up rapidly and still use it, without, regretfully perhaps, however of some research results, [LV86]. Virtual attributes are not stored, but calculated through value expressions, from other attributes in the same base relation or view. The relation may have then both base and inherited attributes. It is thus an RIA. More precisely, it is a self-referencing one, further limited basically to simple arithmetic value expressions only.

The IAs WEIGTH_KG and WEIGHT_T in Example 2 define virtual attributes. Self-referencing RIAs are thus in fact already widely applied. Full RIAs obviously provide for more complex calculus capabilities, by far, through the inheritance from multiple relations. Examples proved these helpful at least for S-P. That DB is the template for countless actual ones, as widely known.

The virtual attributes are an add-on to BRV-model since they create full RIAs in fact. Strict observance of that model would require instead the dedicating view with such attributes. Such views would be always computationally sufficient. The concept is thus only a view-saver, presumed to enhance the usability. The conjecture appears true. As said, major DBMSs propose the concept for decades now. More general than self-referencing only RIAs should accordingly help usability of numerous actual DBs as well.

Logical Navigation

With respect to the logical navigation saver facet of RIAs there were already several proposals claiming this property. One group was using the universal relation idea, we recall in next section. An alternative principle was the implicit joins, sometimes called now also automatic, [L85], [LSW91]. None of universal relation based proposals, despite strong excitement they have created in research milieu, [M04], made it to popular DBMSs. The implicit joins did, e.g., to SQL Server & MsAccess. The industrial versions are again limited with respect to research results. The MsAccess version seems the most extensive up to 2016. Strange enough, at least in the two MS systems, implicit joins are only in the QBE-base visual interface, providing for the data definition and manipulation. They make easier graphical QBE queries, translated then to SQL, with the joins made explicit. In the graphical form, the implicit joins are represented as directed or undirected arcs, popping up in the query graph, once the relations, represented as the graph nodes, are selected. Alternatively, through the definition of so-called sub-tables, the implicit joins help 4GL forms, so called data sheets especially. We’ll recall these terms soon.
The query arcs are derived from directed or undirected arcs, called ambiguously relations between tables in a specific diagram of the DB scheme and views, termed Relationships. The arcs are optionally dragged between the diagram nodes that are boxes representing the actual relations, called then tables. These may be base tables or views. One may declare the referential integrity when appropriate and the type of join to be implicit in queries. This can be an inner equijoin (default) or a half outer-join, translated to left or right in SQL. Alternatively, the joins may be tried out automatically from the query, provided the DBA authorizes this option, called accordingly. The join attributes must share the name then and one must be a primary key. The automatic join is also always only an (inner) equijoin. The attributes involved may be composite. The resulting query can however be easily strange then, at best. The reasons for are clear apparently only for Microsoft, perhaps.

If an arc primary-foreign key exists between two tables, then the table with the foreign one may also automatically become a sub-table, we just spoke about. One can also declare a sub-table more generally, manually among the so-called properties of its super-table. The sub-table is chosen by name and by declaration of an arbitrary atomic attributes per table as implicit join attributes, to select sub-tuple(s) of each super-tuple. Assuming the super-table at the left, the semantic is the implicit join is that of the left equijoin. In this way, e.g., one may declare S a sub-table of SP. MsAccess then automatically chooses SP.S# and S.S# for implicit left join. For unknown reasons, a table may have only one sub-table. If there are several arcs, as it would be for SP, and no manual declaration, one of the arcs is mysteriously preferred. Creation of sub-tables does not avoid the logical navigation in ad-hoc queries. It only let the sub-table tuples to be visible either in as a sub-form of the 4GL form of the super-table, or in the specific view of the super-table, called data sheet view, we mentioned. In the data sheet of SP, for instance, there would be one line for every supply. Right under each such line, one could also see through the implicit join, an on-demand line with all the data of the supplier in S.

