Specification preservation in schema transformations - Application to semantics and statistics

Jean-Luc Hainaut
Institut d'Informatique - University of Namur, rue Grandgagnage, 21 - B-5000 Namur (Belgium)
jlhainaut@info.fundp.ac.be

Abstract

Software design can be modeled as a sequence of transformations applied on initial specifications. In the database domain too, most engineering processes can be described as schema transformations. First, this paper presents a wide spectrum specification model intended to describe data structures at different abstraction levels, and according to the current modelling paradigms. This model includes the representation of statistical data about the instances of the data structures. Then, it defines and discusses the concept of schema transformation. The properties of specification preservation is defined, and applied to two important aspects, namely semantics preservation and statistics propagation. Finally, the impact of the transformational approach on CASE tools is discussed and illustrated by some aspects of DB-MAIN, a representative CASE tool based on this approach.

Keywords. database engineering, wide spectrum data model, schema transformation, statistics modelling, CASE tools

1. Introduction

Modelling software design as the systematic transformation of formal specifications into efficient programs, and building CASE tools that support it, has long been considered as one of the ultimate goals of research in software engineering. For instance, [1] and [10] estimate that *the process of developing a program [can be] formalized as a set of correctness-preserving transformations [...] aimed to compilable and efficient program production*. In this context, according to [28], *a transformation is a relation between two program schemes P and P'* (*a program scheme is the* [parameterized] *representation of a class of related programs*; *a program of this class is obtained by instantiating the scheme parameters*). *It is said to be correct if a certain semantic relation holds between P and P'*.

These definitions still hold for database schemas, which are special kinds of abstract program schemes. The concept of transformation is particularly attractive in this realm, though it has not often been made explicit (for instance as a user tool) in current CASE tools. A (schema) transformation is most generally considered as an operator by which a data structure S1 is replaced by another structure S2 which has some sort of equivalence with S1. Transformations that are proved to preserve the correctness of the origin specifications have been proposed in practically all the activities related to schema engineering : schema normalization [29], DBMS schema translation [17] [31] [32], schema integration [2], schema equivalence [6] [24] [25] [26], data conversion [27], reverse engineering [4] [5] [17] [18] [20], schema optimization [17] [23] and others. The reader will find in [22] an illustration of numerous application domains of schema transformations.

Most authors describe schema transformations as restructuration operators which modify pure data structures. However, the specifications of an information system include other aspects as well, many of them being more or less related to data structures. For instance, several modelling paradigms offer concepts to specify the behaviour of the system, and its interaction with the environment. Moreover, most CASE tools augment the expressive power of the basic models, such the ER or NIAM models, with additional facets such as complex integrity constraints, textual description, visualisation options, data population sizes and other statistics, usage profile and frequencies, ownership and authorisation schemes, etc.

The object of this paper is threefold. First it describes a general purpose data structure model which is to describe most current data structure modelling paradigms at different levels of abstraction, and which includes an original submodel intended to describe the statistics related to the data populations (*Section 2*). Secondly, the paper analyses in some detail the concept of schema transformation (*Section 3*). Thirdly, it studies the problem of specification preservation in schema transformation, specially as far as the semantics (*Section 4*) and the statistics (*Section 5*) are concerned. Finally, some comments are proposed regarding the impact of transformational techniques on CASE technology (*Section 6*).

2. A general purpose data structure specification model

2.1. Motivation

Developing general techniques requires a general framework in which these techniques can be defined and analyzed. Since transformation techniques are to be applied at, and between, different abstraction levels (conceptual, logical and physical), and according to different paradigms, this framework ideally should encompass the concepts, structures and rules making up these levels and paradigms. We have defined the Generic Entity Relationship (GER) model with this goal in mind. GER is a wide spectrum model which can be specialized into most current entity- and object-based models, such as all the variants of the entity-relationship, object-oriented, and operational DBMS models (e.g. relational, CODASYL, IMS, TOTAL, standard files) [16].

The GER model has been given two expressions, namely the concrete view and the abstract view. The *concrete view* is intended to the users of the model, i.e. designers, analysts and developers, and can use, as we will do in this paper, some ER diagrammatic conventions, while the *abstract view* is an extended N1NF model in which all the GER constructs are given a uniform relational interpretation. The abstract view, which has been defined in [13], is mainly intended as a formal basis to specify specific concrete models, to compare them, to define transformations rigorously, and to demonstrate their properties, such as reversibility and constraints propagation. In this section, we will illustrate the main GER constructs at different abstraction levels according to the abstract view (left) and the concrete view (right). Then, we will develop a statistical submodel as an extension to the GER model.

2.2. Conceptual data structures

Interpreted at the conceptual level, a GER schema mainly specifies *entity* (or object) *types*, *relationship types* and *attributes*. To be consistent with the state of the art in conceptual modelling, the model also includes such advanced constructs as *is-a* hierarchies, with *total* and *disjoint* properties, *multivalued* and *compound* attributes, *roles* with cardinality constraints, entity and relationship *identifiers,* attribute *domains* (including entity domains to represent object structures). Some of these constructs are illustrated in *Fig. 1*.

The abstract view

```
PERSON,ORDER,PRODUCT : entities
CUSTOMER,SUPPLIER : PERSON
desc-of-PERSON(PERSON,PID,NAME,
               ADDRESS(NUM[0-1],
               STREET,CITY),PHONE[1-5])
desc-of-CUST(CUSTOMER,CATEGORY)
desc-of-SUPPLIER(SUPPLIER,ACCOUNT)
desc-of-ORDER(ORDER,ONUM,DATE)
desc-of-PRODUCT(PRODUCT,PNUM,PNAME,PRICE)
from(CUSTOMER,ORDER)
DETAIL(ORDER,PRODUCT,SUPPLIER,QTY)
desc-of-ORDER*from: CUSTOMER,ONUM ——→
ORDER
desc-of-PERSON[PERSON] = PERSON
desc-of-CUST[CUSTOMER] = CUSTOMER
desc-of-SUPPLIER[SUPPLIER] = SUPPLIER
desc-of-ORDER[ORDER] = ORDER
desc-of-PRODUCT[PRODUCT] = PRODUCT
from[ORDER] = ORDER
DETAIL[ORDER] = ORDER
```

The concrete view

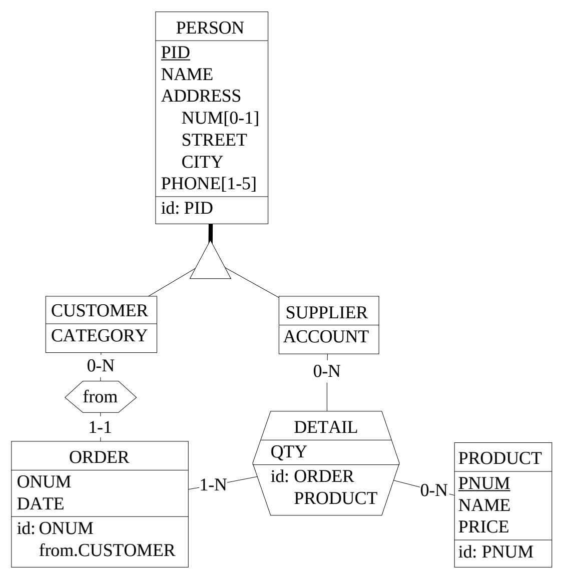


Fig. 1. Abstract and concrete view of a conceptual schema.

In the abstract view, an entity type is declared as an entity domain defined either as a subset of the basic domain `entities`, or as a subset (if it is a subtype) of another entity domain. The attributes of entity type `E`, if any, are declared by a relation called `desc-of-E`, whose relational attributes are as follows : the first attribute is defined on entity domain `E`, and all the others specify the attributes of entity type `E`. Multivalued and optional attributes are defined by cardinality constraints [min-max]. A relationship type is represented by a relation in which each role appears as an entity attribute defined on the corresponding entity domain, and each attribute is represented in the same way as for entity types. The domains of the attributes have not been shown in *Fig. 1* for simplicity sake; for instance, it is understood that the domain of attribute `PERSON` is the entity domain `PERSON`, and that the domain of `NAME` could be, say, `char(32)`. The representation of some major constraints is worth mentioning. First, the identifiers (aka *keys*) are underlined. Secondly, an entity domain can be submitted to existence constraints, stating that each domain element must appear in the current instance of a specified relation; for instance, `from[ORDER]=ORDER` means that each `ORDER` entity must appear in at least one tuple of the instance of `from`. Thirdly, constraints can be defined on derived constructs. For instance, the fact that all the orders of a customer have distinct `ONUM` values is specified by a functional dependency holding in a join : `desc-of-ORDER*from: CUSTOMER,ONUM` ——→ `ORDER`. In the concrete view, this constraint translates into the identifier of `ORDER`.

2.3. Logical data structures

The GER is to represent logical structures as well. For instance, *Fig. 2* can be considered as the relational equivalent of the conceptual schema of *Fig. 1*. Similarly, one could have designed a CODASYL or IMS schema. At this level, the GER structures are interpreted in terms specific to the chosen DBMS model : a logical entity type represents a record type, an object class, a segment type or a table, a logical attribute represents a field or a column, a relationship type represents a set type or a parent-child relationship, and so on.

The abstract view

```
PERSON,CUSTOMER,SUPPLIER,PHONE : entities
ORDER,DETAIL,PRODUCT : entities
desc-of-PERSON(PERSON,PID,NAME,
               ADD_NUM[0-1],ADD-STREET,
               ADD_CITY))
desc-of-CUST(CUSTOMER,PID,CATEGORY)
desc-of-SUPP(SUPPLIER,PID,ACCOUNT)
desc-of-ORDER(ORDER,CUST_PID,ONUM,DATE)
desc-of-PROD(PRODUCT,PNUM,PNAME,PRICE)
desc-of-PHONE(PHONE,PID,PHONE)
desc-of-DETAIL(DETAIL,CUST_PID,ONOM,PNUM,
               SUP_PID,QTY)
desc-of-PERSON[PERSON] = PERSON
desc-of-CUST[CUSTOMER] = CUSTOMER
desc-of-SUPP[SUPPLIER] = SUPPLIER
desc-of-ORDER[ORDER] = ORDER
desc-of-PROD[PRODUCT] = PRODUCT
desc-of-PHONE[PHONE] = PHONE
desc-of-DETAIL(DETAIL] = DETAIL
desc-of-CUST[PID] ⊆ desc-of-PERSON[PID]
desc-of-SUPP[PID] ⊆ desc-of-PERSON[PID]
desc-of-PHONE[PID] = desc-of-PERSON[PID]
desc-of-ORDER[CUST_PID] ⊆
            desc-of-CUST[PID]
desc-of-DETAIL[CUST_PID,ONUM] =
            desc-of-ORDER[CUST_PID,ONUM]
desc-of-DETAIL[SUP_PID] ⊆
            desc-of-SUPP[PID]
desc-of-DETAIL[PNUM] ⊆ desc-of-PROD[PNUM]
```

The concrete view

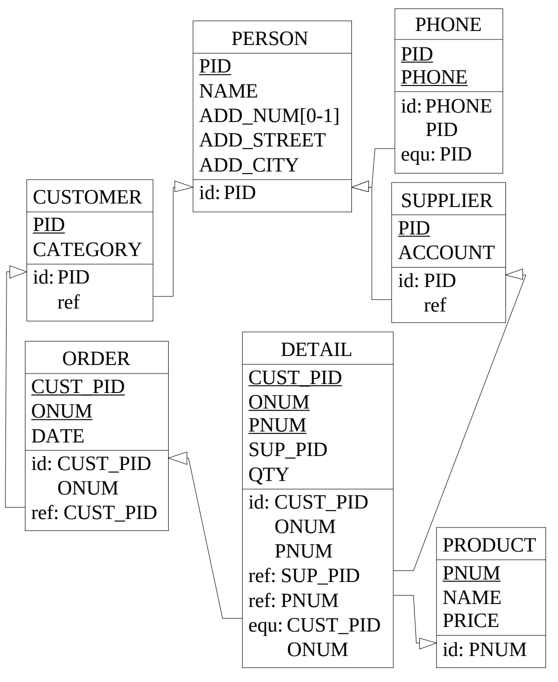


Fig. 2. Abstract and concrete view of a logical relational schema

New constructs appear at this level, such as special kinds of multivalued attributes (bag, list, or array), foreign keys and redundancy constraints. A foreign key is defined by an inclusion constraint, such as in `desc-of-DETAIL[PNUM] ⊆ desc-of-PROD[PNUM]`. When an inclusion constraint also holds in the other direction, both are expressed as an equality constraint, such as in `desc-of-PHONE[PID] = desc-of-PERSON[PID]`. In the concrete view, a foreign key is declared by the `ref` keyword, and by an arc toward the referenced identifier. The `equ` keyword designates an equality constraint.

