Relations with Base and Inherited Attributes

Witold Litwin

(March 2016)

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 construct instead, possibly mixing both types of attributes. We termed it Relation with Inherited Attributes (RIA). Examples showed the proposal attractive. No one jumped however on the idea. We now revisit our proposal. We show that the relational conceptual schemes with RIAs may be more faithful to reality. RIAs may further 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. These are cumbersome in turn. 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 applied Codd’s (relational) model, [C69] & [C70] has two basic constructs: a base relation and a view. Both are finite relations with atomic attributes only, in 1st Normal Form (1NF) thus. A base relation, often called implicitly a relation or a (relational) table, has base attributes (columns) only. These are supposed all stored. A view has only the inherited ones. Those are supposed derived from base relations or other views through some data manipulation language (DML) expression, e.g., an SQL Select. The two constructs are dichotomic with respect to the type of attributes they may contain. In 1992, we proposed instead a unique richer non-dichotomic 1NF construct, [LKR92]. We call it Relation with Base and Inherited Attributes or, more implicitly, Relation with Inherited Attributes, since relation in a DB usually means a base one or RIA simply. An RIA may mix base and inherited attributes, as the names hint. If it actually does, it’s a new 1NF relational construct on its own. We qualify it below of full RIA. Otherwise, it reduces to the current constructs. Examples showed full RIAs useful. The idea seemed promising also for OODBs, à la mode then.

No one followed the lead however. Below, we revisit our proposal systematically. We call it from now on the RIA-model. We refer to Codd’s constructs only as to BRV-model (Either Base Relation or View model). We believe the reader familiar with the BRV-model and SQL in particular. We show that through full RIAs, the RIA-model brings new capabilities. It also naturally preserves all the current ones of the BRV-model. We restate the relational scheme design rules for RIAs. We show that a relational DB should often advantageously consist only from full RIAs. We discuss the implementation of the RIA-model over an existing DBMS. We show it rather easy and with almost no storage or processing overhead. We hope the model enters the practice, “better late than never”.

Next section details the RIA-model. We discuss the basic concepts and the SQL extensions for RIAs. We show the new capabilities of RIAs. The conceptual scheme with full RIAs may be more faithful to the reality. The ER-model becomes rather useless for the relational design. Queries may become free from the customary logical navigation through inter-relational joins. RIAs appear the first generally practical solution to this decades’ old annoyance. Likewise, full RIAs may spare to queries complex aggregate expressions, another old exasperation. Both troubles are unavoidable for BRV-model at present, unless hidden behind dedicated views. These are however also often cumbersome, enough to be rarely practiced.

Section 3 continues with the RIA-model specific schema design rules. We restate for RIAs the NFs other than 1NF. We also restate the Heath’s and Fagin’s lossless decomposition theorems. The
restated theorems create specific full RIAs instead of the usual projections. The decomposition continues to be lossless, but usually is also “logical navigation less” over the projections. It is totally so for the restated Heath’s decomposition and mostly, in practice, for the restated Fagin’s one. Section 4 discusses the implementation of RIAs over existing DBMSs. Section 5 draws finally the conclusions and overviews the future work.

2. RIA Model

2.1 Schema Definition

In the nut-shell, RIA-model is the BRV-model plus full RIAs. Like any base relation or a view already, any RIA is thus a finite subset of a Cartesian product of atomic attributes over some domains, subject to any algebraic or predicative operations, and aggregate or scalar functions, applying to 1NF relations. By the same token, any inherited attribute (IA) of an RIA results from a formula, called from now, inheritance expression (IE), we detail soon. A base attribute, (BA), of an RIA has as usual a base value. A base value is typically stored. An inherited value is basically not stored. However, as known, it might, becoming 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. As said, we have a full RIA when both sets are not empty. Context wise, we may call it simply RIA. Otherwise, when we refer 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 RIA may inherit 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 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 V. We require all such subsets to be disjoint. The attributes of V are their union. Finally, I’s should not present any name conflicts. All IA names should be thus different, original names being perhaps renamed.

Below, we define any IE using an SQL Select. As for views, this seems the most convenient at present. In particular any IE should have a name that may be explicit or implicit as it will appear. The name may serve a DDL (Data Definition Language) statement to refer to the IE, as we’ll show as well. Next, 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’.

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. The Select clause should act then so that the join attributes that are all these of R’ besides, make the final projection selecting for each tuple of R’, only one tuple with
from $R'$. As wished, all this preserves 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 their collection 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 analysis for the future.