The declarations of sub-tables and the arcs of the relationship diagram, avoid the logical navigation. They do it without using some preexisting view of all these tables that would be the only way toward the goal under BRV-model, we recall. The implicit joins they generate act as thus as view-savers. The practice is popular with major DBMSs already for decades, despite its limitations. The RIA-model aims at similar capabilities, but, as for virtual attributes, potentially, beyond the current limitations. E.g. through its IEs, SP can be trivially dealt with as having two sub-tables, S and P. Likewise these IEs avoid the join clauses between S and SP or SP and P, as the implicit joins generated by the arcs also do etc. But, in addition, other discussed IEs may offer the view-saving complex values expressions avoidance capabilities we discussed. The implicit joins do not provide these. Hence, the RIA-model usefully generalizes also this popular practice. Providing a single umbrella for both tools, as our claim stated.

3. RIA-model Schema Design

The relational scheme design rules have been formulated for the BRV-model only. They have to be thus revisited for RIAs with BAs and IAs. We now address this issue, continuing with S-P2 as the motivating example. We first revisit the NFs. Next we restate the Heath’s and Fagin’s theorems. We show that RIAs mixing BAs and IAs effectively benefit only the decompositions according to the former.

3.1 Normal Forms

The basic design rule for a relational DB scheme under BRV-model is the respect of the normal forms (NFs). We recall that these are 1NF, 2NF, 3NF, BCNF, 4NF, 5NF. Any relation in 5NF is furthermore in 4NF, the one in 4NF must be in BCNF etc. Every relation under BRV-model is by default in 1NF we also recall. Next, relations in 4NF that would not be in 5NF are very rare, what makes BCNF and 4NF the most useful NFs in practice. E.g., SP (S#, P#, QTY) is in S-P1 is in BCNF, while if SP’ (S#, SNAME, P#, QTY) with base 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. Contradicting S then, where SNAME is already, anyhow. Using SP instead, avoids the trouble.

To fully transpose the concept of NFs to RIA-model, one has to extent the current definitions of 2NF etc. also to RIAs with both BAs and IAs. These are already always in 1NF by default we recall. Notice then that the above anomalies of SP’ would not concern a view SP’. We therefore state that an RIA \( R(B, V) \) is in \( \mathcal{N}F \) or BCNF, iff \( R[B] \) is in \( \mathcal{N}F \) or BCNF.

Example 7. 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 base attributes conforms to the respective usual NFs. Same, trivially, for S and P in S‐P2. However, as mentioned, the base relation SP’ \([S#, SNAME, P#, QTY]\) would not be in BCNF. But, an RIA SP’ with SNAME inherited would. Hence, if, for any reasons, SNAME or any other IA in SP was rather made a base attribute, SP would not be in BCNF anymore.

3.2 Schema Design

We recall that at present, i.e. for a BRV-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 may present 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 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. Ideally, we start with the attempt of a single universal base relation, say U, for the entire DB. U avoids the logical navigation entirely, as all the attributes are in. Unfortunately, chances for U in 4NF are zilch in practice. We usually perform then a decomposition of U into projections, i.e. we suppose that the DB consists of these projections instead. The decomposition must be lossless, producing thus 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 again any such projections. We continue, until every projection is anomaly-free.

As know, the two basic decomposition theorems are Heath’s and Fagin’s ones. The former may help with annoying FDs. The latter removes MVDs. 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, i.e., leading to more base values, may result. Both theorems decompose a relation into two projections. Hence 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. Nevertheless, several lossless decompositions of a relation through these theorems usually exist. Then, 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 RIA-model, i.e., U and the projections are basic RIAs. Such schemes were out of scope of the original methodology, of course. We suppose consequently that even U may contain IAs, e.g. the aggregate ones we showed. For FDs and MVDs used for the decompositions, we assimilate all these IAs nevertheless to BAs. We naturally apply to the projections the restated NFs. These in contrast, consider any IA as is. We’ll now also restate for RIAs the Heath’s and Fagin’s theorems. The goal is that for any decomposition of an RIA R, one of resulting RIA preserves the attribute cover of R, i.e., has all and only attributes of R. The result aimed on is that the scheme with the projections is possibly as free of the logical navigation as it was with R. This will appear possible only through the use of full RIAs. We leave for the future the possible restatements.
of others of many known rules, intended to help with even better schemes, e.g., perhaps the Rissanen’s theorem.