2.4. Physical data structures

Finally, physical constructs can be specified. *Fig. 3* illustrates two of them. The concept of (physical) entity *collection* can be used as an abstraction of record repositories such as files, table-spaces, data sets. It is defined by the *collect(*ion*)-of* clause in the abstract view, and by a cylinder symbol in the concrete view. The concept of *access key* specifies directed access mechanisms such as indices, hash organisations, access paths, indexed sets, etc. It is defined by the keyword `access-key` in the abstract view, abbreviated to `acc` in the concrete view.

The abstract view

```
CUSTOMER,SUPPLIER : entities
ORDER,DETAIL,PRODUCT : entities
desc-of-CUST(CUSTOMER,PID,CATEGORY)
desc-of-SUPP(SUPPLIER,PID,ACCOUNT)
desc-of-ORDER(ORDER,CUST_PID,ONUM,DATE)
desc-of-PROD(PRODUCT,PNUM,PNAME,PRICE)
desc-of-
DETAIL(DETAIL,CUST_PID,ONOM,PNUM,
               SUP_PID,QTY)
desc-of-CUST[CUSTOMER] = CUSTOMER
desc-of-SUPP[SUPPLIER] = SUPPLIER
desc-of-ORDER[ORDER] = ORDER
desc-of-PROD[PRODUCT] = PRODUCT
desc-of-DETAIL(DETAIL] = DETAIL
desc-of-ORDER[CUST_PID] ⊆
            desc-of-CUST[PID]
desc-of-DETAIL[CUST_PID,ONUM] ⊆
            desc-of-ORDER[CUST_PID,ONUM]
desc-of-DETAIL[SUP_PID] ⊆
            desc-of-SUPP[PID]
desc-of-DETAIL[PNUM] ⊆
            desc-of-PROD[PNUM]
access-key(CUSTOMER):{PID}
access-key(ORDER):{CUST_PID,ONUM}
access-key(DETAIL):{CUST_PID,ONUM,PNUM}
access-key(DETAIL):{SUP_PID}
access-key(DETAIL):{PNUM}
access-key(SUPPLIER):{PID}
access-key(PRODUCT):{PNUM}
access-key(PRODUCT):{NAME}
CUST-FILE:collect-of(CUSTOMER,ORDER,
                     DETAIL)
PROD-FILE:collect-of(SUPPLIER,PRODUCT)
```

The concrete view

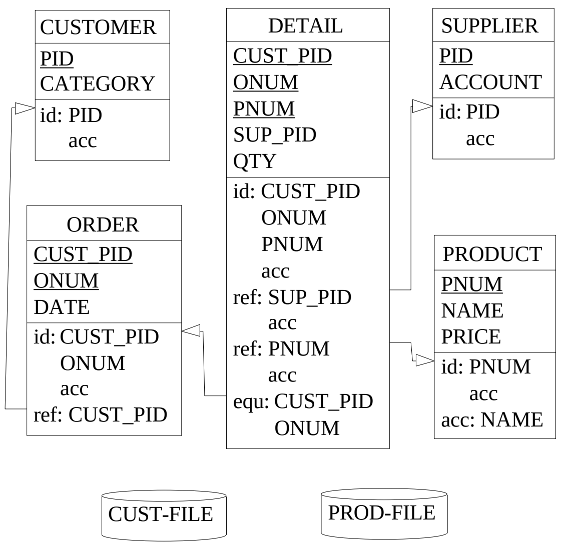


Fig. 3. Abstract and concrete view of a physical relational schema (partial)

2.4. Statistical properties

The statistical model allows the specification of statistics on future or existing data, structured according to the GER model. It describes the size of entity, relationship, attribute and domain populations and of their relations. As such it is an extension of the GER model .
The statistical description of the data has several important applications in database engineering. Let's mention some of them :

- computing the volume of a future database, and its evolution; this application requires some additional information on the physical length of attribute values, on the representation of null values, on physical data storage and on the physical organisation of indices for instance.
- estimating the cost of critical queries and applications; knowing the access strategies of the applications, it is fairly easy to compute the number of logical accesses (number of entities processed); these results can be converted into physical accesses if physical storage parameters are known.
- query optimisation; finding optimal access strategies cannot be based on the knowledge of the logical schema only; knowledge on statistics on the data is also essential.

- logical/physical design; choosing the optimal data structures *w.r.t.* the performance of a set of critical applications needs two analytical tools : generating equivalent schemas [17] [22], and evaluating the cost of the applications [11]. The best schema minimizes the total cost [11].

Though the statistics that will be proposed are often strongly related, we shall present them classified according to the main data object type they describe. In many cases, the statistics are not constants, but are rather time-dependent quantities. In this section, we will consider statistical descriptions related to a reference point of time, i.e. the description of a snapshot of the future or existing data at a given time. Time-dependent evolution of these descriptions have been discussed in [15].

2.4.1 Statistical description of entity types

The most obvious statistics is N_E the average size of the population of entity type E.
Example : $N_{PRODUCT} = 6{,}000$

The statistical relations between entity type A and the m direct subtypes $B_1, B_2, .., B_m$ of A must satisfy the following rules :

$N_{Bi} \leq N_A$ for $i \in [1..m]$ *rel.1*
$N_A \geq \Sigma_i N_{Bi}$ for $i \in [1..m]$ if the B_i's are disjoint *rel.2*
$N_A \leq \Sigma_i N_{Bi}$ for $i \in [1..m]$ if the B_i's are total *rel.3*

As a consequence, we have also :

$N_A = \Sigma_i N_{Bi}$ for $i \in [1..m]$ if the B_i's form a partition *rel.4*

2.4.2 Statistical description of relationship types

Let $R(r_1{:}E_1, r_2{:}E_2, ..., r_n{:}E_n)$ be the structure of relationship type R with degree n. The role r_i is taken by entity type E_i and its cardinality properties are m_{ri}-M_{ri}.

Let N_R be the average number of relationships of R. We can define for each role a *participation* probability function Π_r defined as follows :

$\Pi_{ri}(k)$ gives the probability that an E_i entity participates in exactly k instances of R in role r_i. k must satisfy the condition $m_{ri} \leq k \leq M_{ri}$

Building Π_{ri} for each role of each relationship type in a schema is unrealistic in most situations. Therefore, a simpler version will be proposed, in which we keep only two participation statistics, namely $\Pi0_{ri}$, the probability that an entity participates in no R instances and μ_{ri}, the average number of instances in which an entity participates in role r_i. They are defined as follows :

$\Pi0_{ri} = \Pi_{ri}(0)$ *rel.5*
$\mu_{ri} = \Sigma_k \Pi_{ri}(k) \times k$, for $k \in [m_{ri}..M_{ri}]$ *rel.6*

The first statistics N_R derives from the others through the important relation :

$N_{Ei} \times \mu_{ri} = N_R$ for $i \in [1..n]$ *rel.7*

In addition we have the obvious properties :

$m_{ri} \leq \mu_{ri} \leq M_{ri}$ *rel.8*
$m_{ri} > 0 \Rightarrow \Pi0_{ri} = 0$ *rel.9*

We can immediately derive another statistics, the average number of instances in which *participating* entities participate (i.e. excluding those which don't participate in any instance) :

$\mu'_{ri} = \mu_{ri} / (1-\Pi0_{ri})$ *rel.10*

Example : let R be BUYS(CUSTOMER,PRODUCT)
$\mu_{PRODUCT} = 60$
$\Pi0_{PRODUCT} = 0.4$
$\mu'_{PRODUCT} = 60/(1-0.4) = 100$

There is an average of 60 customers per product; 40% of products are not bought; for products that are actually bought, there is an average of 100 customers.

These relations imply some consequences that are important for checking the consistency of statistics, and to ease their definition.

1. Statistics μ_{ri} of role r_i of R can be inferred from statistics μ_{rj} of any other role r_j of R. Indeed, we have :

   $\mu_{ri} = N_{Ej} \times \mu_{rj} / N_{Ei}$ *rel.11*

   This property states that statistics μ_r needs to be measured for one role of R only.

   Example : let R be `LINE(ORDER,PRODUCT,SUPPLIER)`

   $N_{ORDER} = 8,000$
   $N_{PRODUCT} = 6,000$ $\Rightarrow$ $\mu_{PRODUCT} = 8,000 \times 3/6,000 = 4$
   $N_{SUPPLIER} = 200$ $\quad$ $\mu_{SUPPLIER} = 8,000 \times 3/200 = 120$
   $\mu_{ORDER} = 3$

2. Statistics N_E of entity type E can be inferred from statistics μ_r of any relationship type R in which it appears in role r :

   $N_{Ei} = N_R / \mu_{ri} = N_{Ej} \times \mu_{rj} / \mu_{ri}$ *rel.12*

   Example : let R be `SUPPLIES(COMPANY,PRODUCT)`

   $N_{COMPANY} = 600$
   $\mu_{COMPANY} = 40$ $\Rightarrow$ $N_{PRODUCT} = 600 \times 40/12 = 2,000$
   $\mu_{PRODUCT} = 12$

3. According to rel.8, we have :

   $m_{ri} = M_{ri}$ _ $\mu_{ri} = m_{ri}$ *rel.13*

   Consequently, if $m_{ri} = M_{ri} = 1$, we have also for any role r_j of R :

   $\mu_{rj} = N_{Ei} / N_{Ej}$ *rel.14*

   Example : let R be `SENDS(CUSTOMER,`<u>`ORDER`</u>`); SENDS[ORDER]=ORDER`

   $N_{CUSTOMER} = 600$
   $N_{ORDER} = 1,800$ $\Rightarrow$ $\mu_{CUSTOMER} = 1,800 / 600 = 3$

4. In any (recursive) relationship types, all the roles taken by the same entity type have the same μ_r statistics. Indeed,

   $N_E \times \mu_{ri} = N_E \times \mu_{rj}$ *rel.15*

   implies

   $\mu_{ri} = \mu_{rj}$ *rel.16*

   Example : let R be `MOVE(from:STOCK,to:STOCK,PRODUCT)`

   $N_{STOCK} = 80$ $\quad$ $\mu_{from} = 2,000 \times 40/80 = 1,000$
   $N_{PRODUCT} = 2,000$ $\Rightarrow$ $\mu_{to} = 2,000 \times 40/80 = 1,000$
   $\mu_{PRODUCT} = 40$

   In particular, the statistics of any role of a relationship type having a role with M_r=1 (for instance a binary *one-to-many* relationship type) cannot be greater than 1.