We consider finally that the DDL 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 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. We use the biblical Codd’s Supplier-Part relational DB, originally illustrating his proposal. We refer below to its probably most known version, popularized by C.J. Date, [D4]. It is often named S-P DB in short. The example restates S-P using RIAs. We refer to the original S-P as to S-P1. We call the restated one S-P2.

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 the S-P2 scheme. It 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 with their values. These form $B$ here. It also keeps the original primary key. $SP$ however also carries now IAs. It became thus a full RIA. The choice of IAs means that the DBA considers every property of a supplier or of a part, as also, 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. We named the $IE$ as $I_{SP}$.

Here, $V$ consists from all and only IAs in $S$ and $P$, except the key attributes $S.S#$ and $P.P#$. $V$ 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 $(S\#, P\#, \text{QTY})$. 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 $v \in V'$; $v = (Smith, 20…)$ that 1st line in $SP$ in Figure 2 inherits. The final $SP$ in Figure 2, i.e., $R$ above, is $SP := (S\#, P\#, \text{QTY})$ left join $(S\#, P\#, \text{QTY}, \text{SNAME}…\text{PCITY})$ AS $X$ on $SP.S\# = X.S\#$ and $SP.P\# = X.P\#$ and $SP.QTY = [SP.S\#, SP.P\#, SP.QTY, \text{SNAME}…\text{PCITY}]$. 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.

<table>
<thead>
<tr>
<th>S-P2 Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table S</td>
</tr>
<tr>
<td>S# Char,</td>
</tr>
<tr>
<td>SNAME Char,</td>
</tr>
<tr>
<td>STATUS Char,</td>
</tr>
<tr>
<td>CITY Char;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 1 S-P2 scheme.

<table>
<thead>
<tr>
<th>S-P2 Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table S</td>
</tr>
<tr>
<td>S#</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>S4</td>
</tr>
<tr>
<td>S5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Table SP</td>
</tr>
<tr>
<td>S#</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>S4</td>
</tr>
<tr>
<td>S4</td>
</tr>
<tr>
<td>S4</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 only. 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 hypothetical part P7 supplied by S1. If so, the following IE could do:

I_SP1 (Select SNAME, STATUS, S.CITY, 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, S.CITY 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. E.g. the supplier supplying 100 - 199 parts total has status 1 etc. 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, called SUPPLIES, with P#, PNAME and QTY for each supply. The list elements should be sorted in descending order on QTY. If the supplier does not supply anything for the time being, SUPPLIES should be null. 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), \([L3]\).

\[
(Select \ \text{LIST} \ (P\#, \ PNAME, \ QTY) \ As \ \text{SUPPLIES} \ From \ S, \ SP \ where \ S.# = SP.S# \ Group \ By \ SP.S# \ Order \ By \ Qty \ Dsc, \ PNAME \ Asc)
\]

The result for S1 would be S tuple:

\[
(S1, \ Smith...London, \ (P3, \ Screw, \ 400 ; \ P1, \ Nut, \ 300 ;...P6, \ Cog, \ 100)),..., \ (S5..., \ Athens, \ null)
\]

Notice that without the aggregation, 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. Observe, finally, that the simplest SQL query Select * From S is now equivalent to that requiring basically a half outer join between S and SP at present, i.e., in S-P1. The latter is more procedural thus. Actually, without SUPPLIES, the equivalent query would be more involved even with S-P2.@

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.

   \[
   \text{Alter Table P Add} \ (Select \ \text{WEIGHT} \ / \ 2.1 \ As \ \text{WEIGHT}\_\text{KG}, \ \text{WEIGHT}_\text{KG} \ / \ 1000 \ As \ \text{WEIGHT}_\text{T} \ From \ P \ X, \ P \ Where \ X.\text{WEIGHT} = P.\text{WEIGHT}, \ (Select \ \text{WEIGHT}\_\text{KG} \ / \ 1000 \ As \ \text{WEIGHT}_\text{T} \ From \ P \ X, \ P \ Where \ X.\text{WEIGHT}_\text{KG} = P.\text{WEIGHT}_\text{KG});
   \]

   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:

   \[
   \text{Alter Table P Add I_W (Select \ \text{WEIGHT} / 2.1 As \ \text{WEIGHT}_\text{KG}, \ \text{WEIGHT}_\text{KG} / 1000 As \ \text{WEIGHT}_\text{T} \ From \ P \ X, \ P \ Where \ X.\text{WEIGHT}_\text{KG} = P.\text{WEIGHT}_\text{KG});}
   \]

   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:

   \[
   \text{Alter Table P Add I_W (WEIGHT / 2.1 As \ \text{WEIGHT}_\text{KG}, \ \text{WEIGHT}_\text{KG} / 1000 As \ \text{WEIGHT}_\text{T});}
   \]

   For the same reason, the IE(s) could be the other way around: \(\ \text{WEIGHT}_\text{KG} \ \text{WEIGHT} / 2.1, \ \text{WEIGHT}_\text{T} \ \text{AS} \ \text{WEIGHT}_\text{KG} / 1000\ldots\).

