Understanding the implementation of *IS-A* relations[1,2]

J-L. Hainaut, J-M. Hick, V. Englebert, J. Henrard, D. Roland

Institut d'Informatique, University of Namur, rue Grandgagnage, 21 - B-5000 Namur
jlhainaut@info.fundp.ac.be

Abstract. Generalization/specialization hierarchies (IS-A relations for short) are basic semantic constructs proposed in most information system conceptual models. At the other side of design methodologies, where standard DBMSs are used, and will still be used for several years, there is no explicit representation of these IS-A relations. As a consequence, all the current methodologies include rules through which these semantic constructs are transformed into standard structures. However, it quickly appears that the translation rules proposed are most often incomplete, and sometimes incorrect. This fact has been experienced by many practitioners, who are faced with complex translation problems, but who do not find satisfying help neither in modern text books, nor in CASE tools. The aim of this paper is to analyze IS-A relations in some detail, and to propose a wide range of correct techniques to express IS-A relations into standard constructs. Understanding these techniques has also proved essential in reverse engineering processes.

Keywords ER model, conceptual modelling, supertype/subtype hierarchy, IS-A relation, schema transformation, implementation, logical design, physical design, reverse engineering

1. Introduction

Semantic models commonly include generalization/specialization abstraction mechanisms to structure knowledge according to taxonomic or subsetting structures. Originated in the SIMULA language, the concept was quickly adopted by the artificial intelligence and database communities, then appeared again as a basic programming language paradigm, in Smalltalk [8] and C++ for instance. In the database realm, it was mentioned in [6] then discussed in [23]. This common concept was the origin of some attempts to merge these three domains (e.g. [3] and OODBMS).

Though their names (generalization, specialization, subset, supertype/subtype, inheritance hierarchy, *is-a* relation, etc) and the symbols used to denote them (directed arcs, triangles, inclusion symbols, etc) vary considerably, most methodologies and CASE tools propose some variants of this construct that will be called *IS-A relation* in this paper.

However, most current DBMS do not provide corresponding logical constructs. Hence the need for translation rules to express them into plain data structures such as columns, keys and generic constraints, to be coded into SQL `check` predicates or `triggers`. The problem has been recognized as early as 1976 [6]. Since then, numerous papers and text books have proposed translation rules for IS-A relations, in such a way that this problem can be though to be solved, and therefore considered as out of interest by the scientific community. Fig. 1 shows a simple, but representative example of the way subtypes are implemented according to many text books.

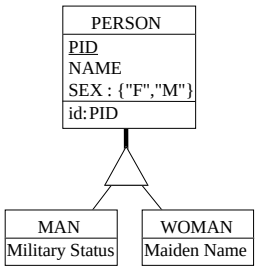
As far as the authors know however, there is no scientific articles, no text books and no CASE tools which propose a complete set of rules that can translate all kinds of specialization hierarchy into equivalent operational data structures. Indeed, they are based on an incomplete IS-A model which ignores useful configurations, or they propose incorrect transformations, or ignore some standard techniques (proposed by others or found in implemented databases through reverse engineering techniques), or they do not translate IS-A relations completely, ignoring important integrity constraints.

---

[1] This study has been carried out in the DB-MAIN project, partially supported by the *Région Wallonne*, the *European Union*, the *Communauté Française de Belgique*, and by the industrial: ACEC-OSI (Be), ARIANE-II (Be), Banque UCL (Lux), BBL (Be), Centre de recherche public H. Tudor (Lux), CGER (Be), Clin. Univ. St Luc (Be), Cockerill-Sambre (Be), CONCIS (Fr), D'Ieteren (Be), DIGITAL, EDF (Fr), EPFL (CH), Groupe S (Be), IBM, OBLOG Software (Port), ORIGIN (Be), Ville de Namur (Be), Winterthur (Be), 3 Suisses (Be).

[2] This paper is an abstract of [16].

This situation must be interpreted as an evidence that IS-A relations are much more complex than generally estimated, and that they deserve some more effort from the scientific community before practitioners can use them reliably.




```
create table PERSON(PID..,NAME..,SEX..
       primary key(PID))
create table MAN(PID..,MILITARY_STATUS..
       primary key (PID),
       foreign key(PID) references PERSON)
create table WOMAN(PID..,MAIDEN_NAME..
       primary key (PID),
       foreign key(PID) references PERSON)
```

Figure 1 *- Standard SQL translation of a subtype structure.*

The purpose of this paper is to analyze in some detail the properties of the standard IS-A model, the translation rules that eliminate IS-A relations, and SQL mapping rules. Though we intend to keep the presentation as simple and intuitive as possible, we will try to propose a generic and fairly rigourous treatment of the problem. In this paper, we will use the notation **`B is-a A`** to state that

> *entity type (or entity set, or object class, etc)* ***B*** *is a subtype (or subset, or specialization, or subclass, etc) of entity type (...)* ***A****, that in turn is the supertype (or superset, or generalization, or superclass, etc) of* ***B****.*

We will also say that an *IS-A relation* holds from B to A.

In order to make the proposals as general as possible, we will develop a three-step framework for processing the IS-A relation, as illustrated in Fig. 2. First, we propose a set of techniques to replace IS-A relations with standard, DBMS-independent, ER constructs, such as relationship types, attributes and constraints, then we propose techniques to translate these standard constructs into DBMS-specific structures and constraints (e.g. relational), and finally we suggest coding patterns to express the latter into the DDL of the DBMS (e.g. SQL). This structure, which can be found in popular database development methods [1], provides a powerful approach to analyze IS-A relation translation according to various DBMS models and to different strategies. By reversing the transformation processes, we are also provided with a generic framework to elicit IS-A relations in existing database schemas, specially if non standard rules were applied when it was developed [11]. It should be clear that this framework is intended to analyze design and reverse engineering practical situations in a rigourous way, and that it is in no way a proposal to deal with IS-A relations in actual DB design methodologies. Indeed, the latter often propose more straighforward mapping schemes.

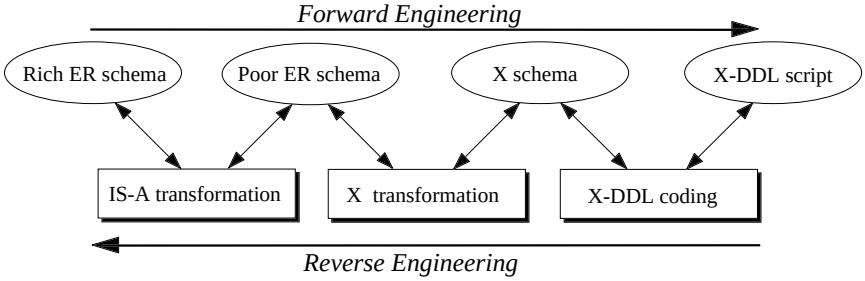


Figure 2 *- General framework to analyse the translation of a schema with IS-A relations (rich ER schema) into the DDL of a DBMS based on data model X, and conversely (X = relational in this paper).*

The paper is organized as follows. Section 2 describes the IS-A model underlying most semantic modeling approaches. Section 3 presents the notion of schema transformation. In Section 4, we present neutral IS-A transformation techniques, i.e. translation rules that replace IS-A relations with standard ER constructs, independently of the target DBMS. Section 5 describes how standard ER constructs can be expressed into relational data

structures, and Section 6 proposes some coding rules to express relational data structures into SQL.

2. An extended Entity-relationship model

2.1 Main concepts

In the discussion that follows we will make use of the DB-MAIN extended ER model, which is a wide spectrum formalism that supports both standard and non standard processes, such as conceptual, logical and physical design, schema optimization, normalization and integration, database reverse engineering, system migration, conversion, maintenance and evolution. This model has been described in several recent papers [11, 12, 13, 14], so that we can limit its presentation to mentioning the constructs that will be used in this paper (Fig. 3).