   Example : let R be `is-son-of(`<u>`son`</u>`:PERSON,father:PERSON)`
   the statistics of both roles cannot exceed 1. In other words, the fact that a person cannot have more than one father implies that persons cannot have more than one son on average. For instance,

   $\mu_{son} = 0.3$ $\Rightarrow$ $\mu_{father} = 0.3$

   Though it is quite correct, this result is far from intuitive at first glance. Of course, the average number of sons of the persons who do have sons can be more than 1.

5. For any role with cardinality properties `0-1`, we have

$$\Pi 0_{ri} = 1 - \mu_{ri} \qquad \textit{rel.17}$$

2.4.3 Statistical description of attributes

Let us consider attribute A of parent object E with cardinality properties $[m_A - M_A]$ and domain D.

There are two statistics concerning domain D :

N_D the number of values in D.
λ_D the average length of the values in D, in any convenient unit.

The first statistics of attribute A are

N_A the number of distinct domain values that appear in A.
λ_A the average length of the values of A.

The other statistics describe and quantify the connection between A values and instances of E,

$\mu_{A/E}$,	that specifies the average number of A values corresponding to an instance of E.
$\mu_{E/A}$,	that specifies the average number of E instances corresponding to a value of A.
$\Pi 0_{A/E}$,	that specifies the probability that an E instance has no A values.

Some immediate properties :

$$N_A \leq N_D \qquad \textit{rel.18}$$
$$m_A \leq \mu_{A/E} \leq M_A \qquad \textit{rel.19}$$
$$m_A > 0 \quad \Rightarrow \quad \Pi 0_{A/E} = 0 \qquad \textit{rel.20}$$
$$N_E \times \mu_{A/E} = N_A \times \mu_{E/A} \qquad \textit{rel.21}$$

if A values have fixed-length l, $\lambda_A = \lambda_D = l$ *rel.22*

We can derive two secondary statistics

- the average number of A values for E instances that actually have A values (i.e. excluding those which don't have A values) :

$$\mu'_{A/E} = \mu_{A/E} \,/\, (1 - \Pi 0_{A/E}) \qquad \textit{rel.23}$$

- the average length of A values for E instances that actually have A values :

$$\lambda'_A = \lambda_A \,/\, (1 - \Pi 0_{A/E}) \qquad \textit{rel.24}$$

Example : let E be entity type `EMPLOYEE` with attribute `PHONE[0-5]`; let's use abbreviation E for `EMPLOYEE` and P for `PHONE`;

$N_E = 1{,}200$	$\mu_{P/E} = 1.5$
$N_P = 900$	$\mu_{E/P} = 2$
$\Pi 0_{P/E} = 0.2$	$\mu'_{P/E} = 1.5/(1-0.2) = 1.875$

There is an average of 1.5 telephones per employee and 2 employees per telephone; 20% of the employees have no telephones; employees who do have telephones have an average of 1.875 telephones.

By similarity with the statistical description of relationship types, the following relations can be derived immediately from the definitions given above.

1. Inference of statistics $\mu_{A/E}$ or $\mu_{A/E}$:

$$\mu_{A/E} = \mu_{E/A} \times N_A \,/\, N_E \qquad \textit{rel.25}$$
$$\mu_{E/A} = \mu_{A/E} \times N_E \,/\, N_A \qquad \textit{rel.26}$$

2. Inference of statistics N_A :

$$N_A = \mu_{A/E} \times N_E \,/\, \mu_{E/A} \qquad \textit{rel.27}$$

3. Inference from rel.19 :

$$m_A = M_A \;\Rightarrow\; \mu_{A/E} = m_A \qquad \textit{rel.28}$$

Consequently, if $m_A = M_A = 1$, we have also :

$\mu_{E/A} = N_E / N_A$ *rel.29*

Remarks.

1. It can be noted that the description of attributes seems asymmetrical when compared with that of (binary) relationship types. Indeed, there is no such statistics as $\Pi 0_{E/A}$, stating the probability that an A value corresponds to no E entities. Should it exist, it would be 0 in any case. On the contrary, this notion is pertinent for domain values, since they do not mandatorily appear as A value. However, it has not been included in the model. If needed, it can be computed as $1-N_A/N_D$.
2. Another basic statistical function can be thought of as lacking in the model, namely the distribution function according to the actual values of attribute A (or more generally domain D of A). It would give the probability that an E entity has a given value v as attribute A. It could be defined as,

   $\Pi V_A(v)$, for $v \in D$

   This function has been discarded to keep the model simple. However, it is used in specific problems such as query optimization in some commercial RDBMS.

2.4.4 Statistical description of groups

Let us call a *group* any collection of attributes and/or roles which defines an identifier, a foreign key, an access key, etc. A group G is associated to a parent object E, which is an entity type, a relationship type or an attribute. The importance of statistical description of an access key is obvious, but the following principles are valid for any group, whatever its function.

The proposed statistics are :

N_G, the average number of distinct values of G that appear in the database,

μ_G, the average number of E instances corresponding to a value of G.

Let's first consider a group G made of *more than one component*. We have the property :

$0 \leq N_G \leq N_E$ *rel.30*

if group G is an identifier, we have :

$N_G = N_E$ *rel.31*

$\mu_G = 1$ *rel.32*

If G *includes one component only* (which is supposed to be an attribute A), we have :

$N_G = N_A$ *rel.33*

$\mu_G = \mu_{E/A}$ *rel.34*

Since relation rel.21 of statistical descriptions of attributes is still valid :

$N_E \times \mu_{A/E} = N_G \times \mu_G$ *rel.35*

if $m_A = M_A = 1$, we have also :

$\mu_G = N_E / N_G$ *rel.36*

if G is an identifier, $\mu_G = 1$, so that, *rel.37*

$N_G = \mu_{A/E} \times N_E$ *rel.38*

2.4.5 Statistical description of collections

A collection S is described by

$\mu_{E/S}$, that specifies the average number of E entities that are contained in collection S,

In a consistent (e.g. completed) physical schema, the population of each entity type is completely assigned to at least one collection. Therefore, considering that E has been associated with collections $S_1, S_2, .., S_p$, we have

$N_E = \Sigma_i \mu_{E/Si}$ for $i \in [1..p]$ *rel.39* however, in an inconsistent schema, we have

$$N_E \geq \Sigma_i \mu_{E/Si} \quad \text{for } i \in [1..p] \qquad \textit{rel.40}$$

2.4.6 Summary

The basic statistics that make up the proposed model are illustrated in *Fig. 4*.

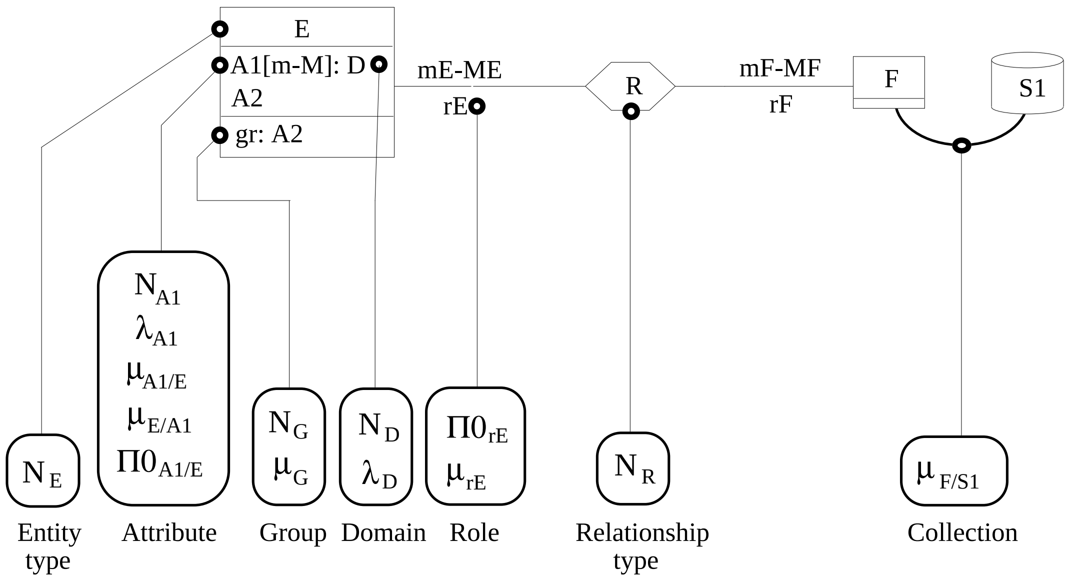


Fig. 4. The components of the statistical model - Summary.

2.4.7 Extensions of the statistical model

Until now the statistical specifications of a schema appear as a set of statistics, a set of inter-statistics relations and a set of constants assigned to the statistics. From an algebraic point of view, the specifications appear as a system of variables (the statistics) and equations (inter-statistics relations) together with one solution (statistics values).

Practically speaking, things appear differently. Since some variables can be derived from the others, the problem is to distinguish the *independent* or ***basic** variables,* to which we must assign values, from the *dependent* or ***derived*** variables, the values of which will be computed. For instance, $N_R = N_{Ei} \times \mu_{ri}$ suggests to calculate N_R from the N_{Ei} and μ_{ri} of an arbitrary role, and therefore to consider N_{Ei} and μ_{ri} as basic variables and N_R as a derived variables.

Unfortunately, an inter-statistics relation is basically *undirected*, i.e. it can be used to define any of its variables as a function of the other ones. For instance, the expression above can be seen also as a way to calculate the μ_{ri} statistics from N_{Ei} and N_R or even to calculate all the μ_{ri} of R from one of them.

Deciding which are the basic statistics and which are the derived ones results in a system of computable specifications. A given set of undirected specifications can generate a fairly large set of computable systems. Finding them, and more specifically choosing one of them is not a straightforward task. The problem is twofold : (1) how to determine basic and derived statistics of a given schema and (2) how to determine the order in which the derived statistics must be computed. A procedure to find computable specifications is proposed in [15].

3. Schema transformation

In this section we will define the concept of schema transformation as a pair of mappings, and we will propose an adequate notation.

3.1. Definition

At first glance, a schema transformation is an operator T that replaces a source construct C in schema S with construct C', leading to schema S'. C' is the target of source construct C through T, i.e. C' = T(C). Before further developing this first definition, we will consider a widespread example (*Fig. 5*), namely replacing a binary relationship type, say PURCH, with an entity type and two one-to-many relationship types.

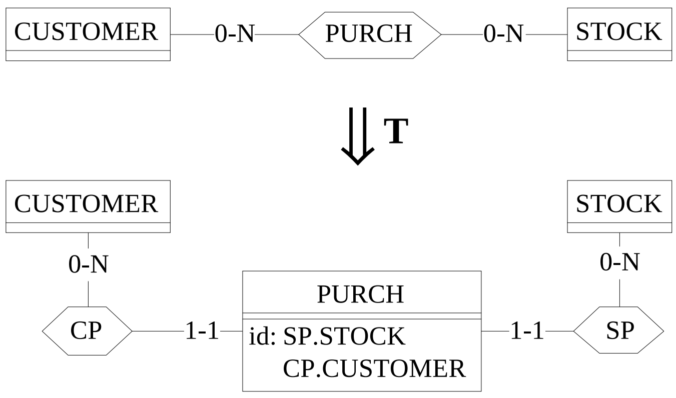


Fig. 5. A popular transformation : transforming a relationship type into an entity type.

We feel that these schemas are *equivalent*, i.e. they express the same *semantics*. If this is true, T can be called *semantics-preserving*. However we are so far unable to prove this fact.