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#);

Notice that after the alterations to P above and to S here, S-P2 has no base relations anymore, only (full) RIAs.

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#);

Notice that this Alter refers to the IE I-SP by its name (only). Observe also that with all the Alter statements taken care of, S-P2 would still have its three relations only, but they would all become full RIAs. Actually, it would be the optimal scheme for our DB, as it will appear in Section 3.@

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 obviously to base or view RIAs, but also thus to full RIAs. One may project, select or join thus full 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 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 it is 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 all their inherited ones.

2.3 Utility of RIAs

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

Both the logical navigation and the value 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, i.e., avoiding to queries both the logical navigation and complex expressions, without dedicated views. The result should be useful. One reason is that these capabilities of RIAs expand and put under a single umbrella in fact those 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*, helps with the logical navigation. As RIAs do, sometimes more extensively.

To justify our claims, we focus on S-P1. Countless actual DBs use nevertheless S-P1 as the 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, 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 drop I_SP from the S-P2 scheme. Any queries to S-P1 under RIA-model, amount then to these under BRV-model only.@

We now justify our 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 in a single base relation scheme. That is why we use n-ary relations and not, e.g., 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 anymore. A query or some view, say Status, must provide its values it instead. No such constraint for S in S-P2, benefiting from the IE STATUS.@

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 easily imagine practical use of these data, despite the inherent simplicity of 1NF. In contrast, many did not swallow SP scheme. It’s hard to imagine an actual supply characterized by the three attributes there only. Names at least seem a practical must.

Some could think that the obvious way-out was simply to 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, because of 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. SP as in S-P1 seemed after all preferable, as widely known. Its conceptual modelling insufficiencies could be compensated somehow again by additional dedicated view(s), the full (universal) one of S, P and of SP in S-P1 in particular. But this would constitute the other of the discussed limitations of the BRV-model. SP as RIA in S-P2 avoids also these troubles.

Actually it’s worth recalling that a heated debate followed the SP scheme proposal. The popular Entity-Relationship model, (ER), resulted from, [C76]. The ER diagram was supposed to be the actual conceptual scheme of a relational DB. This one should be modelled further by the relation schemes, constituting then internal logical scheme. For S-P, S & P relations were then modelling ER-entities. SP
was modelling in contrast “only” the ER-relationship between. The three attribute could suffice then. However, convincing folks that a box of parts they face is “only” a relationship seemed not obvious. Likewise, the question whether a marriage is an entity or a relationship never got a unanimous answer. Again, using RIAs, avoids such esoteric troubles. No need for the ER-model neither.

Logical Navigation and Complex Value Expressions

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. It annoys most users, as known for decades, [MUW84]. 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.@

Virtual Attributes

Our next claim was that full RIAs generalize in fact some popular practices already beyond BRV-model, as originally defined. The first 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 WEIGHT_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.

Implicit Joins

There were several the logical navigation savers proposed. One group steamed from the universal relation idea, we recall in next section. An alternative principle was called implicit joins, sometimes
called now also automatic, [L85], [LSW91]. The universal relation based proposals, despite strong excitement they have created in research milieu, [M04], did not make to popular DBMSs. The implicit joins made it, e.g., to SQL Server & MsAccess. As for the virtual attributes, the industrial versions again limited the 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 for the QBE interface. They make easier graphical queries, translated then to SQL, with the joins made explicit. In a QBE query graph, the implicit joins are directed or undirected arcs. They pop up once one selects the query relations, represented as the graph nodes. 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 of 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. The join attributes must share the name then and one must be a primary key. The automatic join is always an inner equijoin.

The attributes involved may be composite. The SQL query generated from can however be often strange then, at best. The reasons for are perhaps clear for Microsoft.

If an arc primary-foreign key exists between two tables, then the table with the foreign one may also automatically become 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 some preexisting view of all these tables, perhaps even the universal one, that would be the only way toward the goal under BRV-model. The implicit joins act as view-savers, like 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 virtual attribute did for their goal. 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 view-saving complex values expressions avoidance capabilities we discussed. The implicit joins do not provide these. Summing up, the RIA-model usefully generalizes also this popular practice. Revealing finally a single umbrella for both discussed practices, what we claimed as well.