The major gain that will appear is that, for the same size optimal schemes for a DB, the one with RIAs is largely free of the logical navigation, unavoidable otherwise. More precisely, as we’ll show the optimal RIA scheme will be always as follows:

(a) In the absence of MVDs, no decomposition introduces the logical navigation.
(b) Otherwise, a decomposition removing an MVD may still avoid the logical navigation to some queries to the projections, but not to all such queries.
(c) For a real-life DB, we may reasonably expect most or even all queries the discussed logical navigation free.

Indeed, first, the Heath’s theorem states, we recall, that for any base relation ABC (A, B, C) and an FD A \(\rightarrow\) 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, we may have several choices for A, B and C. As every decompositions doubles A, if we have choice we tend to choose A wisely with fewest attributes. Possibly, we choose A also the primary key of AB, in 3NF at least then. Also wisely, for reasons already invoked, we hunt for the largest possible B. We restate the theorem for RIAs, to the decomposition into AB (A, B) and ABC (A, B, C), where ABC.B is an IA defined by the IE: (select B from AB where AB.B = ABC.B). This decomposition is also into two schemes and clearly lossless. But, while AC was a base table, ABC is a full RIA. This decomposition is thus possible only for the RIA model. Unlike the original one, it preserves the attributes A, B, C together, in resulting ABC. It avoids thus, as promised, the logical navigation for queries to B and C.

Next, the Fagin’s theorem also states that in presence of MVD A \(\rightarrow\rightarrow\) B | C in the presumably base relation ABC (A, B, C), its decomposition into AB (A, B) and AC (A, C) is lossless. Let us now denote as B’ a (perhaps empty) subset of B and as C’ a (perhaps empty) subset of C such that A \(\rightarrow\) B’ and A \(\rightarrow\) 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: the decomposition creates AB (A, B, C’) and AC (A, B’, C) where the IE (select C’ from AC where AB.A = AC.A) defines C’ and the IE (select C’ from AC where AB.A = AC.A) defines B’. The result avoids thus the navigation for any query to B and C’ or to B’ and C in the projections. Only the queries to B/B’ and C/C’ still need it. We thus do not avoid completely the logical navigation that the decomposition creates. But we limit it to fewer queries.

On these bases, the generic schema generation algorithm for RIAs is then quite analogous to that for the base 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 again should work first. We naturally end up with the same size scheme, but also with lesser need for the logical navigation, as claim (b) states. If there are no MVDs, we remove the discussed logical navigation need entirely, as claim (a) states. Finally, the rationale for claim (c) is that in a real-life DB, MVDs are at most rare with respect to annoying FDs. Also, B’ or C’ should usually have several attributes, unlike B/B’ or C/C’. Even for a decomposed MVD, most queries to the projections should then normally be navigation free as well.

The following example illustrates all the debated points.

Example 8. The biblical S-P scheme results from Heath’s theorem only. Similar schemes are countless in practice, as widely known. Our scheme in Example 1 would need then the restated Heath’s theorem only as well. To illustrates the restated Fagin’s one, we modernize S-P a little. We call the result S-P3. We suppose that any supplier there may have one or several email addresses for contact about any of its supplies. Each address is the value of new base attribute EMAIL. Needless to stress that email values are not shared. Every address belongs to one supplier only. We redesign the S-P scheme under RIA-model accordingly, as follows.
We start optimistically with the universal relation $U$, [MUV84]. In short notation we have:

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

Notice the necessarily different names for the supplier and part cities, unlike in $S$ and $P$ of $S$-$P1$ or $S$-$P2$. $U$ is possibly the optimal base relation for $S$-$P3$, unless proven otherwise. What’s easy, since $\text{EMAIL}$ already introduces the MVD: $\text{S#} \rightarrow\rightarrow \text{EMAIL}$ | $(\text{SNAME, CITY, STATUS, P#...QTY})$. $\text{SPE}$ is not in 4NF thus. Regrettfully, the optimal $S$-$P$ scheme cannot thus be $\text{SPE}$ (only). We have to decompose it. We have MVDs and obviously FDs. We start as above indicated with the restated Fagin’s theorem. The decomposition creates two relations:

$$SE(\text{S#}, \text{EMAIL}, (\text{select SNAME, SCITY As CITY, STATUS from SP, SE Where SE.S# = SP.S#})), SP(\text{S#}, \text{SNAME, SCITY, STATUS, P#...QTY}).$$

$SE$ is now a full RIA, while it would be a base relation (and RIA) only for the original Fagin’s decomposition. We have $C' = (\text{SNAME, CITY, STATUS})$ and $B' = \emptyset$. $SE$ is in the (restated) BCNF. It would not be if any of its IAs, e.g., $\text{SNAME}$, was a base attribute. The IAs of $SE$ make many queries navigation free. Otherwise, these queries would need to navigate over $SE$ and $SP$. For instance, the query that one may expect frequent, selecting every email of a given supplier. In contrast queries selecting emails and an attribute in $SP$ that was not inherited in $SE$ would still need to navigate, i.e. would require the $SE$ join $SP$ clause. Such queries, e.g., all emails and all names of parts supplied by a supplier, seem nevertheless here clearly of by far lesser practical interest than those to $SE$, saved from the navigation, like the cited one.

$SE$ has no more MVDs, hence 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 decomposition:

$$S(\text{S#}, \text{SNAME, CITY, STATUS}), SP(\text{S#}, \text{P#}, \text{PNAME...CITY, QTY}, (\text{Select}/S\# \text{ From } S \text{ Where } S.S# = SP.S#)).$$

In the projections, we could by the way more conveniently rename $\text{PCITY}$ and $\text{SCITY}$ to simply $\text{CITY}$. The projection $SP$ is again a full RIA, with the same attributes as the decomposed one. Some became now however inherited from $S$. Notice that this does not change anything for $SE$ scheme. $S$ is a base RIA and in BCNF, whether thus restated or not. $SP$ however still isn’t. Its projection on the base attributes indeed isn’t in BRV-model, given the FD : $\text{P#} \rightarrow \text{PNAME, COLOR, WEIGHT, PCITY}$. We thus apply the restated Heath’s theorem again to $SP$. On the $SP$ gets decomposed to:

$$P(\text{P#}, \text{PNAME, COLOR, WEIGHT, CITY}) \text{ and } SP(\text{P#, S#, QTY}, (\text{Select}/S\# \text{ From } S \text{ Where } S.S# = SP.S#), (\text{Select} */P\# \text{ From } P \text{ SP Where } P.P# = SP.P#)).$$

Now $S$-$P3$ has every RIAs in BCNF, hence 4NF as there are no more an MVD. The optimal scheme is as follows. We underlined the primary key base attributes.

$$S(\text{S#}, \text{SNAME, CITY, STATUS}),$$

$$P(\text{P#, PNAME, COLOR, WEIGHT, CITY}),$$

$$SE(\text{S#, EMAIL}, (\text{select SNAME, SCITY As CITY, STATUS from SP, SE Where SE.S# = SP.S#})),$$

$$SP(\text{P#, S#, QTY}, (\text{Select}/S\#.S# *From } S \text{ SP Where } S.S# = SP.S#), (\text{Select} */P.P# \text{ From } P \text{ SP Where } P.P# = SP.P#)).$$