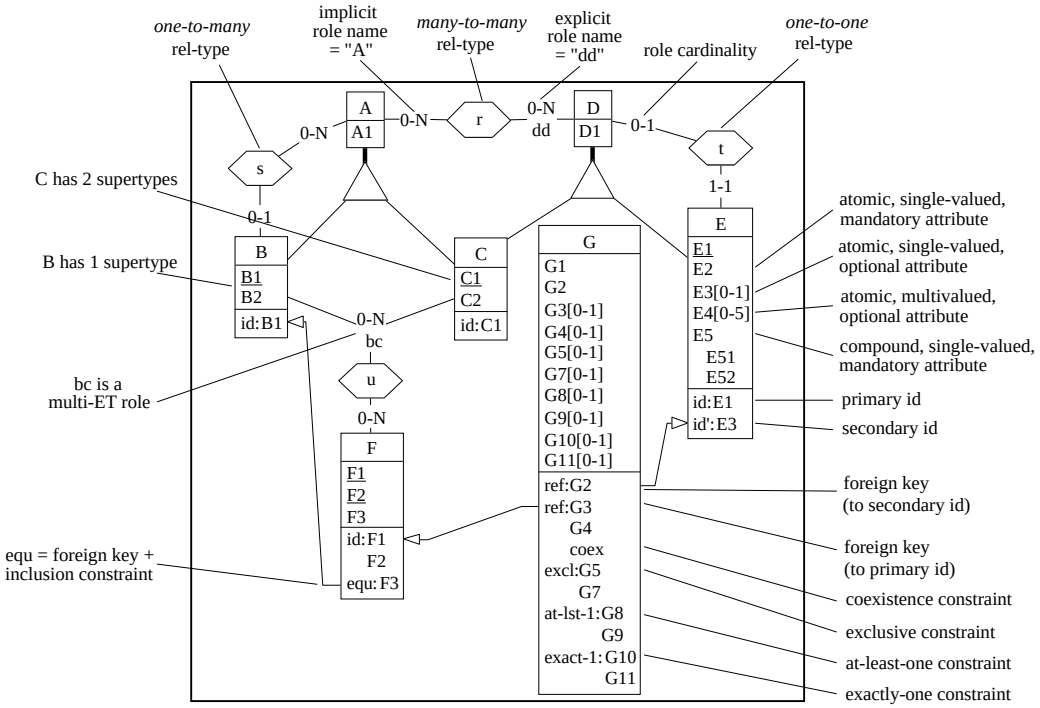


Figure 3 *- Main concepts of an extended ER model that offers IS-A constructs.*

A **conceptual schema** comprises mainly entity types, entity type attributes and relationship types. A collection of one or more entity types can be declared the subtypes of one or several entity types. An entity type can have one primary identifier (noted `id`, and generally underlined), and any number of secondary (or alternate) identifiers (`id'`). An identifier comprises attributes and/or roles. A relationship type, or *rel-type*, has two or more roles and any number of attributes; integrity constraints, such as identifiers can be associated with a rel-type. A role is generally defined on one entity type. It can also be defined on the union of several entity types (multi-ET role). In this case, in each relationship, this role is taken by an entity from one of the entity types. A role is characterized by cardinality constraints that specify through an interval in how many [min-max] relationships each entity plays this role. An attribute can be atomic or compound. It is characterized by cardinality constraints, stating how many [min-max] attribute values are associated with each parent object. Cardinality [1-1] is default, and is not represented. A group of attributes (and/or roles) can be constrained by one or several integrity constraints. A *coexistence* constraint (`coex:G3,G4`) states that either all the attributes have a definite value, or none have a value. An *exclusive* constraint (`excl:G5,G7`) states that at most one of the attributes has a definite value. An *at-least-one* constraint states that at least one of the attributes has a definite value. An *exactly-one* constraint (`exact-1:G10,G11`) means both *exclusive* and *at-least-one*.

A schema can also include **logical** (i.e. DBMS-dependent) constructs, a feature that is needed in logical design, but also in reverse engineering projects. For instance, a relational logical schema comprises (logical) entity types, to be interpreted as tables, (logical) attributes, to be interpreted as columns, identifiers (primary or alternate keys). In addition, a group of attributes can be used to reference a target entity (`ref:G3,G4`), and is to be interpreted as a foreign key. Such a key can reference either a primary id (`ref:G3,G4`), or a secondary id (`ref:G2`). If a reference group is associated with an inverse inclusion constraint, it is noted *equality* (`equ:F3`). Such a combined constraint specifies that the `F3` value of each `F` entity must be the value of `B1` of some `B` entity, *and conversely*.

Additional constraints will be described where they first appear in the discussion.

2.2 Semantics and typology of IS-A relations

The semantics of IS-A relations can vary from one model to another. Before going further, we have to define which semantics we will adopt in this paper. We consider first the two functions `char(E)` (returning the set of structural characteristics of E) and `pop(E)` (that returns the set of entities of E). Now, we consider the assertion `B is-a A` where A and B are two entity types of the same database. In the database realm, following Brachman's recommendation [2], the prevailing interpretation of this assertion is $pop(B) \subseteq pop(A)$. We consider the property $char(A) \subseteq char(B)$ as a mere consequence of this assertion. Hence the relations:

$$B \text{ is-a } A \quad \Rightarrow \quad pop(B) \subseteq pop(A) \quad \Rightarrow \quad char(A) \subseteq char(B)$$

Finally we consider the time-independent properties of *totality* and *disjunction* that apply on the set of subsets of a supertype, and that follow the standard definition (Fig. 4).

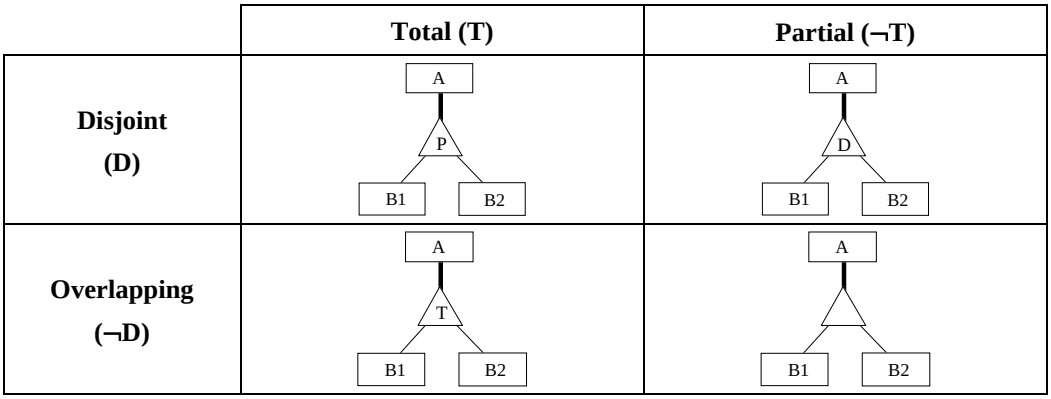


Figure 4 *- Graphical notation of subtype constraints. Two combinations have been given specific names:* partition = *total + disjoint,* free = *partial + overlapping.*

In the models proposed in the literature, the concept of IS-A relation has been refined in several ways. For instance, *predicate-based*, or qualified, *subtyping* [17, 25] consists in deriving subtypes from a predicate that states to which subtype each supertype entity must belong. Generally, this predicate defines a partition according to the values of an attribute of the supertype. According to another extension found in some ER models, a supertype can be given more than one collection of subtypes, sometimes called a *cluster*. Finally, we mention the *category* concept, introduced in [7], that generalizes the IS-A relation.

3. Schema transformations

A schema transformation is an operator that applies on a construct C of a schema S, and that replaces it with other constructs C', leading to new schema S'. C' is the target of source construct C through T, i.e. C' = T(C). Fig. 5 represents a popular example through which a binary relationship type, say PURCH, is replaced with an entity type PURCH and two one-to-many relationship types CP and SP.

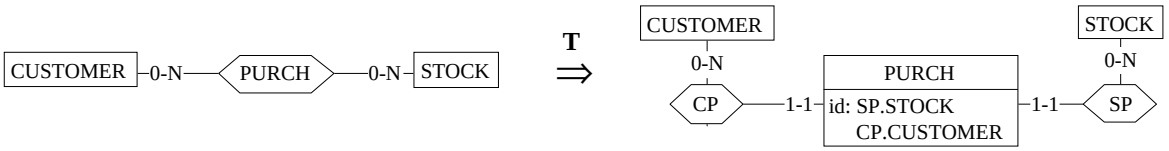


Figure 5 *- A standard transformation: expressing a relationship type as an entity type.*

To define transformations more precisely, we need a second mapping t, that specifies how valid instances of construct C are translated into C' instances: if c is an instance of C, then c' = t(c) is an instance of C'. We can denote a transformation by: Σ = <T,t>

There is a special class of transformations which satisfy the criterion of *semantics preservation*. Intuitively, a transformation is *semantics-preserving*, or *reversible*, if schemas S and S' describe the same real world portion. More precisely, if Σ1=<T1,t1> is a reversible transformation, there must exist a transformation Σ2=<T2,t2> such that:

for any construct `C`*, and any valid instance* `c` *of* `C`*:* `[T2(T1(C))=C]` $\wedge$ `[t2(t1(c))=c]`

There exists a higher-order kind of reversibility. S1 and S2 are called symmetrically reversible transformations, or SR-transformations, *iff*:

for any construct `C`*, and any valid instance* `c` *of* `C`*:* `[T2(T1(C))=C]` $\wedge$ `[t2(t1(c))=c]`

for any construct `C'`*, and any valid instance* `c'` *of* `C'`*:* `[T1(T2(C'))=C']` $\wedge$ `[t1(t2(c'))=c']`

A more detailed presentation of the concept of transformation and of its properties can be found in [9, 14] for instance. Several authors have proposed schema transformations as basic building blocks for database engineering. We will mention [1, 11, 13, 22] as some representative examples.