The umbrella role brings an additional worth mentioning practical advantage on its own. Examples clearly showed that if there is a choice for a full RIA, say R again, multiple IEs should be usually preferable. To avoid the discussed troubles without RIAs, i.e., under the BRV-model, one way is to create for each IEs a somehow equivalent partial view. This one should have as Create View scheme the IE augmented with the join attributes with R in the Select clause. At the end, one must combine
all the partial views into a final one, equivalent to a full view of R. Using a full RIA instead, (the umbrella), one first avoids the partial views through simpler expressions. Those are in addition implicitly integrated. Avoiding perhaps the fancy naming conventions on the views, we spoke about, hinting to the common purpose. Most advantageously, the umbrella totally avoids the task of the final view, since the combination of IEs is always implicit. That task should usually be boring and error prone, at best. At worst, it could have an unfortunate end altogether, nesting perhaps too many views for DBMS operational capabilities.

3. RIA-model Schema Design

The relational scheme design rules have been formulated for the BRV-model only. They have to be revisited for full RIAs. We now address this issue, continuing with S-P2 as the motivating example. We first restate the NFs. Next we restate the Heath’s and Fagin’s theorems. We show that full RIAs benefit both lossless decompositions resulting from. They avoid the logical navigation however more for 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) in S-P1 is in BCNF, while 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. This could contradict S, where SNAME is anyhow already. Using SP instead, avoids the trouble.

To restate the NFs for the RIA-model, recall first that any RIA is in 1NF by definition. Hence no need to restate this NF. The other forms have to be restated to include full RIAs somehow. Observe in this context that the above anomalies of SP’ would not exist for a view SP’. We therefore state that a full RIA R (B, V) is in iNF or BCNF, iff R[B] is in iNF 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 these NFs. Same happens, 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, a full RIA 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 base attribute, SP would cease to be in BCNF etc.@

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 may be full RIAs. 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 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 any decomposition of an RIA R is not only lossless, but also at least one of the projections possibly preserves the attribute cover of R, i.e., possibly has all and only attributes of R. The result aimed on is that the scheme with the projections possibly as the “logical navigation less” as was R. This will appear possible only through the projections being 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 full RIAs advantageously spares the discussed 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 discussed logical navigation.
(b) Otherwise, a decomposition removing an MVD may still spare the logical navigation to some queries addressing the projections, but not to all.
(c) For a real-life DB, we may reasonably expect the discussed logical navigation spared to most or even all queries.

Indeed, first, the Heath’s theorem states, we recall, that for any base 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, 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 to queries selecting B and C.

Next, the Fagin’s theorem also states that in presence of MVD A ->> 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 -> B’ and 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: 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 B’ 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-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 as well. To illustrates the restated Fagin’s one, we modernize S-P. Each supplier may have email addresses for contact about any of its supplies. Each address is the value of new base attribute EMAIL. Every address is for only one supplier. We redesign the S-P scheme under RIA-model accordingly. We call the result S-P3.

We start optimistically with the universal relation U, [MUV84]. In short notation we have:

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

Notice the necessarily different names for the supplier and part cities, unlike in S and P of S-P1 or S-P2. U is possibly the optimal base relation for S-P3, unless proven otherwise. What’s easy, since EMAIL already introduces the MVD: S# \(\rightarrow\) EMAIL | (SNAME, CITY, STATUS, P#...QTY). SPE is not in 4NF thus. Regretfully, the optimal S-P scheme cannot thus be 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:

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

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’ = (SNAME, CITY, STATUS) and B’ = \(\emptyset\). SE is in the (restated) BCNF. It would not be if any of its IAs, e.g., 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:

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

In the projections, we could by the way more conveniently rename PCITY and SCITY to simply 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: P# -> PNAME, COLOR, WEIGHT, PCITY. We thus apply the restated Heath’s theorem again to SP. On the It gets decomposed to:

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

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

\[ S \ (S#, \ SNAME, \ CITY, \ STATUS), \]
\[ P \ (P#, \ PNAME, \ COLOR, \ WEIGHT, \ CITY), \]
\[ SE \ (S#, \ EMAIL, \ (select \ SNAME, \ SCITY \ As \ CITY, \ STATUS \ from \ SP, \ SE \ Where \ SE.S# = SP.S#)), \]
\[ SP \ (P#, \ S#, \ QTY, \ (Select */S.S# \ *From \ S, \ SP \ Where \ S.S# = SP.S#), \ (Select */P.P# \ From \ P, \ SP \ Where \ P.P# = SP.P#)). \]

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 SPE could use the FD: EMAIL -> S#, leading to:

\[ SE \ (S#, \ EMAIL), \ SP \ (S#, \ SNAME… \ (Select \ EMAIL \ From \ SE \ Where \ SE.S# = SP.S#)) \]