According to this scheme, two extreme situations can be defined, in which either C or C' are empty. If C is empty, the transformation consists in adding C' to S ($S' = S \cup C'$), and is called *semantics-augmenting*, while if C' is empty, the transformation consists in deleting C from S ($S' = S - C$), and is called *semantics-decreasing*. Though they can easily be integrated in this discussion (see [3] for instance), we will concentrate on semantics-preserving techniques only.

Let us observe that a transformation such as that of *Fig. 5* describes the mapping between data types, but it tells us nothing about the relation between the underlying data instances, be they actual or fictive. Considering the example in *Fig. 5*, we can assert without great risk that *each PURCH relationship is represented by a PURCH entity, and conversely*. However, many other instance mappings can be imagined, such as the following : *whatever the instance of relationship type PURCH, the instance of entity type PURCH is made empty*, or this one : *the first 100 PURCH relationships are represented by PURCH entities, and the other ones are ignored*. These interpretations may seem counterintuitive, to say the least, but they fully comply with the source and target data types. Obviously, specifying a transformation requires specifying not only inter-schema (**T**) but inter-instance (**t**) relations as well, as depicted in *Fig. 6*.

$$\begin{tikzcd}
C \arrow[r, "T"] & C'=T(C) \\
c \arrow[u, "\text{instance-of}"] \arrow[r, "t"] & c'=t(c) \arrow[u, "\text{instance-of}"']
\end{tikzcd}$$

Fig. 6. Structural (T) and instance (t) mappings.

T is the structural mapping, and represents the syntax of the transformation, while **t** is the instance mapping which conveys the semantics of the transformation. A transformation Σ is completely defined by its mappings :

$$\Sigma = <T,t>.$$

Expression of mapping T

Several ways to define mapping T have been proposed. One of them consists in specifying the sequence of insert/delete/update operations that produces the target schema from the source one : *remove rel-type PURCH, insert entity type PURCH, insert rel-type CP, insert rel-type SP, insert identifier of PURCH consisting of ...* [28]. Though intuitive, this procedural approach provides a weak support for formal processing.

The predicative approach consists in defining T as a pair of predicates <P,Q>, where P is the minimal precondition, stating how C must look like in order to be a valid candidate, and Q the maximal postcondition, stating how C' will look like when transformed, in such a way that, for any construct C :

$$P(C) \Rightarrow Q(T(C))$$

Hence the alternative notation :

$$\Sigma = <P,Q,t>$$

P and Q are second-order predicates that assert the existence and the properties of schema constructs, and that identify each of their components. Instead of using a pure predicate notation, we will use equivalent, but more intuitive formalisms, such as the abstract or concrete GER notations. *Fig. 7* represents the abstract GER expression of the P and Q predicates of the transformation of *Fig. 5*.

P	CUSTOMER,STOCK:entities
	PURCH(CUSTOMER,STOCK)
Q	CUSTOMER,STOCK,PURCH:entities
	CP(CUSTOMER,PURCH)
	SP(STOCK,PURCH)
	CP[PURCH]=PURCH
	SP[PURCH]=PURCH
	CP*SP:CUSTOMER,STOCK ⟶ PURCH

Fig. 7. Predicative expression of the structural mapping of the transformation of *Fig. 5*.

Expression of mapping t

The instance mapping can be expressed through any query language, such as some extension of relational algebra, as will be illustrated later on. For example, the transformation discussed so far can be given an instance mapping which expresses procedurally the fact that each entity PURCH in the target database denotes a relationship PURCH in the source database (*Fig. 8*).

t	for each p = (c,s) of the current instance of PURCH do generate arbitrary entity p' of PURCH insert (p',c) in the current instance of CP insert (p',s) in the current instance of SP

Fig. 8. Procedural expression of the instance mapping of the transformation of *Fig. 5*.

Transformation scheme vs *transformation*

Before going into further detail, we still need to put forward the fact that some expressions can describe one transformation, while others describe a class of transformations. The example developed in this section belongs to a class of transformations, which could be defined as in *Fig. 9*. Such a definition can be called a generic transformation, or a transformation scheme.

P	E1,E2:entities R(E1,E2)

Q	E1,E2,R:entities
	R1(E1,$\underline{R}$)
	R2(E2,$\underline{R}$)
	R1[R]=R
	R2[R]=R
	R1*R2:E1,E2 $\longrightarrow$ R

t	for each r = (e1,e2) of the current instance of R do generate arbitrary entity r' of R insert (r',e1) in the current instance of R1 insert (r',e2) in the current instance of R2

Fig. 9. The transformation scheme from which the transformation of *Fig. 7* and *8* has been instantiated.

In this transformation scheme, the symbols E1, E2, R, R1, R2 are scheme parameters which must be instantiated, i.e. replaced by actual names in order to get an actual transformation. This transformation in *Fig. 9* is called generic, while that of *Fig. 7* and *8* is one of its instantiations, obtained through the substitutions {E1 $\leftarrow$ CUSTOMER, E2 $\leftarrow$ STOCK, R $\leftarrow$ PURCH, etc}. In fact, this generic transformation, which deals with acyclic binary relationship types, can be seen as a partial instantiation of an even more generic transformation describing how to process N-ary, possibly cyclic, relationship types.

The reader will find in [23], [33], [29] and [7] alternative approaches and notations for transformational techniques.

3.2. Specification preservation

These definitions concern only one aspect of the specifications of an information system, namely the data structures. In addition, they encompass a large class of manipulation operators. In this paper, we are primarily interested in transformations which preserve the specifications in some way. In the next two sections, we will analyse the problem of propagating two important components of information system specifications, namely the semantics, or intention of the data structures, and their statistical description.

4. Semantical aspects of schema transformations

Let us consider that the source schema is given an instance, i.e. a collection of data that are structured according to the constructs of this schema. The problem of semantics preservation can be formulated as follows : under which conditions will these data be preserved when the instance mapping is applied ? To tackle this problem, we will discuss three variants of a very simple transformation based of projecting a relation. Confining this analysis in the classical relational context will simplify the discussion, but its conclusion will easily be generalized to higher-order models.

4.1. Semantics-preservation and reversibility

Let us consider the transformation scheme of *Fig. 10,* called Σ1 in this discussion. It expresses a very broad family of decomposition techniques which can apply on any relation R. The T part (i.e. P and Q) is fairly intuitive. The t part of Σ1 expresses how the instances of the target schema can be computed from the current instance of the source schema.

P	`R(U)`
	`I∪J = U;I ≠ J;I↔J ≠ {}`

Q	`R1(I)`
	`R2(J)`
	`R1[I∩J] = R2[I∩J]`

t	let `r` be the current instance of `R`,
	let `r1` be an instance of `R1`,
	let `r2` be an instance of `R2`,
	`r1 = r[I]`
	`r2 = r[J]`

Fig. 10. A non-reversible, or lossy, transformation (Σ1).

We observe that, once the current source instance `r` of `R` has been transformed into instances `r1` and `r2` of `R1` and `R2`, we can in no way recover it from `r1` and `r2`. Indeed, there are no algebraic, nor procedural operators that could process arbitrary instances of `R1` and `R2` to yield the source instance of `R`. In other words, in general, Σ1 partially destroys the data contents of `R`, in such a way that we can claim that these schemas do not have the same semantics. This transformation is *not semantics-preserving*, or is *not reversible*.

Let us now consider the most popular transformation of the relational theory : the project-join decomposition [8]. One of its versions can be paraphrased as in *Fig. 11*.

P	`R(U)`
	`I∪J∪K = U;I ≠ J ≠ K ≠ I`
	`R:I →→ J|K`

Q	`R1(I,J)`
	`R2(I,K)`

t	let `r` be the current instance of `R`,
	let `r1` be an instance of `R1`,
	let `r2` be an instance of `R2`,
	`r1 = r[I,J]`
	`r2 = r[I,K]`

Fig. 11. A (simply) reversible transformation (Σ1').

The outstanding property of this transformation, called Σ1' from now on, is that the instance of `R` can always be recovered by the natural join of the corresponding instances of `R1` and `R2`. In other words, there exists an inverse transformation, called Σ2', which can *undo* the effect of Σ1'. Its **t** part is clearly : `r = r1*r2`. Σ1' is a data-preserving or *semantics-preserving* transformation. We also can call it *reversible*.

However, transformation Σ2' suffers from an annoying drawback. Indeed, let us suppose that we are given `r1` and `r2`, two arbitrary instances of `R1` and `R2`. We can compute their join : `r = r1*r2`. As could be deduced from the previous discussion, Σ2' seems to have a natural inverse transformation, namely Σ1'. Unfortunately this is wrong, since Σ2' is not semantics-preserving. Indeed, projecting the target instance `r` onto {`I,J`} and {`I,K`} does not yield the source instances `r1` and `r2`, since non-matching rows are lost. All this leads to amazing conclusions : though Σ2' is the inverse of Σ1', Σ1' is **not** the inverse of Σ2'; Σ1' is reversible but Σ2' is **not**.

Finally, we can refine the decomposition principle by adding a derived property to its Q predicate (transformation Σ1'') as shown in *Fig. 12*.

P	`R(U)` $I \cup J \cup K = U; I \neq J \neq K \neq I$ $R: I \rightarrow\rightarrow J \mid K$

Q	`R1(I,J)`
	`R2(I,K)`
	`R1[I]=R2[I]`

t	let `r` be the current instance of `R`,
	let `r1` be an instance of `R1`,
	let `r2` be an instance of `R2`,
	`r1 = r[I,J]`
	`r2 = r[I,K]`

Fig. 12. A symmetrically reversible transformation (Σ1'').

Σ1'' too is reversible, as well as its inverse transformation Σ2'', which reads as in *Fig. 13*.

P2	`R1(I,J)`
	`R2(I,K)`
	`R1[I]=R2[I]`

Q2	`R(U)` $I \cup J \cup K = U; I \neq J \neq K \neq I$ $R: I \rightarrow\rightarrow J \mid K$

t2	let `r1` be the current instance of `R1`, let `r2` be the current instance of `R2`, let `r` be an instance of `R`, `r = r1*r2`

Fig. 13. The reverse of the transformation of *Fig. 12* (Σ2").

Σ1" and Σ2" enjoy a higher-order kind of reversibility, and are said to be *symmetrically reversible*. Σ2" is the inverse of Σ1", Σ1" is the inverse of Σ2", Σ1" and Σ2" are reversible.

More formally, Σ1 = <T1,t1> = <P1,Q1,t1> is **reversible** *iff* there exists an inverse transformation Σ2 = <T2,t2> = <P2,Q2,t2> such that, for any arbitrary construct `C`, and any arbitrary instance `c` of `C` :

```
P1(C) ⇒ [T2(T1(C))=C] and [t2(t1(c))=c]
```

In addition, Σ1 = <T1,t1> = <P1,Q1,t1> is **symmetrically reversible** *iff* Σ1 is reversible and its inverse Σ2 is reversible. In other words,

```
P1(C)  ⇒ [T2(T1(C))=C] and [t2(t1(c))=c]
P2(C') ⇒ [T1(T2(C'))=C'] and [t1(t2(c'))=c']
```

A transformation which is reversible but not symmetrically reversible will be called an *R-transformation*; if it is symmetrically reversible, it will be called an *SR-transformation*. Applying transformation Σ to a construct of schema `S1` will be noted

- `S1` ⇒ `S2` if Σ is an R-transformation,
- `S1` ⇔ `S2` if Σ is an SR-transformation.

Observing that Σ2 = <Q1,P1,t2> leads to the concise notation Σ = <P,Q,t1,t2> for both transformations Σ1 and Σ2.