4. Neutral transformations to eliminate IS-A relations

We will concentrate on reversible transformations that cope with IS-A relations, and more specifically that express IS-A relations through standard constructs such as relationship types, attributes and constraints. These techniques are called *neutral*, in that they do not target any specific DBMS. We will discuss relational expressions later on, as the relational translation of the result of neutral transformations. Traditionally, three basic techniques have been proposed [1]:

- *IS-A materialization*, through which each subtype is represented by an independent entity type related to the supertype by a *one-to-one* relationship type (Fig. 6.1);
- *upward inheritance*, that represents the supertype only (Fig. 6.2);
- *downward inheritance*, that represents the subtypes only (Fig. 6.3); entity type A collects the entities that do not belong to subtypes B or C, if any.

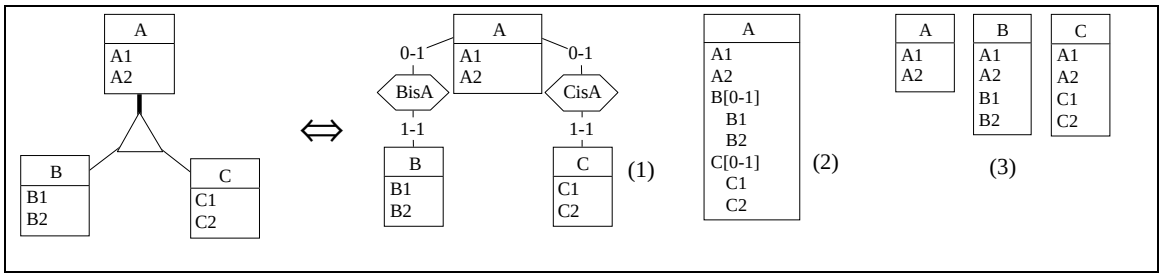


Figure 6 *- The three standard techniques to replace IS-A relations.*

In techniques (1) and (2), some authors include a *type* attribute (with domain {`'B','C'`}) in the new version of A, in order to indicate to which subtype(s) each A entity belongs. We have omitted such an attribute in this discussion since it would be redundant. Indeed, its value(s) could be deduced from the presence/absence of `BisA` and `CisA` relationships in which each A entity is involved (1), or of values of attributes B and C (2).

Generally, the authors who propose a general introduction to database design have to stop their presentation here, due to space limit. Generally too, here begin the problems of practitioners who use IS-A relations to structure their models, and who try to retrieve IS-A

structure in legacy systems. Indeed, addressing subtype constraints (T, ¬T, D, ¬D), roles played by supertypes and subtypes, and constraints such as identifiers, leads to much more complex schemas.

We will analyse these three aspects in some detail in the following. To keep the presentation simple, these constructs will be discussed independently. The rules for combined structures can be derived easily. Due to space limits, some special patterns have been discarded from this discussion; they are addressed in [16].

4.1 Expressing subtype constraints

The absence of constraints, i.e. *free* or (¬T & ¬D), leads to the three schemas of Fig. 6. Enforcing the D and T constraints in the *IS-A materialization* and *Upward inheritance* techniques is fairly straighforward: they translate into *exclusive* and *at-least-one* constraints holding among the representations of the subtypes (Fig. 7). The partitioning constraint translates naturally into *exclusive* and *at-least-one*, i.e. an *exactly-one* constraint.

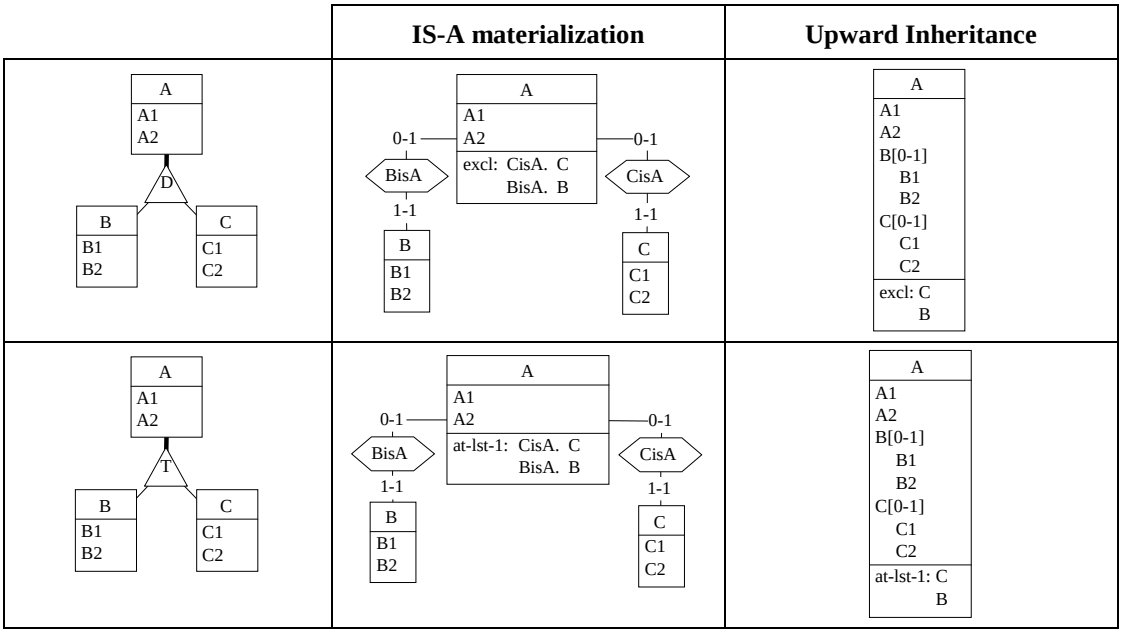


Figure 7 *- Expression of D and T subtype constraints (techniques 1 and 2).*

Let us observe that the *Downward inheritance* technique is valid for **disjoint** subtypes only. The other cases provide no means to identify the entities that belong to both B and C types, and therefore are not semantics-preserving. If ¬D holds, then A must have an identifier. On the other hand, the T constraint implies that only entity types B and C remains, while ¬ T leads to a schema including A, B and C. To summarize, constraints (¬T & D) are translated into Fig. 6.3 and constraint (T & D = P) are translated into the same schema, where A has been discarded; the two other cases require that A have an identifier, and will be discussed in the next section.

4.2 Expressing identifiers

There are two cases to consider: supertype A has an identifier (say A1) and subtype B has an identifier (say B1).

4.2.1 Supertype A has an identifier (A1)

The first two techniques preserve the supertype as well as its identifier A1. The third technique induces complex patterns that are illustrated in Fig. 8. The **D** constraint between subtypes must be expressed as a *disjoint* constraint among the identifier value sets: no A entity can have the same A1 value as any B or C entity, and so on for B and C. In addition, in the overlapping (¬D) situations, we have to state that whenever a B entity and a C entity

agree on A1, they must have the same A2 value as well, since they represent a single entity in the source schema. Hence the functional dependency holding in the union of the entity type populations.