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:

\[ S \ (S#, \ SNAME, \ CITY, \ STATUS) \text{ SE (S#, \ EMAIL) } P \ (P#, \ PNAME, \ COLOR, \ WEIGHT) \]
\[ SP' \ (P#, \ EMAIL, \ QTY, \ (Select */S.EMAIL, \ P.P# \ From \ S, \ P \ Where \ S.EMAIL = SP.EMAIL \ AND \ P.P# = SP.P#)) \]

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

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

\[ S \ (S#, \ SNAME, \ CITY, \ (Select \ INT (SUM(QTY)/100) \ As \ STATUS \ FROM \ SP \ GROUP \ BY \ S# \ WHERE \ S.S# = SP.S#)). \]

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

\[ (Select \ INT(SUM(QTY)/100) \ As \ STATUS, \ COUNT (*) \ As \ STATUS1 \ FROM \ SP \ GROUP \ BY \ S# \ WHERE \ S.S# = SP.S#).@ \]

Observe that we could restate Fagin’s theorem so to avoid completely the logical navigation through AB and AC. The price would be the RIA that would be in fact an additional view ABC (select A, B, C From AB, AC Where A.B = AC.A). We do not feel this price worth in practice, as our credo, is the avoidance of any additional views. Notice however that such decompositions lead to the minimal schemes fully avoiding the logical navigation even for MVDs. So maybe we’re too stringent, after all.

4. Implementing RIAs

The most tempting 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. One way is to implement a layer managing then RIAs, say RIA-layer, using the services of the DBMS, Figure 3. The DDL and DML statements for RIAs, i.e., at RIA-layer, should extend to RIAs the statements of the DBMS dialect for base tables, views and queries, as already discussed. The RIA-layer should parse accordingly these statements back into DDL and DML statements of the dialect. The layer should then pass these down for the execution, reformatting perhaps the returned results.
The RIA-layer may represent every base RIA as is with the dialect’s base table sharing the name and rest of the scheme. Likewise, a view RIA may implement as is. A Create Table statement passes down then as is or changes the name to Create View. Other DDL statements and DML ones (queries) about such RIAs only, also pass as they are.

Creation of a full RIA, say R(B, V), with V resulting from some IE(s) is in contrast clearly more involved. R is neither a base relation nor a view, hence no actual DBMS can execute the RIA-layer’s Create Table. RIA-layer must then represent R for the dialect as B and view(s) somehow taking care of IE(s). To evaluate the queries to R by the DBMS, the easiest seems to have at least the full view of R there, i.e. the one defined as select * from R, as if R was a base table or a view for the DBMS. This is of course impossible for a full RIA under the DBMS. One must reconstruct an equivalent full view from B and from the view(s) reflecting somehow the IE(s). 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^F_j 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:

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

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^F_j as a view, say I^F_j, 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^F_2 is 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^T_j, 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, 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^F_j 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^F_j 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_1, then SP_2 successively as:

Create View SP_1_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.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 illustrates a possible implementation of the following variant of our RIA-based S-P DB, say S-P3. We suppose this one to have SP with its usual BA and IAs with the latter inherited from S through IEs I_S and I_P in Example 1. Next, S ignores EMAIL from Example 8, having only its S-P1 attributes, but with STATUS inherited from SP in Example 1.4 and SUPPLIES from Example 1.6. 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 IAs. Recall from the discussion of the Alter statement that they are all full RIAs. In other words, unlike for any current DB, S-P1 in particular, S-P3 has no base relations. The dimensions of each rectangle reflect the size of the content for the user, i.e., the number of tuples and the number of attribute values per tuple in each relation in Figure 2, plus the two IAs for P. 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 the three full RIA schemes, the RIA-layer would generate fifteen 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**

The proposed construct appears useful for relational DBs. First, since it is unique and including the two current ones. Those are in fact only specific RIAs, i.e. a base or a view RIA. Next, we have shown that full RIAs may make the relational conceptual schemes usually more accurate. They 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 contains only or mostly full RIAs. A full RIA may then reduce the logical navigation. It may also avoid views hiding complex value expressions, often necessary at present for user’s comfort. RIAs usefully generalize with respect to these features the already popular practices of virtual attributes and of implicit joins. Finally, the implementation of the RIAs using an existing DBMS appeared rather easy and without storage and processing overhead in practice. Better late than never, the existing DBMSs should be expanded accordingly.

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. The additional freedom from the logical navigation appears a major gain. However, the decompositions based only on these theorems are only a tip of the iceberg of known proposals. Future work could explore more of those. 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 My-SQL. 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.