It is interesting to compare these definitions with some representative quotations :

- *... decomposition transformations which are not only 1-1 but also onto valid instances of the second schema, in such a way that any instance of the latter schema can be obtained by mapping one valid instance of the first schema with* ***f***. [30]
- *... a transformation from one database schema into another is a mapping* ***f*** *from the valid instances (e.g.* ***s****) of the first database schema into valid instances (denoted by* ***f(s)****) of the second one. ... a lossless transformation is a 1-1 mapping, such that* ***f(s)*** *uniquely determines* ***s****.* [9]
- Schemas are *content-equivalent* if *there is an invertible mapping between their possible instantiations* Moreover, *an instance mapping is invertible if it is total, surjective and injective [...]*. [5] [31]
- *Schemas S1 and S2 have the same information content (or are equivalent) if for each query Q that can be expressed on S1, there is a query Q' that can be expressed on S2 giving the same answer, and vice versa.* [2]

4.2. Proving the reversibility of a transformation

It must be kept in mind that the set of practical transformations, i.e. those which concern operational models (ER, OO, SQL, etc) and which are used to support various data engineering processes, can be fairly large. In addition, new techniques can be defined to solve either new problems, or current problems in new ways. Therefore, proving the reversibility, as well as all the other properties, of each of them can be considered intractable in general. Hence the idea to adopt a layered approach in which (1) a small set of general basic techniques are defined on the relational interpretation of the GER, and are proved to be reversible, then (2) these results are translated into the selected operational model. It can be shown that each basic technique can generate a large family of practical transformations, which enjoy the properties of the former. The set of practical transformations can also be enriched by combining simpler techniques into more sophisticated ones.

This approach has been proposed by several authors, which hoped that a very small number of simple basic operators could explain all, or at least most practical transformations. Let us mention some examples. In [12] we proposed 4 generic binary transformations *covering most needs* in model translation; [6] claimed that *all entity-preserving transformations are synthesised* into 6 semantics-preserving graph restructuring operators (the problem is that entity-generating techniques are needed too); [25] proposes 4 generic transformations that *cover most of the useful schema retructuring needs*. Further investigations, including the observation of actual practices, has led to more modest claims. For instance, we currently estimate that a set of some 15 basic SR-transformations should be sufficient to formally specify more than 150 among the most used practical transformations [22].

The benefit of such an approach is twofold. First, it provides us with a simple and easy way to prove the reversibility of most practical transformations. Secondly, it allows us to find in a systematic way new useful techniques. In [14], a single basic transformation, which is proved to be symmetrically reversible, induces at least 24 practical transformations ranging from expressing attributes as entity types, to un-normalizing relationship types.

To illustrate this approach, we will describe two important basic transformations, prove their reversibility, then derive some representative practical transformations from them. Both rely on the project-join SR-transformation Σ1" and Σ2" described in section 4.1.

4.2.1 The Extension-decomposition transformation

Principles

This technique, which has been described in more detail in [14] and [22], is based on the following idea : *a subset (I) of attributes in a relation (R) seems to represent an outstanding concept which would deserve being described by a new surrogate domain (X).*

Let us consider the relational schema $R(U)$ and the arbitrary partition $\{I,J\}$ of U, with $I \neq \{\}$. R can be rewritten $R(I,J)$[1]. We now consider an arbitrary domain X, which is to denote (i.e. be a surrogate for) the sub-tuples $R[I]$ in any instance of R. The denotation function is expressed by the bijection B :

$$B(\underline{X},\underline{I})$$
$$B[I]=R[I]$$

Since the precondition P of Σ2" is satisfied, we can apply this transformation, in order to produce the new schema :

$$R'(X,I,J)$$
$$R':I \longrightarrow X;\ R':X \longrightarrow I$$

If J is empty, R' degenerates into $R'(X,I)$ which is identical to B. If J is not empty, R' is not in BCNF, so that we can apply transformation Σ1" based on FD $X \longrightarrow I$, and we obtain, according to whether J is empty or not,

if J is empty	*if J is not empty*
$T(\underline{X},\underline{I})$	$T(\underline{X},\underline{I})$
	$S(X,J)$
	$S[X] = T[X]$

[1] When I and J are attribute sets, the expression $R(I,J)$ is to be interpreted as $R(I \cup J)$. When X is an attribute, and J a set of attributes, the expression $R(X,J)$ is to be interpreted as $R(\{X\} \cup J)$.

When J is not empty, the following interpretation can be proposed : S is a new version of R where I has been replaced by its surrogate X while T is a translation table where X values are defined in terms of I values. If I is made of more than one attribute, then I can be partitioned into m subsets of attributes : $I = I_1 \cup I_2 \cup .. \cup I_m$. Therefore, R1' can be further decomposed into $\{T_1, T_2, .., T_m\}$ through multiple application of Σ1" :

if J is empty	*if J is not empty*
$T_i(\underline{X}, I_i)$, *i∈[1..m]*	$T_i(\underline{X}, I_i)$, *i∈[1..m]*
$T_1 {*} T_2 {*} .. {*} T_m : I \longrightarrow X$	$S(X,J)$
$T_i[X] = T_j[X]$, *i,j∈[1..m]*	$T_1 {*} T_2 {*} .. {*} T_m : I \longrightarrow X$
	$T_i[X] = S[X]$, *i∈[1..m]*

The final transformation has been defined as a chain of SR-transformations, and therefore is an SR-transformation as well. *Fig. 14* gives the definition of this transformation when *J is not empty*. The case when J is empty can be derived easily.

P	R(I,J) $I = I_1 \cup I_2 \cup .. \cup I_m$; *m ≥ 1* *$I_i$ not empty (i∈[1..m]); J not empty*
Q	$T_i(\underline{X}, I_i)$ *i∈[1..m]* S(X,J) $T_1 {*} T_2 {*} .. {*} T_m : I \longrightarrow X$ $T_i[X] = S[X]$ *i∈[1..m]* *X appears in T_i,S only*
t1	let r be the current instance of R, let t_i be an instance of T_i, i∈ [1..m] let s be an instance of S; choose an arbitrary domain X such that $\lvert X \rvert \geq \lvert R[I] \rvert$ generate arbitrary instances t_i of T_i such that : $(t_1 {*} t_2 {*} .. {*} t_m)[I] = r[I]$ $s = (r {*} t_1 {*} t_2 {*} .. {*} t_m)[X,J]$
t2	let t_i be the current instance of T_i, i∈ [1..m] let s be the current instance of S, let r be an instance of R; $r = (r {*} t_1 {*} t_2 {*} .. {*} t_m)[I,J]$

Fig. 14. The extension-decomposition transformation (*J is not empty*).

By applying the extension-decomposition transformation to various abstract GER expressions, we can generate a large number of practical GER SR-transformations. Here below, we present some of them in a graphical notation. A more in-depth development will be given for the first application.

Application : transforming an attribute into an entity type

Any attribute can be represented by an entity type. Conversely, under definite conditions, an entity type can be transformed into a mere attribute. First, we present a simplified generic example in which attribute A2 is transformed into entity type EA2 according to two different techniques, and we prove that these transformations are symmetrically reversible. The first technique is depicted in *Fig. 15*.

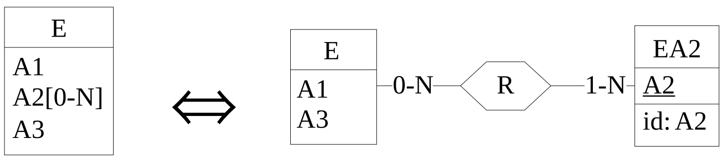


Fig. 15. Transforming an attribute into an entity type by *value representation*.

The abstract GER representation of the left schema is as follows :

```
desc-of-E(E,A1,A2[0-N],A3)
desc-of-E[E] = E
```

First, we get rid of the multivalued attribute by applying the basic `unnest` transformation [22] :

```
desc-of-E(E,A1,A3)
R(E,A2)
desc-of-E[E] = E
```

We can then apply the extension-decomposition transformation to the attribute subset `I={A2}` of `R`. By calling `EA2` the new surrogate domain, we get :

```
desc-of-E(E,A1,A3)
R(E,EA2)
desc-of-EA2(EA2,A2)
desc-of-E[E] = E
R[EA2] = desc-of-EA2[EA2] = EA2
```

The concrete view of this final schema is exactly the right schema of *Fig. 16*. Since we have used basic SR-transformations only, the concrete transformation is symmetrically reversible as well. This transformation is said *by value representation* since each distinct value of `A2` is represented by a distinct `EA2` entity.

This scheme can lead to another version of the transformation in which `I={E,A2}`, and which is said *by instance representation* since each `EA2` entity represents an instance of `A2` :

```
desc-of-E(E,A1,A3)
R(EA2,E)
desc-of-EA2(EA2,A2)
R*desc-of-EA2:E,A2 ⟶ EA2
desc-of-E[E] = E
desc-of-EA2[EA2] = R[EA2] = EA2
```

The concrete view is depicted in *Fig. 16*.

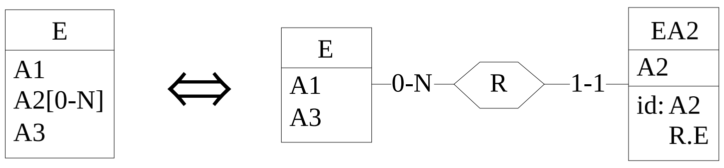


Fig. 16. Transforming an attribute into an entity type by *instance representation*.

Let us apply these transformations to an actual schema. In *Fig. 17*, the multivalued attribute `KEYWORD` is processed through the instance representation technique, while the value representation variant is used for the single-valued attribute `PUBLISHER`.

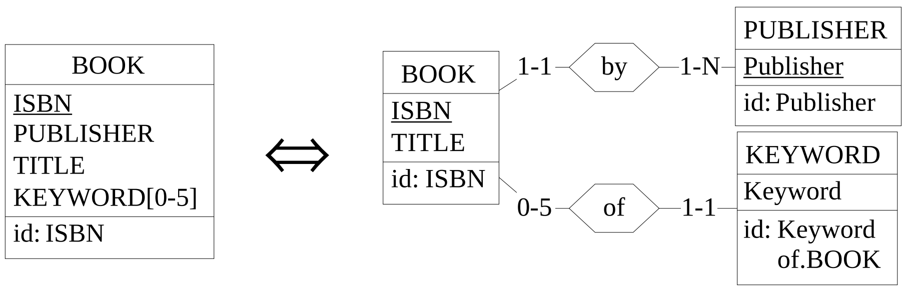


Fig. 17. The attributes PUBLISHER and KEYWORD are transformed into entity types

Application : transforming a relationship type into an entity type

Fig. 18 shows a generalization of the transformation of *Fig. 5* to N-ary relationship types. It is obtained by assigning the surrogate domain `orders` to the set of attributes {`ORDER`,`PART`,`SUPPLIER`} of the relation `orders(`<u>`ORDER,PART,SUPPLIER`</u>`,QTY)`.

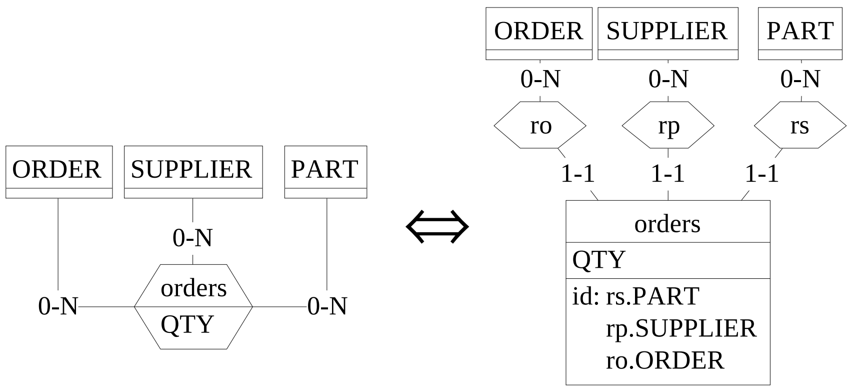


Fig. 18. The components {`ORDER`,`SUPPLIER,PART`} of relationship type `orders` are transformed into entity type `orders`.