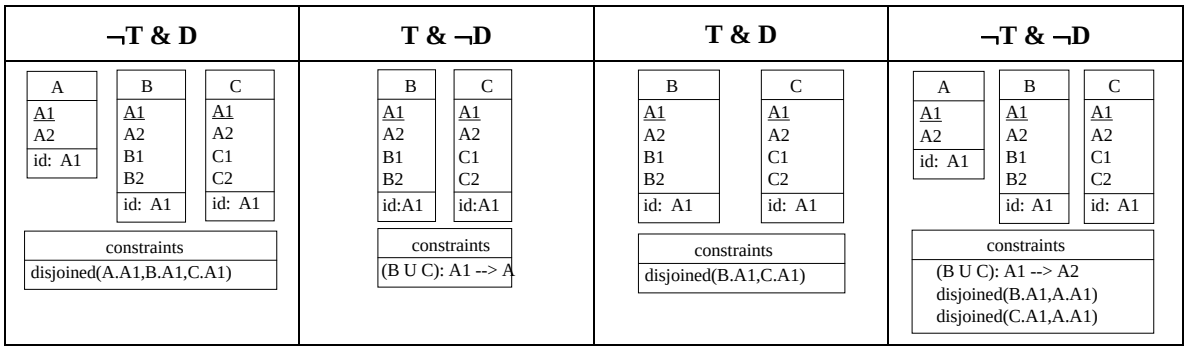


Figure 8 *- Propagation of supertype identifier A1 (Downward inheritance).*

4.2.2 Subtype B has an identifier (B1)

Techniques 1 and 3 preserve entity type B, so that its identifier B1 is preserved as well. Through technique 2, entity type B is replaced with attribute B, whose component B1 is made an optional identifier of A (Fig. 9).

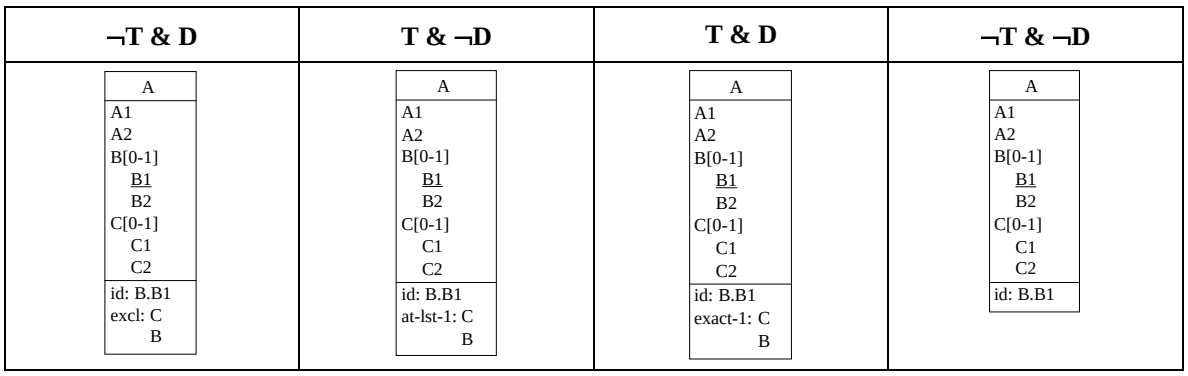


Figure 9 *- Propagation of subtype identifier B1 (Upward inheritance).*

4.3 Expressing roles

Here too, we have two cases to consider: supertype A plays a role in R (Fig. 10a) and subtype B plays a role in R (Fig. 10b).

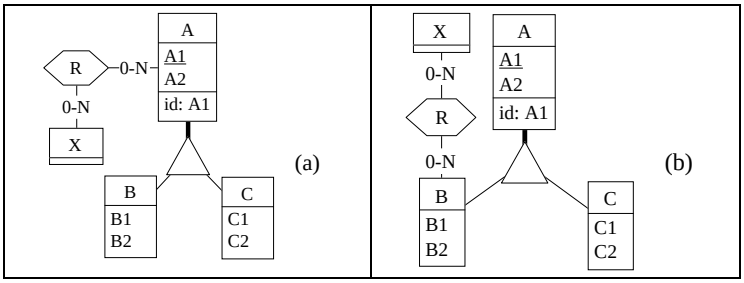


Figure 10 *- The supertype and/or subtypes can play roles.*

4.3.1 Supertype A plays a role in R

Techniques 1 and 2 preserve the supertype A as well as its roles. When technique 3 is applied, the role *R.A* is replaced with multi-ET role *R.bac* taken by A, B and C if the subtypes overlap, and by B and C if they don't (Fig. 11).

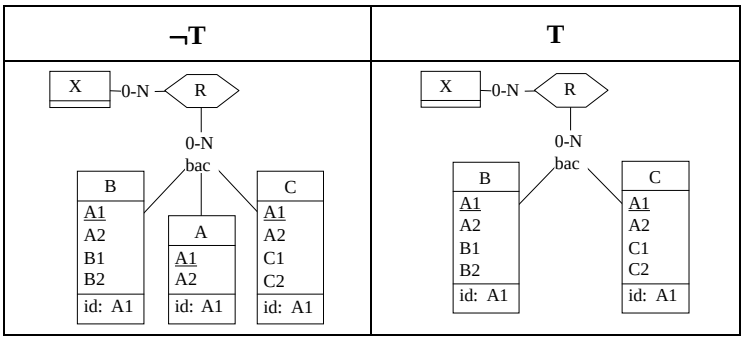


Figure 11 *- Expression of role* R.A *by multi-ET role* R.abc *(constraints of Fig. 8 omitted)*

4.3.2 Subtype B plays a role in R

Techniques 1 and 3 preserve subtype B as well as its roles. Following technique 2, the role R.B migrates to supertype A (Fig. 12). This migration induces an implication constraint stating that whenever an A entity is associated with X entities, then it must have a B attribute value.

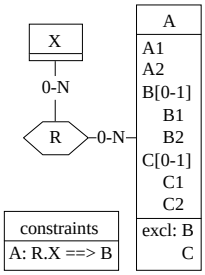


Figure 12 *- Through* Upward inheritance, *role R.B is replaced with role R.A.*

5. Relational representation techniques

The transformations proposed in Section 4 provide us with a rigourous way to get rid of IS-A relations, but they keep, and introduce several ER constructs which cannot be explicitly expressed into, e.g., relational structures. In particular, they produce *one-to-one rel-types*, *complex constraints on rel-types*, *optional compound attributes*, *multi-ET roles* and *implication constraints*. We will examine how each of these constructs can be transformed into pure relational structures. Standard ER structures (e.g. entity type, rel-type, all kinds of attributes) have long been given relational expressions [1, 24], and can be ignored.

5.1 Translation of one-to-one relationship types

We first note that such a relationship type is optional for the supertype (A) and mandatory for the subtype (say B). We mention three popular techniques.

(a) The most obvious translation of R consists in adding to B a mandatory, identifying, foreign key A1 referencing A, *provided A has an identifier* (Fig. 13a).

(b) If A has no identifier, but *B has one*, then R can be translated into an inverse foreign key B1 (Fig. 13b). This is a more complex technique, since it produces an optional, identifying foreign key + an inverse inclusion constraint (every B.B1 value must be the B1 value of some A entity), denoted by the equ clause.

(c) A third technique consists in merging B into A (Fig. 13c). There is no preconditions on identifiers. The result is the same as that of *Upward inheritance*, and will be studied in Section 5.3.

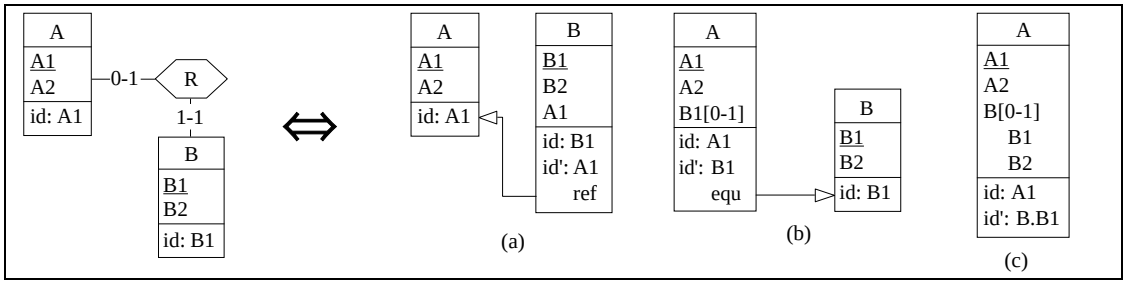


Figure 13 *- Three standard transformations of a one-to-one relationship type.*

5.2 Translation of *exclusive, at-least-1* and *exactly-1* constraints on relationship types

We must distinguish two situations, according to whether the relationship types translate into foreign keys in subtypes B and C (Fig. 13a) or in supertype A (Fig. 13b), which the constraint holds in.