Notice that $SP$ scheme is that of $S$-$P2$ from Example 1. That is why the $S$-$P2$ scheme is the optimal one as well. 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 one we spoke about. Indeed, the first decomposition of $\text{SPE}$ could use the FD : $\text{EMAIL} \rightarrow \text{S#}$, leading to:

$$SE(\text{S#, EMAIL}), SP(\text{S#, SNAME... (Select EMAIL From SE Where SE.S# = SP.S#})$$
SE is again in BCNF, SP is also free from any MVD, but isn’t (yet) in BCNF. Through successive decompositions of Heath’s theorem, the final scheme for S-P would be:

\[
S \left( S\#, \text{SNAME}, \text{CITY}, \text{STATUS} \right) \quad \text{SE} \left( S\#, \text{EMAIL} \right) \quad P \left( P\#, \text{PNAME}, \text{COLOR}, \text{WEIGHT} \right)
\]

Now, if a supplier had \( m \) email addresses on the average, \( \text{SP’} \) would have \( m \) time more base values than \( \text{SP} \). Clearly, we got a sub-optimal result.

Finally, suppose \( \text{STATUS} \) calculated in Example 1. The only change would be the IE defining it in \( S \). I.e., we would have:

\[
S \left( S\#, \text{SNAME}, \text{CITY}, \left( \text{Select} \sum \text{QTY} \text{As STATUS From SP Group By S\# Where S.S\# = SP.S\#} \right) \right)
\]

If we had also an IA, say \( \text{STATUS1} \) in \( S \), defined as the number of parts supplied, the following IE would trivially define both \( \text{STATUS} \) and \( \text{STATUS1} \):

\[
\left( \text{Select} \sum \text{QTY As STATUS, COUNT (*) As STATUS1 From SP Group By S\# Where S.S\# = SP.S\#} \right).
\]

### 4. Implementing RIAs

The obvious basic way for creating an RIA-enabled DBMS, is to have RIAs transparently managed by an existing DBMS with some kernel SQL dialect. We spoke about abundantly. We then should implement a layer above the DBMS, say RIA-layer, parsing the RIA definition and manipulation statements. Those should extend to RIAs the dialect statements for base tables, views and queries, as already discussed. To operationally process the statements for RIAs, the RIA level should uses the service of these dialect statements only.

The RIA-layer processing a statement managing a base or view RIA, obviously may pass it down as is or almost. Any query to such RIAs only pass for execution as is. In contrast the Create Table for such an RIA, requires an easy analysis first, to generate either the kernel Create Table or Create View statement. Similar need occurs, e.g., for an Alter statement.

Creation of a full RIA, say \( R(B, V) \), with \( V \) resulting from some IEs is clearly more involved. Since \( R \) is neither a base relation nor a view no actual DBMS can execute its Create Table as is. We must represent \( R \) for the DBMS as \( B \) and view(s) somehow taking care of all IEs. To evaluate the queries to \( R \) by the DBMS as well, the easiest seems to have at least the full view of \( R \) there, i.e. that would be defined as select * from \( R \) if \( R \) was a base table or view managed by the DBMS. As this is impossible under the DBMS, one must construct an equivalent view from \( B \) and the view(s) we just spoke about. This is (fortunately) always possible through the following recursive processing. We define it using pseudo SQL notation.

Let \( I_1...I_n \) be the IEs defining \( R \), to be evaluated in order. Let \( K \) be the key of \( B \). For each \( I_j \), let \( I_j^F \) be the view formed from \( I_j \) by adding \( K \) to all the attributes in Select clause of \( I_j \) with values being these of \( K \) in each tuple selected by \( I_j \) within the Cartesian defined by its From clause. Let it be \( R_0 = B \) and let \( R_1...R_n \) be the views produced by successive evaluations of \( I_1, I_2...I_n \) starting with \( B \). Then \( R_n \) is the full view of \( R \), resulting from the following recursive formula. For \( j = 1...n \), one should evaluate:

\[
R_j = \text{select } R_{j-1}.*, I_j.* \text{ From } R_{j-1} \text{ Left Join } I_j^F \text{ On } B.K = I_j^F.K
\]

Indeed this formula results from the characteristic properties of any full RIA detailed in Section 2. The left join formula here transforms the original one there for operational use. We using \( K \) instead of entire \( R_{j-1} \) denoted as \( R’ \) there. \( K \) should usually have by far less attributes than any \( R_j \). It creates thus less join clauses and making the overall calculation faster.