Application : transforming components of a relationship type into an entity type

The example in *Fig. 19* illustrates how a generic basic transformation can give birth to new, possibly unusual, practical techniques. This one has been obtained by assigning the surrogate domain `OFFERING` to the subset of attributes {`PART,SUPPLIER`} of the relation `orders(`<u>`ORDER,PART,SUPPLIER`</u>`,QTY)`.

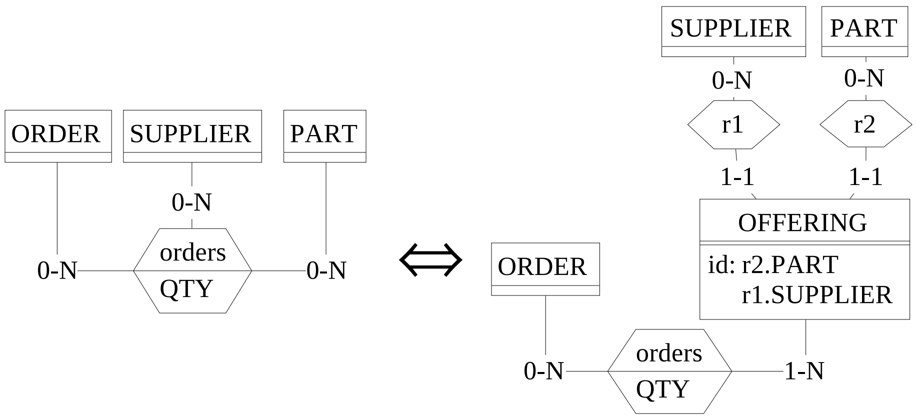


Fig. 19. The components {PART,SUPPLIER} of relationship type orders are transformed into entity type OFFERING

Application : normalizing an entity type
In this application (*Fig. 20*), a subset of attributes is extracted from an entity type to form a new entity type. These attributes are those of a non-key functional dependency. The application of the extension-decomposition transformation is straightforward : R = desc-of-EMPLOYEE (<u>EMPLOYEE</u>,<u>EMP-ID</u>,NAME,...,LOCATION) and I = {DEPARTMENT, LOCATION}.

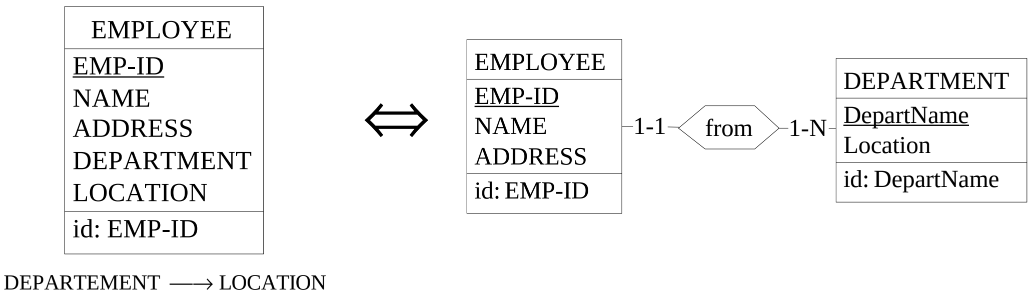


Fig. 20. The attributes of the non-key FD are transformed into entity type DEPARTMENT.

4.2.2 The Composition transformation

Principles
In general, we can not replace two arbitrary relations R and S by one of them, say R, and their composition RoS. This transformation states the conditions under which this substitution can be carried out safely, i.e. without data loss. Let us consider the following schema, in which I, K, L are subsets of attributes :

    R(I,K); S(K,L);
    S[K] ⊆ R[K]

Since we can always partition R instances into R1 = R*S[I,K] and R2 = R-R*S[I,K], we can rewrite this schema as follows :

    R1(I,K); R2(I,K); S(K,L);
    R1[I] ∩ R2[I] = {}; S[K] = R1[K]

Now, R1 and S satisfy predicate P of transformation Σ2", which leads to :

    R2(I,K); RS(I,K,L);

$RS[I] \cap R2[I] = \{\}$; $RS{:}I \longrightarrow K$; $RS{:}K \twoheadrightarrow L|I$

Obviously, RS is not in BCNF, and can be decomposed through Σ1" into $RS1(\underline{I},K)$ and $RS2(I,L)$:

$R2(\underline{I},K)$; $RS1(\underline{I},K)$; $RS2(I,L)$;

$RS1[I]=RS2[I]$; $RS1[I] \cap R2[I] = \{\}$; $RS1*RS2{:}K \twoheadrightarrow L|I$

We observe that RS1 is exactly R1. Therefore, we can replace RS1 and R2 by their former union R. In addition, we rename RS2 as T, so that we get the final schema :

$R(\underline{I},K)$; $T(I,L)$;

$T[I] \subseteq R[I]$

$R*T{:}K \twoheadrightarrow L|I$

When R is bijective in I and K (i.e. $R(\underline{I},\underline{K})$), we get an interesting variant in which the multivalued dependency vanishes. In addition, the demonstration is still valid when R includes non-key attributes. Finally, we can propose the transformation of *Fig. 21*.

P	$R(\underline{I},\underline{K},M)$; *I,K not empty* $S(K,L)$; *K not empty* $S[K] \subseteq R[K]$
Q	$R(\underline{I},\underline{K},M)$; *I,K not empty* $T(I,L)$; *I not empty* $T[I] \subseteq R[I]$
t1	let r be the current instance of R, let s be the current instance of S, let t be an instance of T, t = (r*s)[I,L]
t2	let r be the current instance of R, let t be the current instance of T, let s be an instance of S, s = (r*t)[K,L]

Fig. 21. The attributes of the non-key FD are transformed into entity type DEPARTMENT

Intuitively, this transformation substitutes an identifier for another. We will propose two applications of this principle (see also [17]).

Application : transforming a relationship type into a foreign key

This is probably the most common application of this transformation. It consists in expressing a relationship type as a foreign key. As we proceed here above, we first present a formal example (*Fig. 22*), for which we prove the reversibility property, then we propose an actual application.

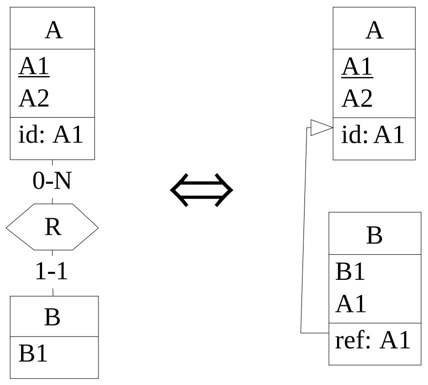


Fig. 22. Relationship type `R` is transformed into the foreign key `A1` in `B` toward `A`

The left schema has the following GER abstract expression :

$\mathtt{desc\text{-}of\text{-}A}(\underline{\mathtt{A}},\underline{\mathtt{A1}},\mathtt{A2})$
$\mathtt{R}(\mathtt{A},\underline{\mathtt{B}})$
$\mathtt{desc\text{-}of\text{-}B}(\underline{\mathtt{B}},\mathtt{B1})$
$\mathtt{desc\text{-}of\text{-}A[A]=A}$
$\mathtt{desc\text{-}of\text{-}B[B]=R[B]=B}$

Applying the composition transformation to `desc-of-A` and `R` yields :

$\mathtt{desc\text{-}of\text{-}A}(\underline{\mathtt{A}},\underline{\mathtt{A1}},\mathtt{A2})$
$\mathtt{R'}(\mathtt{A1},\underline{\mathtt{B}})$
$\mathtt{R'[A1]} \subseteq \mathtt{desc\text{-}of\text{-}A[A1]}$
$\mathtt{desc\text{-}of\text{-}B}(\underline{\mathtt{B}},\mathtt{B1})$
$\mathtt{desc\text{-}of\text{-}A[A]=A}$
$\mathtt{desc\text{-}of\text{-}B[B]=R'[B]=B}$

Now, we can apply the inverse of the project-join transformation (Σ2") to `R'` and `desc-of-B` :

$\mathtt{desc\text{-}of\text{-}A}(\underline{\mathtt{A}},\underline{\mathtt{A1}},\mathtt{A2})$
$\mathtt{desc\text{-}of\text{-}B}(\underline{\mathtt{B}},\mathtt{B1},\mathtt{A1})$
$\mathtt{desc\text{-}of\text{-}B[A1]} \subseteq \mathtt{desc\text{-}of\text{-}A[A1]}$
$\mathtt{desc\text{-}of\text{-}A[A]=A}$
$\mathtt{desc\text{-}of\text{-}B[B]=B}$

This is the abstract view of the proposed target schema.

If `R` is functional, i.e. many-to-one or one-to-one as in the transformation schema above, the foreign key is single-valued, while if `R` is one-to-many or many-to-many, the foreign key is multivalued. *Fig. 23* illustrates the transformation of a many-to-one relationship type into a single-valued foreign key.

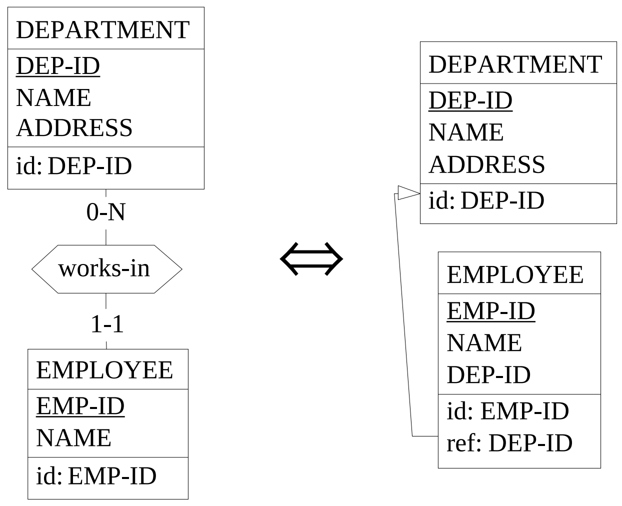


Fig. 23. The many-to-one relationship type `works-in` is transformed into the foreign key `DEP-ID`.

Application : transforming a role of a relationship type
The transformation presented in *Fig. 24* is more complex. It relies, among others, on the composition transformation. It is a nice example of conceptual normalization in which an inclusion constraint is expressed into simpler constructs.

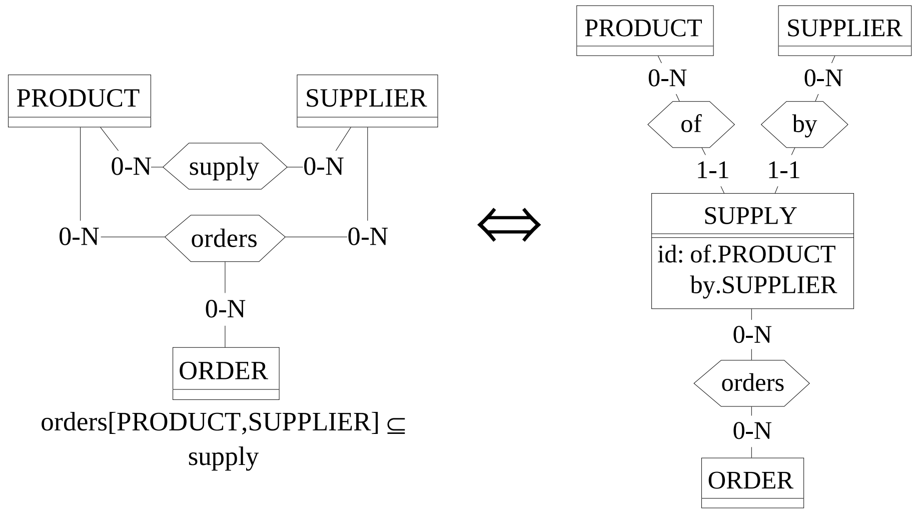


Fig. 24. The inclusion constraint is structurally expressed by replacing the roles {`PRODUCT`,`SUPPLIER`} with the role `SUPPLY`.