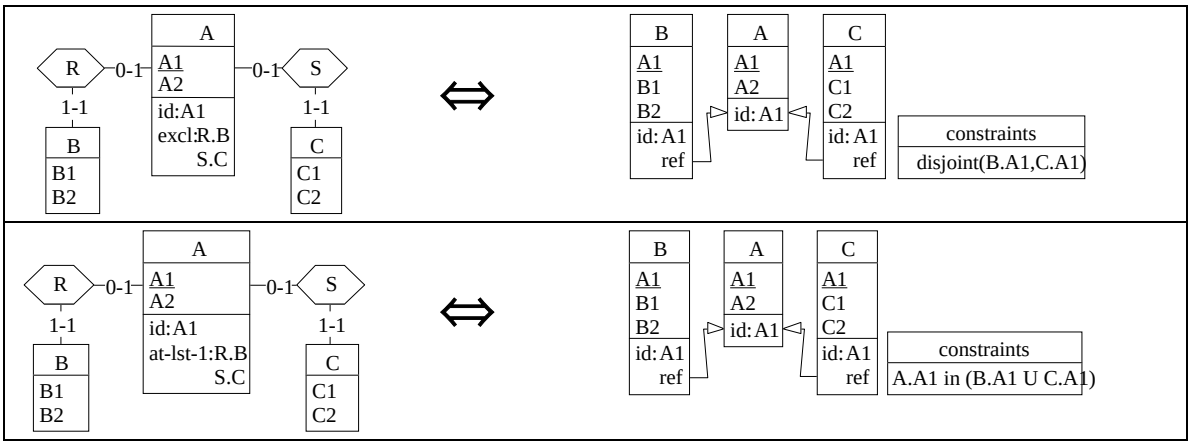


Figure 14 *- Transformation of constraints on relationship type (1).*

In the first case (Fig. 14), these constraints on rel-types are translated into constraints on subtype identifiers. Their interpretation is as follows:

- *exclusive* ⇒ `disjoint(B.A1,C.A1)`: the set of A1 values of B entities and the set of A1 values of C entities are disjoint;
- *at-least-1* ⇒ `A.A1 in (B.A1 ∪ C.A1)`: the A1 value of any A entity must belong to the union of the set of A1 values of B entities and the set of A1 values of C entities;
- *exactly-1 = exclusive* ∧ *at-least-1*.

The second case makes it possible to avoid these derived inter-entity constraints (Fig. 13b). Since these keys belong to the supertype, the source constraints are easier to assert.

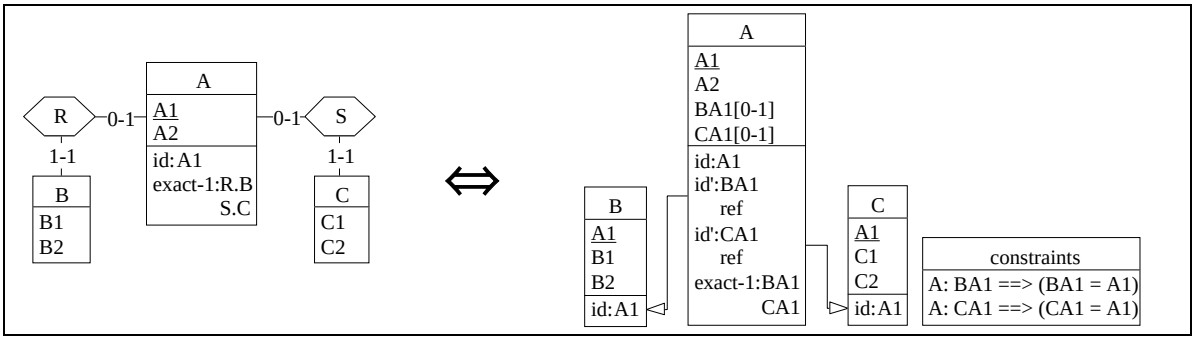


Figure 15 *- Transformation of constraints on relationship type (2).*

However, other constraints must be defined to garantee that each foreign key (BA1, CA1), when it has a value, is a copy of the identifier of A (Fig. 15)[3].
Another solution, based on foreign keys from the subtypes to the supertype, as in Fig. 14, consist in making these foreign keys reference *secondary identifiers* instead. The source constraints apply on the latter (Fig. 16).

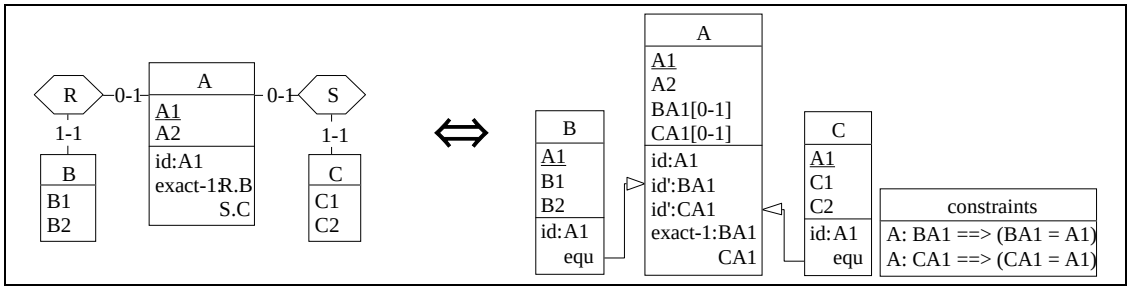


Figure 16 *- Transformation of constraints on relationship type (3).*

Predicate-based subtyping, based on discriminating attributes, has also been proposed by many authors. In [1] for instance, the supertype includes a subtyping attribute, say TYPE, that indicates to which subtype(s) each super-entity belongs. The cardinality of this attribute translates the subtype constraints as follows (*n* is the number of subtypes):

- disjoint ($D \wedge \neg T$) $\Rightarrow$ `TYPE[0-1]`
- total ($\neg D \wedge T$) $\Rightarrow$ `TYPE[1-`*n*`]`
- partition ($D \wedge T$) $\Rightarrow$ `TYPE[1-1]`
- free ($\neg D \wedge \neg T$) $\Rightarrow$ `TYPE[0-`*n*`]`

In order to simplify the implementation, the authors also propose to replace the multivalued versions by a single-valued attribute in which the subtype names are coded as a single value. Other authors propose to implement TYPE as a sequence of boolean attributes, each dedicated to one subtype. However, most authors ignore the constraints holding between the values of TYPE and the existence of referencing subtypes:

$\forall$ `a` $\in$ `pop(A): (a.TYPE="B")` $\Leftrightarrow$ ($\exists$ `b` $\in$ `pop(B): b.A1=a.A1)`
$\forall$ `a` $\in$ `pop(A): (a.TYPE="C")` $\Leftrightarrow$ ($\exists$ `c` $\in$ `pop(C): c.A1=a.A1)`

5.3 Translation of optional compound attributes

The optional compound attribute B can be extracted to form an individual entity type and a one-to-one relationship type. This technique has no particular interest since it produces a non relational construct, which is precisely the source schema of Fig. 13. Two specific techniques can be proposed.

(a) The first one consists in disaggregating B, replacing it with its components, and adding a coexistence constraint (Fig. 17a).

(b) The other is simpler but far less elegant. It consists in representing B by atomic attribute B', whose values are made of the concatenation of the values of the components of B (Fig. 17b). The lost structure of B can be recovered in a relational view, or in the application programs by storing B' values in variables with adhoc decomposition structures[4]. This technique is not applicable if individual components of B are involved in some integrity constraints, such as identifiers.

---

[3] When the subtypes are disjoint, some authors propose to merge all the foreign keys into a single one, and to add one or several discriminating boolean attributes indicating to which subtype(s) the entity belongs [18]. Such multi-target foreign keys can be translated into a series of SQL foreign keys defined on the same colomns.

[4] This technique is very frequent in actual files and databases, and makes one of the major problems in database reverse enginering [10, 15].

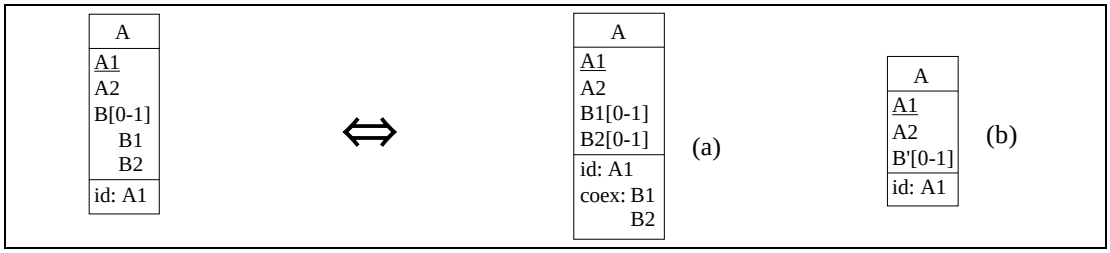


Figure 17 *- Transformation of an optional compound attribute.*

The problem is a bit more complex for compound attributes among which another constraint holds. Fig. 18 shows two translations of an *exclusive* constraints based on the same approaches as above. We observe that the source exclusive constraint `excl:B,C` should have been translated into `excl:{B1,B2},{C1,C2}`. However, thanks to the coexistence constraints, it has been simplified into `excl:B1,C1`.