In this way, to process a Create Table \( R \) statement, the DBMS may first create a base table, say \( R_B \), representing \( B \). Then it may store each \( I_j^F \) as a view, say \( I_j^F \text{-}V \), produced each from IE \( I_j \) with the renaming of \( R \) to \( R_B \) in \( I_j \) at least. Some \( R_j \) may need alternatively the renaming to \( R_{j-k} \); \( k > 0 \), when it refers to an IA created by \( R_{j-k} \). For instance, in Example 2, \( I_2 \) is \( \text{WEIGHT\_T} \), hence \( R_2 \) would refer to \( R_1 \).
where \( I_1 \) named WEIGHT_KG creates the IA WEIGHT_KG. Each \( R_j \) may be a temporary view, say named \( R_j_T \), except for \( R_n \), renamed simply \( R \). These views would be produced by DBMS while it evaluates a query. Alternatively, the DBMS could keep them persistent as all the others. Yet alternatively, RIA layer could dynamically create only a single view defining \( R_i \) defined then by a single imbricated left join expression, combining all views \( R_j \). This strategy seems less general however. Several imbrications could exceed the operational possibilities of a DBMS. Whatever is the strategy for the views, he RIA layer passes afterwards any query to its RIA \( R \) to DBMS. This one processes the query as is, but towards the view \( R \). It sends the result back to the RIA-layer.

Example 9.1. Consider \( R = SP \) from Example 1.3, with IEs \( I_S \) and \( I_P \) thus. We have \( R_0 = SP_B = (S#, \ P#, \ QTY) \) and \( K = (S#, \ P#) \). Let it be also that \( I_1 = I_S \) (SNAME, STATUS, S.CITY) and \( I_2 = I_P \) (PNAME, COLOR, WEIGHT, P.CITY), although it could be the other way around. Now, the RIA layer generates \( I_1^F \) as:

Create View \( I_S_F \) As select S#, P#, SNAME, STATUS, S.CITY From \( SP_B \), S Where SP.S# = S.S# ;

Also, it declares \( I_2^F \) as:

Create View \( I_P_F \) As select S#, P#, PNAME, COLOR, WEIGHT, P.CITY From \( SP_B \), P Where SP.P# = P.P# ;

When a query comes in or before if the temporary view definitions should stay persistent, the DBMS represents \( SP_0 \), then \( SP_2 \) successively as:

Create View \( SP1_T \) As (select S#, P#, QTY, SNAME, STATUS, S.CITY From \( SP_B \) Left Join \( I_S_F \) On \( SP_B.S# = I_S_F.S# \) And \( I_S_F \) On \( SP_B.P# = I_S_F.P# \) ) ;

Create View \( SP \) As (select S#, P#, PNAME, COLOR, WEIGHT, P.CITY From \( SP1_T \) Left Join \( I_P_F \) On \( SP_B.S# = I_P_F.P# \) And \( I_P_F \) On \( SP_B.P# = I_P_F.P# \) ) ;

The RIA-layer passes then any query to RIA \( SP \) to DBMS as the same query but to view \( SP \).