4.3. Constraints preservation

One has long admitted than a schema comprises data structures and integrity constraints. Therefore, the structural mapping (T) of a transformation should also cope with the constraints that hold in the source schema. Not only should equivalent constraints be defined in the target schema, but the same degree of reversibility should be ensured as for the data structures. As discussed in [25], constraint propagation is a complex issue for which no general approach exists so far.

Practically speaking, however, things appear somewhat more favourable. Observing that many practical ER transformations can be based on project-join operations (i.e. Σ1" and Σ2"), and considering that the rules governing FD-preserving decomposition schemes have long been stated, both for 1NF and N1NF relations, we can conclude that propagating FD-based constraints, such as identifiers, is a problem which can be treated in the framework of the relational theory. In addition, lost FD can be expressed explicitly as constraints on derived expressions, as illustrated in several of the schemas developed in the previous sections. Some common rules for the propagation of FDs in extension-decomposition transformation are proposed in [14].

5. Statistical aspects of schema transformations

5.1 Introduction

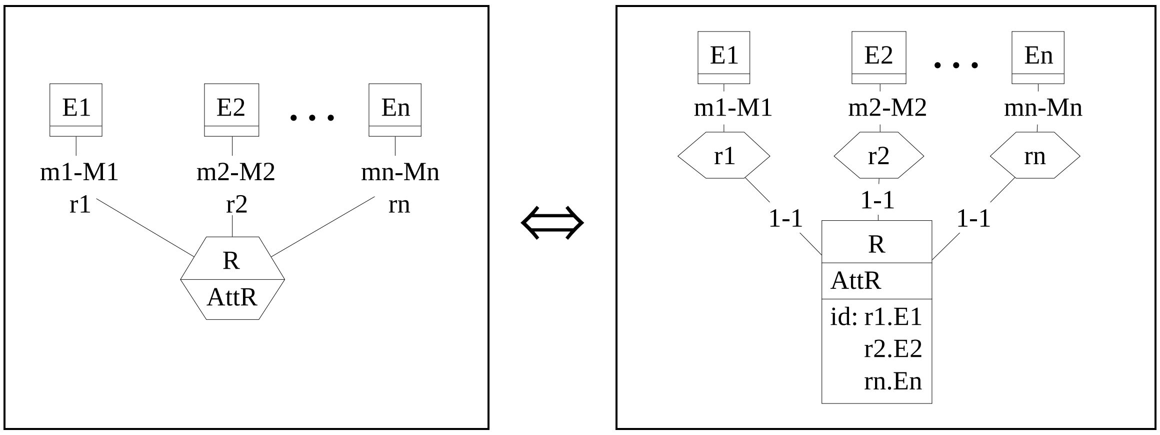
In this section, we will discuss the statistics-preservation properties of schema transformations. The point is of particular importance in the context of forward database engineering since gathering and validating the statistics of a future or existing database is a complex and costly task. It is essential that the statistical specifications in a schema be preserved when this schema is transformed. Anyway, the idea of transforming statistics associated with a conceptual schema into statistics related to the corresponding logical and physical schemas can be questioned. Indeed, statistics are mainly useful for performance evaluation, query optimisation or simulation, i.e. for problems related with logical and physical structures. Therefore, it would be more realistic to collect these statistics for the final, physical schema, and therefore to ignore the problem of transforming statistics. The answer is straightforward :

- collecting statistics at the conceptual level is much easier and more natural since they describe conceptual objects; therefore, users can be involved in this process;
- the conceptual schema is more stable than physical schemas, which may evolve according to changes in the performance requirements and in the state of the technology;
- conceptual schemas, as well as physical schemas, may be transformed for other reasons than pure top-down implementation; view derivation and integration are only two examples.

This section will complement the three most used schema transformations with their statistics translation rules. These rules are also reversible, so that the transformations can be used both ways. It should be noted that the proposed rules are independent of the nature of the statistics, be they constants or temporal functions [15]. For each case, we describe the transformation schema, then we apply it to an example.

5.2. Transforming a relationship type into an entity type

This transformation has been illustrated in *Fig. 18*. It is valid for any kind of relationship type: binary or N-ary, with or without attributes, cyclic or acyclic, many-to-many, one-to-many or one-to-one. *Fig. 25* complements the structural mapping with the statistics transformation rules, expressed in both ways. For each member of the transformation, a complete statistical model is given, comprising basic and derived statistics. Choosing other basic variables would change the model, but not the translation rules. The statistics which degenerate into constants have been omitted (e.g. $\mu_{ri2} = 1$). The statistics (such as N_{Ei}) describing objects involved, but kept, in the transformation are enclosed between parentheses.



$$\boxed{\begin{array}{ll} N_{Ei} & i \in [1..n] \\ \mu_{rj} & \\ N_R = N_{Ej} \times \mu_{rj} & \\ \mu_{ri} = N_R / N_{Ei} & i \in [1..n] \end{array}} \Rightarrow \boxed{\begin{array}{ll} (N_{Ei} & i \in [1..n]) \\ N_{R'} = N_R & \\ \mu_{ri.Ei} = \mu_{ri} & i \in [1..n] \\ N_{Ri} = N_R & i \in [1..n] \end{array}}$$

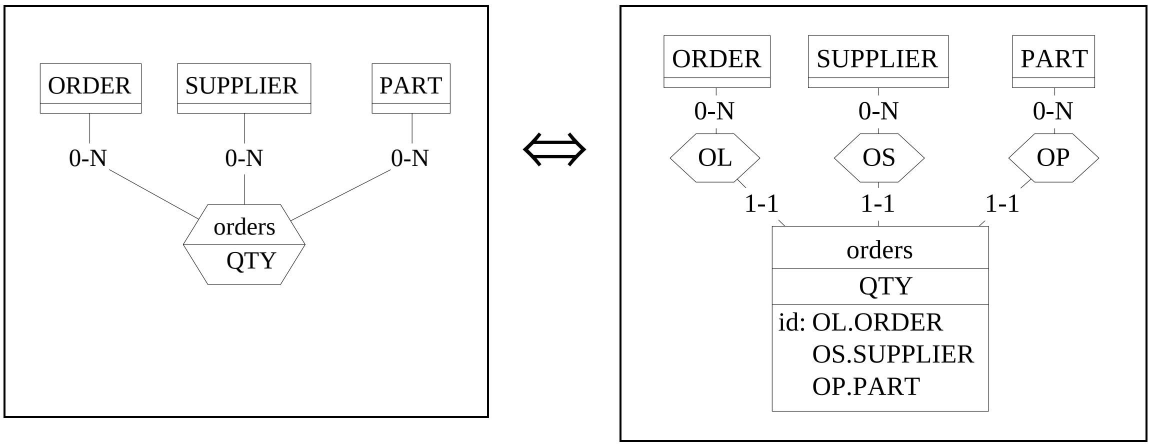
$$\boxed{\begin{array}{ll} (N_{Ei} & i \in [1..n]) \\ N_R = N_{R'} & \\ \mu_{ri} = \mu_{ri.Ei} & i \in [1..n] \end{array}} \Leftarrow \boxed{\begin{array}{ll} N_{Ei} & i \in [1..n] \\ N_{R'} = N_{Ej} \times \mu_{rj.Ej} & \\ \mu_{ri.Ei} = N_{R'} / N_{Ei} & i \in [1..n] \\ N_{Ri} = N_{R'} & i \in [1..n] \end{array}}$$

Fig. 25. Statistics conversion in Relationship-type/Entity-type transformation

Proving the reversibility of these rules by substitution and by application of the laws of the statistical model is straightforward. For instance, let us analyse the translation rules of N_R :

$$N_R = N_{R'} = N_{Ej} \times \mu_{rj.Ej} = N_{Ej} \times \mu_{rj} = N_R$$

Fig. 26 illustrates this transformation scheme with a practical example. In statistics names, some conventions have been adopted to unambiguously designate the roles : for instance, `OS.SUPPLIER` designates the role `SUPPLIER` in relationship type `OS`.



$N_{ORDER} = 2{,}000$
$N_{SUPPLIER} = 300$
$N_{PART} = 30{,}000$
$\mu_{orders.ORDER} = 3$
$N_{orders} = 6{,}000$
$\mu_{orders.SUPPLIER} = 20$
$\mu_{orders.PART} = 0.2$

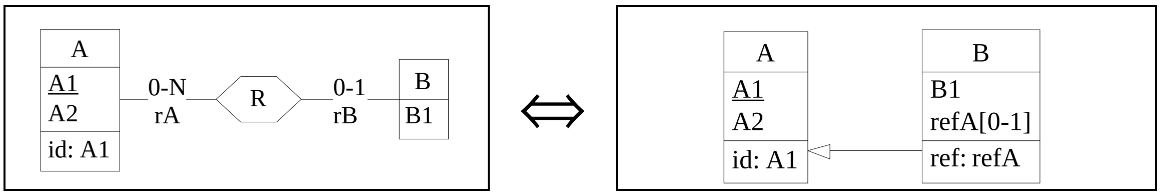
$=$

$N_{ORDER} = 2{,}000$
$N_{SUPPLIER} = 300$
$N_{PART} = 30{,}000$
$N_{orders} = 6{,}000$
$\mu_{OL.ORDER} = 3$
$\mu_{OS.SUPPLIER} = 20$
$\mu_{OP.PART} = 0.2$

Fig. 26. Statistics conversion in Relationship-type/Entity-type transformation - Application

5.3. *Transforming a relationship type into a foreign key*

This transformation has been described in *Fig. 22*. In *Fig. 27*, the rules for statistics translation are given for many-to-one relationship types, which yield single-valued foreign keys.