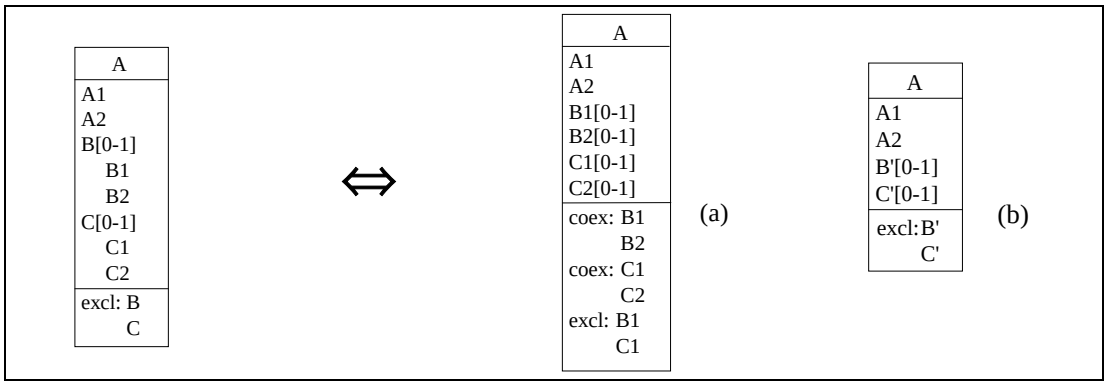


Figure 18 *- Transformation of optional compound attributes with an exclusive constraint.*

5.4 Translating multi-ET roles

Multi-ET role `bc` is eliminated by splitting its relationship type `R`, so that it is distributed among the entity types `B` and `C` on which it is defined (Fig. 19). This leads to plain ER structures which are easy to translate into relational structures.

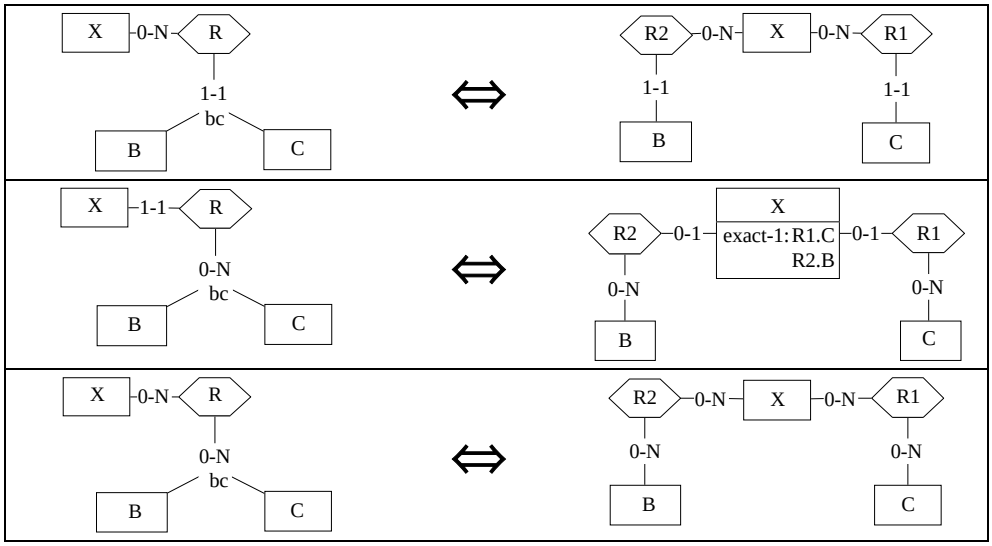


Figure 19 *- Distribution of a multi-ET role.*

5.5 Translating implication constraints

We consider the translation of two typical situations in which R is one-to-many from X to A (Fig. 20) and R is one-to-many from A to X (Fig. 21). In the left-side schema on Fig. 20, the constraint states that an A entity can be linked to an X entity only if it has a B value. The right-side schema translates this constraint for foreign key X1.

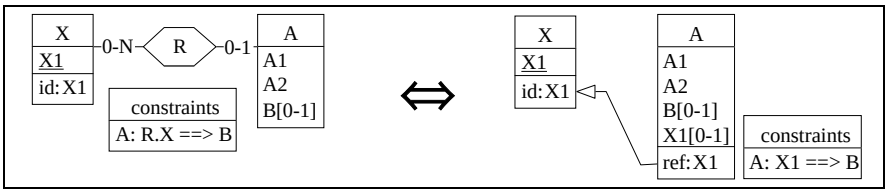


Figure 20 *- Expression of an implication constraint from a relationship type to an attribute (1).*

In Fig. 21, the constraint in the right-side schema tells that an A1 value in some X entity is the A1 value of an A entity only if this entity has a B value. More complex versions of R (many-to-many, N-ary, etc) can be coped with by applying transformation of Fig. 5 first.

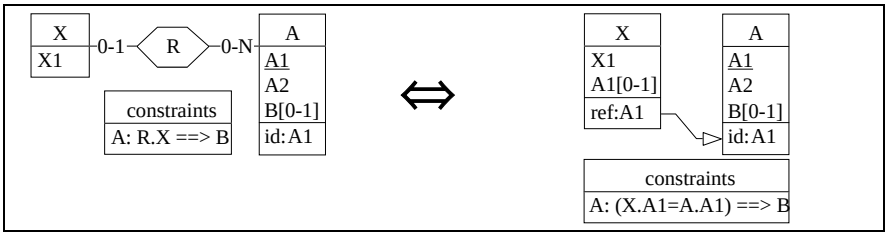


Figure 21 *- Expression of an implication constraint from a relationship type to an attribute (2).*

6. SQL translation of relational structures

Expressing entity types, single-valued and atomic attributes, primary, secondary and foreign keys in SQL is straighforward. However, the transformations proposed in Section 5 introduced some non standard constraints whose SQL expression can be delicate. In this section, we assume that the target RDBMS offers some variant of the `check` predicate as well as the `trigger` mechanism. Whenever possible, we will propose a translation into a `check` predicate. However, some RDBMS offers only a weak form of this mechanism, limited to local columns only. In this case, triggers will be used instead.

6.1 Translation of non standard keys

Some special cases of primary, secondary and foreign keys deserve special attention.

- Let us first consider *optional identifiers*. The first idea that comes to mind is to define a `unique` constraint or a `unique` index on the optional (nullable) columns. However, many RDBMS manage optional unique keys in an inadequate way: only one row is allowed to have a `null` value for this key. Therefore the programmer is forced, either to split the table or to resort to a `trigger` clause to express the uniqueness constraint (Fig. 22)
- The SQL expression of plain foreign keys is straighforward. That of Fig. 13b is less simple due to the inverse *inclusion constraint* (Fig. 22). This technique is not applicable in RDBMS that do not accept check predicates which mention other tables.