---

Figure 3 The S-P3 DB. Above: RIAs. Below: Views and base tables possibly implementing these within an existing DBMS. Temporary views are in dotted lines.
Figure 3 illustrates a possible implementation of our RIA-based S-P DB. This one, say S-P3, has SP with its usual BA and IAs, the latter being inherited here from S through IEs I_S and I_P in Example 1. Next, S ignores EMAIL from Example 8, having only its usual BAs, but with STATUS inherited from SP in Example 1.4. Similarly, P has its usual BAs, but also IAs WEIGHT_KG and WEIGHT_T, calculated through the two IEs in Example 2. The upper part shows the three RIAs. The dimensions of each rectangle reflect the size of the contents for the user, i.e., the number of tuples and the number of attribute values per tuple in each relation in Figure 2. The lower part shows the base tables and views possibly implementing S-P over some current DBMS. These are generated by the RIA-layer following the rules we just stated. The dimensions of each base table rectangle again reflect the number of tuples and their size, in number of (base) attributes. The length is thus the same as for the RIA, but not the width. The dotted rectangles represent the temporary views. These could be alternatively permanent as well, as discussed.

As the figure shows, for three RIA schemes, the RIA-layer would generate thirteen relational schemes in DBMS. Three would be the base table schemes. Hence, they would also define the relational conceptual scheme of S-P in the BRV-model. The views would be there to help the queries to S-P with the logical navigation and value expressions. The virtual attributes could spare views for P and for P only. The currently available implicit join capabilities would spare nothing. As said, the RIA-layer could alternatively generate even only a single, hence perhaps by far more complex, view scheme per RIA. It would lead to the minimum of two schemes per each of our RIAs to represent it in an actual DBMS. But, this strategy could end up a bad idea, as pointed out. @

Performance wise, the storage for a full RIA R is in practice the same as for R_B. We have also shown that the optimal scheme with RIAs has the same size and the same base attributes as the optimal one for BRV-model. Hence, the storage for the values of these RIAs under the dialect DBMS is the same. The storage for the views is negligible (provided it is not materialized, as we suppose). The optimal DB with full RIAs, should cost thus only negligibly more than the optimal DB with the base relations only for the same application. Finally, for the above processing scheme, the query parsing overhead by the RIA-layer, appears clearly negligible as well. Altogether, the new capabilities that RIAs bring, comes thus practically with no operational overhead.

5. Conclusion

RIAs appear useful for relational DBs. First, they are a unique generic construct instead of the two current ones. Those are in fact only specific RIAs, i.e. a base or a view RIA only. We have illustrated with the biblical S-P database, that the relational conceptual schemes with RIAs should be usually more accurate. Full RIAs alleviate the well-known limitations of such schemes, the dark side of the normalization constraints. RIA-model makes in particular the popular ER model, proposed precisely because of these limitations, rather useless.

Next, we have shown that the optimal conceptual scheme for a DB under RIA-model and the current one under BRV-model, have the same number of relation schemes, with the same base attributes. The RIA-based scheme however reduces the logical navigation. It also may avoid views hiding complex value expressions, often necessary at present for user’s comfort. RIAs usefully generalize with respect to this feature the already popular practices of virtual attributes and of implicit joins.

The design rules for RIAs based on restated NFs and, also restated, Heath’s and Fagin’s theorems, appear about as easy to use as the current ones. However, the decompositions based only on these theorems are only a tip of the iceberg of known proposals. Future work could explore more of these proposals. Perhaps, by adapting to full RIAs the rules for the independent projections already mentioned. Or, one could look upon more rules for the lossless decomposition using an outer join, [JS90]. Next, we mentioned the formal analysis of the conditions for well-formed IEs. Finally, the implementation of the DDL and DML for RIAs over an existing SQL DBMS, using its SQL dialect as the
kernel, appears simple and without any practical loss of performance. The future work should perhaps start with such an implementation, e.g., over MySQL. Whatever major DBMS is chosen, the improvement to the usability should be a win-win deal. Finally, most of major DBMSs are now interoperable multidatabase systems, [LA86]. RIAs with multibase IEs seem therefore attractive as well.

References

[C70] Codd, E., F. A Relational Model of Data for Large Shared Data Banks. CACM, 13,6,1970.