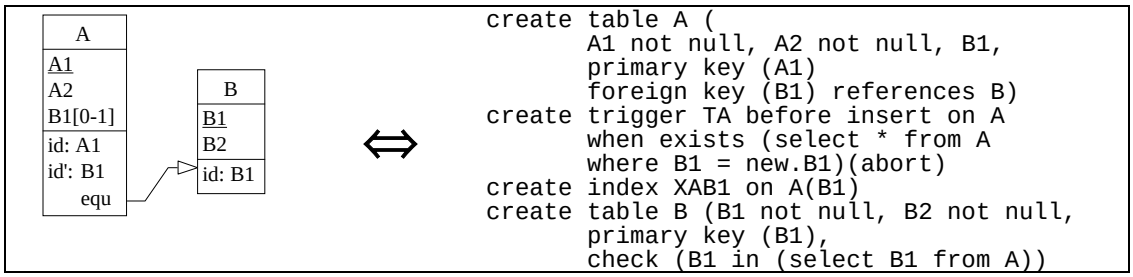


Figure 22 *- SQL expression of optional identifiers and inclusion constraints.*

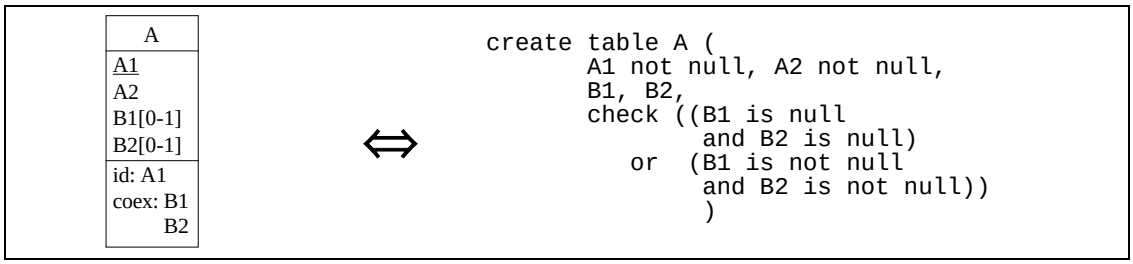


Figure 23 *- Expression of a simple coexistence constraint.*

6.2 Translation of *coexistence, exclusive* and *at-least-1* constraints on attributes

There is no problem to express these constraints into `check` or `trigger` clauses that are proposed by most current RDBMS. Fig. 23 and 24 illustrate the expression of simple and complex *coexistence* and *exclusive* constraints. *At-least-one* constraints will be translated in a similar way. *Exactly-one* constraints are formed by *exclusive* + *at-least-one* constraints.

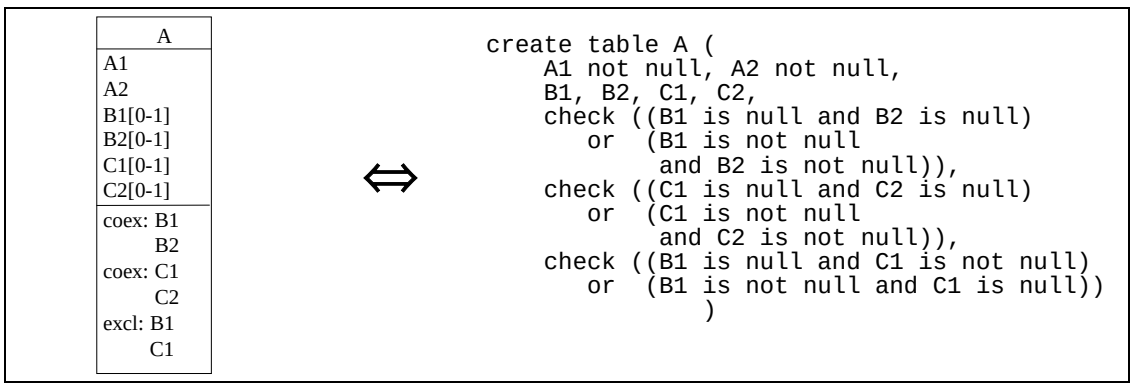


Figure 24 *- Expression of complex constraints.*

6.3 Translation of constraints on identifiers

These constraints involve the identifier value sets of several entity types. They are fairly easy to translate into `check` predicates, provided the RDBMS accept multi-table predicates. Otherwise, `triggers` will be used instead.

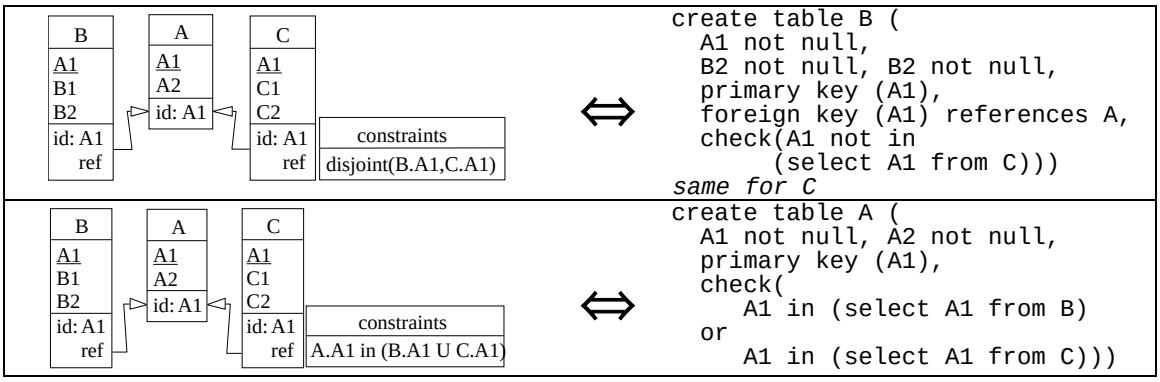


Figure 25 *- Expression of constraints on identifiers.*

6.4 Translating FD among entity types

Any attempt to normalize the schema by factoring the common {A1,A2} fragments of A, B and C would produce the result of *IS-A materialization* technique, and therefore is not worth being considered. The proposed technique, applied to table C for example, is as follows: the row (a1,a2, ...) can be inserted into C if no other tables include a row (a1,a2', ...) such that a2 ≠ a2'. This technique is not applicable in RDBMS that do not accept check predicates which mention another table.

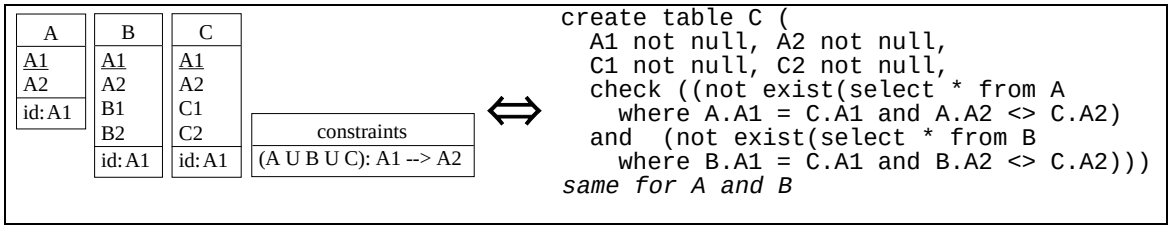


Figure 26 *- Expression of distributed functional dependencies.*

6.5 Translating implication constraints

Using the logical definition of the implication operator ($(P \Rightarrow Q) = (\neg P \vee Q)$), we can propose the translation of the two schemas of Section 5.5 (Fig. 27 and 28).

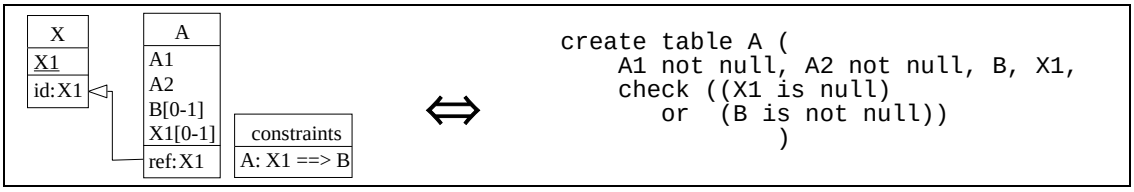


Figure 27 *- Expression of an implication constraint involving an internal foreign key.*

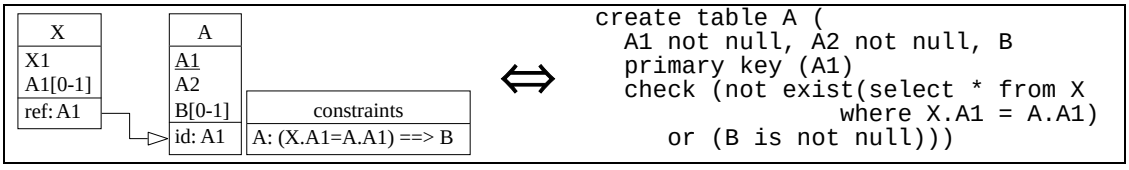


Figure 28 *- Expression of an implication constraint involving an external foreign key.*

The implication constraint in Fig. 15 and 16 can be expressed as in Fig. 29.

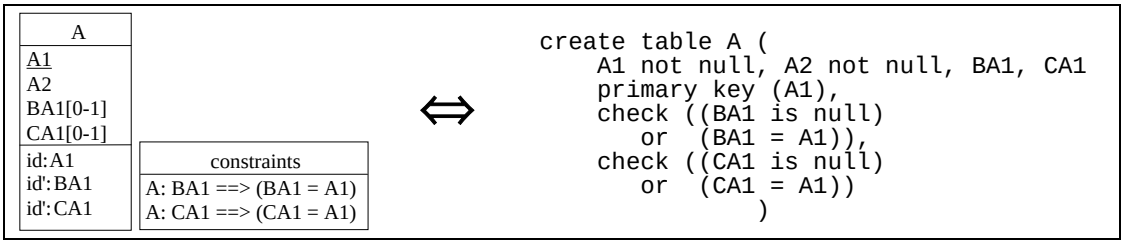


Figure 29 *- Expression of a conditional equality constraint.*

7. Conclusions

Through a precise analysis of IS-A relations translation techniques, we have shown that the problem is far more complex than generally presented in the literature, and that most proposed translation rules are too weak to provide reliable databases. We mention some of these proposals. Few of them include the four combinations of Fig. 4; [1, 26, 24] are some examples. [1] proposes the three basic implementation techniques, but with incomplete structure propagation and without derived constraints. [5] proposes the three implementation techniques, but for two combinations only, namely partition and free, and ignores derived constraints. [4] mention the first two techniques, but without any constraints. [24] and [21] propose the first technique without constraints.

The analysis performed in this paper can also translate into the following two conclusions.

1. When we compare the simplicity of the IS-A relations with the complexity of their complete translation into standard database structures, it appears that the availability of built-in supertype/subtype constructs (e.g. in SQL-3) will simplify database design and programming considerably.
2. Understanding how IS-A relations can be translated into standard structures is essential, not only in database design, but also in database reverse engineering. Indeed, traces of IS-A implementation have been found in most legacy databases. However, these implementations often use more complex translation techniques than those assumed in reverse engineering methodologies [10, 15]. In particular, all the techniques proposed so far are limited to the *foreign key + unique key* pattern illustrated in Fig. 1 and Fig. 13a. Not only this technique is only one of the possible translation approaches, but such a key pattern can be the translation of a mere one-to-one relationship type as well, in which case most published algorithms fail.

It is needless to explain why CASE support of IS-A relations is particular weak. Indeed, at best, these tools implement the uncomplete translation processes proposed in the literature. Even when the OO paradigm is included in the target implementation system (OO languages, OODBMS, AI shells), the developer has to cope with severe limitations. For instance, the subtype constraints often are weaker than those presented in Section 2.2 (e.g. the disjoint hypothesis is often assumed), leading the developer to transform natural semantic structures into an artificial ones.

To be complete from the methodological viewpoint, this analysis should include further development such as the following.

1. What SQL techniques can be used to recover the source supertypes and subtypes ? For instance, what are the virtues and the limitations of (1) *Upward inheritance* complemented by views that extract subtypes through selection/projection, and of (2) *IS-A materialization* complemented by views that rebuild subtypes through joins ?
2. What is the best way to translate an arbitrarily complex IS-A hierarchy into plain SQL structures ? Such a heuristics should address the problem of multiple supertypes, multiple clusters, the satisfaction of definite design requirements (simplicity, space efficiency, time efficiency, evolutivity) and the specific features of the target DBMS (for instance, some RDBMS accept local check predicates only). These issues are partially addressed in [20] for example.
3. Which heuristics can be used to elicit the complex integrity constraints developed in this paper from the source code of legacy application programs ?

8. References

[1] Batini, C., Ceri, S., Navathe, S., *Conceptual Database Design - An Entity-Relationship Approach*, Benjamin/Cummings, 1992

[2] Brachman, R., J., What IS-A Is ans Isn't: An Analysis of Taxonomic Links in Semantic Networks, *IEEE Computer*, Oct. 1983

[3] Brodie, M., Mylopoulos, J., Schmidt, J., W., (Ed.), *On Conceptual Modelling*, Springer-Verlag, 1984

[4] Casanova, M., Tucherman, L., A., Laender, A., H., F., Algorithms for designing and maintaining optimized relational representations of entity-relationship schemas, in *Proc. of the 9th Int. Conf. on ERA*, 1990, North-Holland, 1991

[5] Catarci, T., Ferrara, F., M., OPTIM-ER: an Automated Tool for Supporting the Logical Design a Complete CASE Environment, in *Proc. of the 7th Int. Conf. on ERA*, 1988, North-Holland, 1989

[6] Chen, P., P., The Entity-Relationship Model - Towards a Unified View of Data, *ACM TODS*, Vol. 1, No. 1, 1976

[7] Elmasri, R., A., Weeldryer, J., Hevner, A., The Category Concept: An Extension to the Entity-Relationship Model, *J. of Data & Knowledge Engineering*, Vol. 1, No. 1, 1985

[8] Goldberg, A., Robson, D., *Smalltalk-80: The language and its Implementation*. Addison-Wesley, 1983

[9] Hainaut, J-L., Entity-generating Schema Transformation for Entity-Relationship Models, in *Proc. of the 10th ERA*, San Mateo (CA), North-Holland, 1991

[10] Hainaut, J-L., Chandelon M., Tonneau C., Joris M., Contribution to a Theory of Database Reverse Engineering, in *Proc. of the IEEE Working Conf. on Reverse Engineering*, Baltimore, May 1993, IEEE Computer Society Press, 1993.

[11] Hainaut, J-L, Chandelon M., Tonneau C., Joris M., Transformational techniques for database reverse engineering, in *Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas*, E/R Institute and Springer-Verlag, LNCS, 1993

[12] Hainaut, J-L, Englebert, V., Henrard, J., Hick J-M., Roland, D., Evolution of database Applications: the DB-MAIN Approach, in *Proc. of the 13th Int. Conf. on ER Approach*, Manchester, Springer-Verlag, 1994.

[13] Hainaut, J-L., *Transformation-based database engineering*, Tutorial notes, VLDB'95, Zürich, Switzerland, Sept. 1995 (available at jlh@info.fundp.ac.be)

[14] Hainaut, J-L., Specification Preservation in Schema transformations - Application to Semantics and Statistics, *Data & Knowledge Engineering*, Vol. 19, pp. 99-134, Elsevier, 1996.

[15] Hainaut, J-L, Roland, D., Hick J-M., Henrard, J., Englebert, V., Database Reverse Engineering: from Requirements to CARE tools, *Journal of Automated Software Engineering*, Vol. 3, No. 2, 1996.

[16] Hainaut, J-L, Hick J-M., Englebert, V., Henrard, J., Roland, D., *Representation of IS-A Relations*, Research Report, DB-MAIN Project, Institut d'Informatique de Namur, March 1996 (available at jlh@info.fundp.ac.be)

[17] Halpin, T.A., Proper, H.A., Subtyping and Polymorphism in Object-role Modeling, *Data & Knowledge Engineering*, Vol. 15, pp. 251-281, Elsevier, 1995

[18] Hohenstein, U., Automatic Transformation of an Entity-Relationship Query Language into SQL, in *Proc. of the 8th Int. Conf. on ERA*, North-Holland, 1989

[19] Jajodia, S., Ng, P., Translation of Entity-Relationship Diagrams into Relational Structures, *Journal of System and Softwar*e, Vol. 4, 1984

[20] Laender, Casanova, Carvalho, Ridolfi, An Analysis of SQL Integrity Constraints from an Entity-Relationship Model Perspective, *Information System Journal*, Vol. 19, No. 4, 1994

[21] Markowitz, V., M., Shoshani, A., Abbreviated Query Interpretation in Extended Entity-Relationship Oriented Databases, in *Proc. of the 8th Int. Conf. on ERA*, 1989, North-Holland, 1990

[22] Rosenthal, A., Reiner, D., Tools and Transformations - Rigourous and Otherwise - for Practical Database Design, *ACM TODS*, Vol. 19, No. 2, 1994

[23] Smith, J., M., Smith, D., C., P., Database Abstractions: Aggregation and Generalization, *ACM TODS*, Vol. 2, No. 2, 1977
[24] Teorey, T. J., *Database Modeling and Design: the Fundamental Principles*, Morgan Kaufman, 1994
[25] Tucherman, L., Casanova, M., A., Gualandi, P., M., Braga, A., P., A Proposal for Formalizing and Extending the Generalization and Subset Abstractions in the Entity-Relationship Model, in *Proc. of the 8th Int. Conf. on ERA*, North-Holland, 1989
[26] Wagner, C., W., Implementing Abstraction Hierarchies, in *Proc. of the 7th Int. Conf. on ERA*, 1988, North-Holland, 1989