N_A, N_B
$\mu_{rA}, \Pi0_{rA}$
λ_{A1}
$N_R = N_A \times \mu_{rA}$
$\mu_{rB} = N_R / N_B$
$\Pi0_{rB} = 1 - \mu_{rB}$

$\Rightarrow$

(N_A, N_B)
(λ_{A1})
$\lambda_{refA} = \lambda_{A1} \times (1 - \Pi0_{rB})$
$\mu_{refA/B} = \mu_{rB}$
$\mu_{B/refA} = \mu_{rA} / (1 - \Pi0_{rA})$
$N_{refA} = N_A \times (1 - \Pi0_{rA})$
$\Pi0_{refA/B} = \Pi0_{rB}$

(N_A, N_B)
(λ_{A1})
$\mu_{rA} = \mu_{refA/B} \times N_B / N_A$
$\mu_{rB} = \mu_{refA/B}$
$\Pi0_{rA} = 1 - N_{refA} / N_A$
$\Pi0_{rB} = \Pi0_{refA/B}$
$N_R = N_B \times \mu_{refA/B}$

$\Leftarrow$

N_A, N_B
λ_{A1}
$\mu_{refA/B}$
$\mu_{B/refA}$
$\Pi0_{refA/B} = 1 - \mu_{refA/B}$
$N_{refA} = N_B \times \mu_{refA/B} / \mu_{B/refA}$
$\lambda_{refA} = \lambda_{A1} \times \mu_{refA/B}$

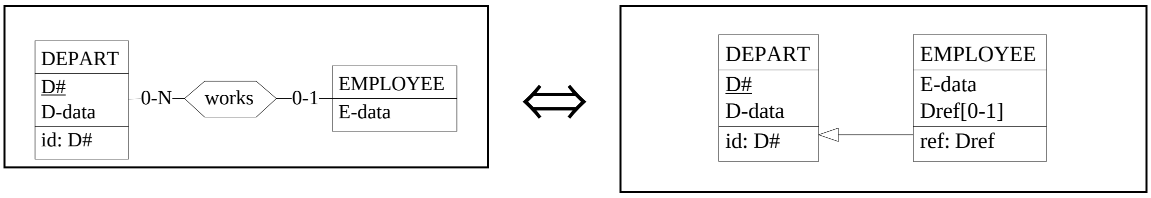
Fig. 27. Statistics conversion in Relationship-type/Foreign-key transformation

This transformation is fully reversible if the cardinality properties of `rA` are `0-N` or `1-N` (or `0-1` or `1-1` of course). The other cases need additional integrity constraints that would make the development more complex without any profit. Let us prove the reversibility of μ_{rA} and $\Pi0_{rA}$:

$$\mu_{rA} = \mu_{refA/B} \times N_B/N_A = \mu_{rB} \times N_B/N_A = N_R/N_B \times N_B/N_A = N_R/N_A = N_A \times \mu_{rA}/N_A = \mu_{rA}$$

$$\Pi0_{rA} = 1 - N_{refA}/N_A = 1 - (N_A \times (1 - \Pi0_{rA}))/N_A = \Pi0_{rA}$$

Fig. 28 illustrates the transformation with a practical example. In statistics names, abbreviations have been used to shorten expressions : `E` stands for `EMPLOYEE` and `D` for `DEPART`.



$N_{D} = 40$
$N_{E} = 2{,}000$
$\mu_{D} = 35$
$\Pi\Theta_{D} = 0.125$
$\lambda_{D\#} = 15$
$N_{R} = 1{,}400$
$\mu_{E} = 0.7$
$\Pi\Theta_{E} = 0.3$

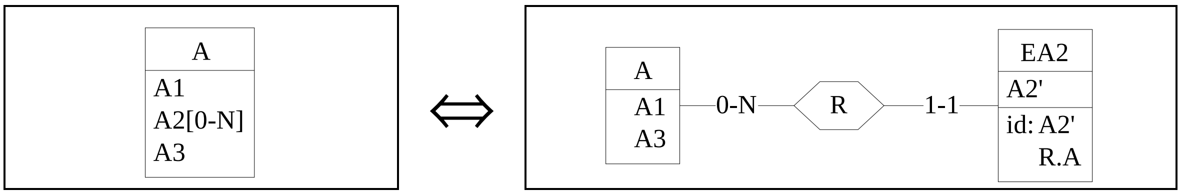
=

$N_{D} = 40$
$N_{E} = 2{,}000$
$\lambda_{D\#} = 15$
$\lambda_{Dref} = 10.5$
$\mu_{Dref/E} = 0.7$
$\mu_{E/Dref} = 40$
$N_{Dref} = 35$
$\Pi\Theta_{Dref/E} = 0.3$

Fig. 28. Statistics conversion in Relationship-type/Foreign-key transformation - Application

5.4. Transforming a attribute into an entity type

We will develop the second variant (*Fig. 29*), called *by instance representation* (*Fig. 16*). The other variant can be processed easily. The statistics which degenerate into constants have been omitted ($\mu_{rEA2}=1$, $\Pi\Theta_{rA2}=0$, etc).



N_{A}
N_{A2}
$\mu_{A2/A}$
$\mu_{A/A2}$
$\Pi\Theta_{A2/A}$
λ_{A2}

$\Rightarrow$

(N_{A})
$N_{EA2} = N_{A} \times \mu_{A2/A}$
$N_{R} = N_{A} \times \mu_{A2/A}$
$\mu_{R.A} = \mu_{A2/A}$
$\Pi\Theta_{R.A} = \Pi\Theta_{A2/A}$
$N_{A2'} = N_{B}$
$\mu_{EA2/A2'} = \mu_{A/A2}$
$\lambda_{A2'} = \lambda_{A2}$

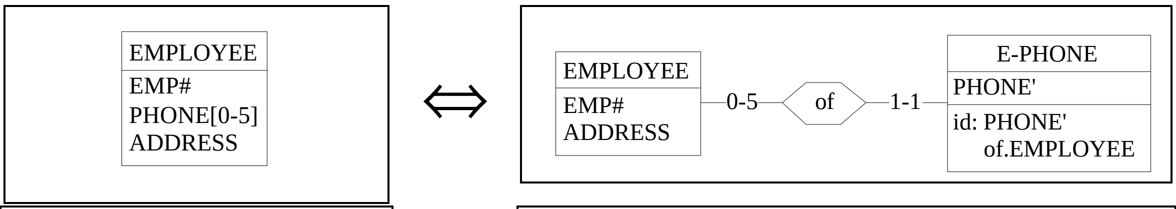
(N_A)
$N_{A2} = N_{A2'}$
$\mu_{A2/A} = N_{EA2}/N_A$
$\mu_{A/A2} = \mu_{EA2/A2'}$
$\Pi\theta_{A2/A} = \Pi\theta_{rA}$
$\lambda_{A2} = \lambda_{A2'}$

$\Leftarrow$

N_A
N_{EA2}
$\Pi\theta_{R.A}$
$N_{A2'}$
$\mu_{EA2/A2'}$
$\lambda_{A2'}$
$N_R = N_{EA2}$
$\mu_{R.A} = N_{EA2}/N_A$

Fig. 29. Statistics conversion in Attribute/Entity-type transformation (instance representation)

Fig. 30 gives a practical example. In statistics names, abbreviations have been used to shorten expressions : `E` stands for `EMPLOYEE`, `P` for `PHONE`, `EP` for `E-PHONE` and `P'` for `PHONE'`.



$N_E = 1{,}200$
$N_P = 900$
$\mu_{P/E} = 1.5$
$\mu_{E/P} = 2$
$\Pi\theta_{P/E} = 0.2$
$\lambda_P = 12$

$=$

$N_E = 1{,}200$ $\quad N_{EP} = 1{,}800$
$\mu_{of.E} = 1.5$ $\quad N_{P'} = 900$
$\Pi\theta_{of.E} = 0.2$ $\quad \mu_{EP/P'} = 2$
$N_{of} = 1{,}800$ $\quad \lambda_{P'} = 12$

Fig. 30. Statistics conversion in Attribute/Entity-type transformation - Application

6. CASE tool support

6.1. CASE tools and transformation-based engineering

Many database engineering CASE tools use, most often implicitly, schema transformations to carry out design processes such as schema normalization, operational schema production or reverse engineering. Indeed, many database design activities can be described as chains of reversible schema transformations. Therefore, automating these operations increases the power and the reliability of the tool. For instance, when producing the relational version of a conceptual schema, the tool can check the precondition (P) and enforce the postcondition (Q) of each transformation in order to garantee that the final schema has the same semantic expressive power as the source schema. However, few of these CASE tools propose transformational operators as an explicit toolbox to be used freely by the developers [16] [21] [32]. In addition, few of them fully exploit the power of the transformations. For instance, the reversibility property can be used as a sound basis for reverse engineering activities, as proposed in [17]. They can also be used to record the engineering activities, in order to build a formal trace which can be processed to support system evolution and maintenance [19].

6.2. The DB-MAIN CASE tool

DB-MAIN[2] is a CASE environment which is to support most of the activities in database engineering, such as analysis and design, normalization, optimization, DBMS schema production, reverse engineering, migration, conversion and maintenance. Its specification model is based on the GER formalism described in section 2. It provides a rich transformational toolset which can be used to support flexible engineering strategies. Since these operators are neutral, they can be used at any level of abstraction and for various paradigms. For example, the transformation of an attribute into an entity type (*Fig. 15* and *16*) can be used to support requirement analysis (e.g. promoting `SUPPLIER-NAME` to entity type `SUPPLIER`), to normalize a conceptual schema or to make it more readable, to optimize a logical schema or to express a secondary access key in CODASYL.

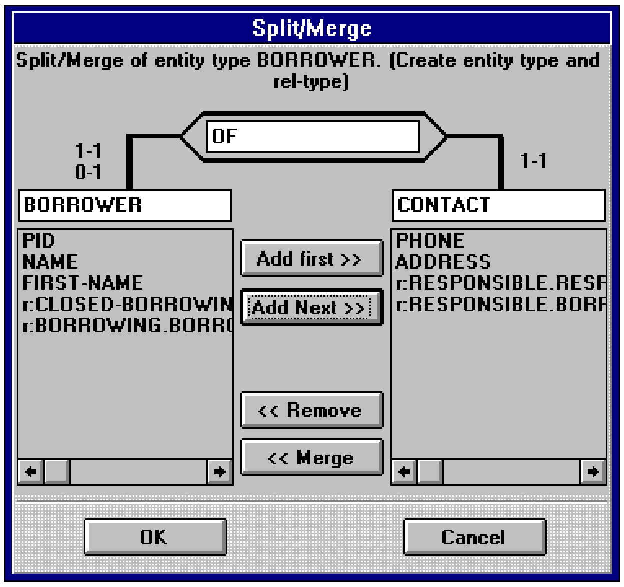
Before examining the transformational aspects of the tool in further detail, let us mention that besides the traditional CASE functions, DB-MAIN also includes powerful processors intended to help users to carry out complex, tedious and knowledge-based activities both in forward and reverse engineering. For instance, a series of assistants are proposed to the users. They supports design processes which require schema evaluation and analysis, schema restructuring, source text analysis and parsing. It provides a repository-based 4GL, called *Voyager 2*, which allows the development of customized functions such as specialized generators, parsers, analyzers, checkers and interfaces with other CASE tools. An in-depth description of the baselines, of the functions and of the architecture of DB-MAIN can be found in [21].

DB-MAIN offers three levels of transformations, namely elementary, filtered and model-driven. The architecture is user-oriented, in that in many cases, several GER transformations has been collected into one practical transformation to give the users an intuitive view of the operations.

6.3. Elementary transformations

An elementary transformation can be defined by $\mathbf{e}(\Sigma, O)$, where Σ is a transformation and O the current object. It consists in applying transformation Σ to object O. *Fig. 31* shows the panel of the Split/Merge transformation which allows to fragment an entity type, to merge two entity types, and to migrate attributes, roles and constraints between two entity types.

[2] This tool is one of the products of the DB-MAIN R&D project. This multinational project is dedicated to Database Application Engineering, and is intended to develop models, methods and tools to support complex processes such as normalization, DBMS translation, optimization, reverse engineering, maintenance and evolution control. The DB-MAIN project, together with its joint projects DB-MAIN/O1 and InterDB, is partially supported by the *Région Wallonne*, the *Communauté Française de Belgique*, the *European Union*, and a consortium of 18 companies and laboratories, and is opened to new partners. An education version of the tool is available for non-profit organizations.


Split/Merge

Split/Merge of entity type BORROWER. (Create entity type and rel-type)

OF

1-1
0-1

1-1

BORROWER

CONTACT

PID
NAME
FIRST-NAME
r:CLOSED-BORROWIN
r:BORROWING.BORR(

PHONE
ADDRESS
r:RESPONSIBLE.RESP
r:RESPONSIBLE.BORF

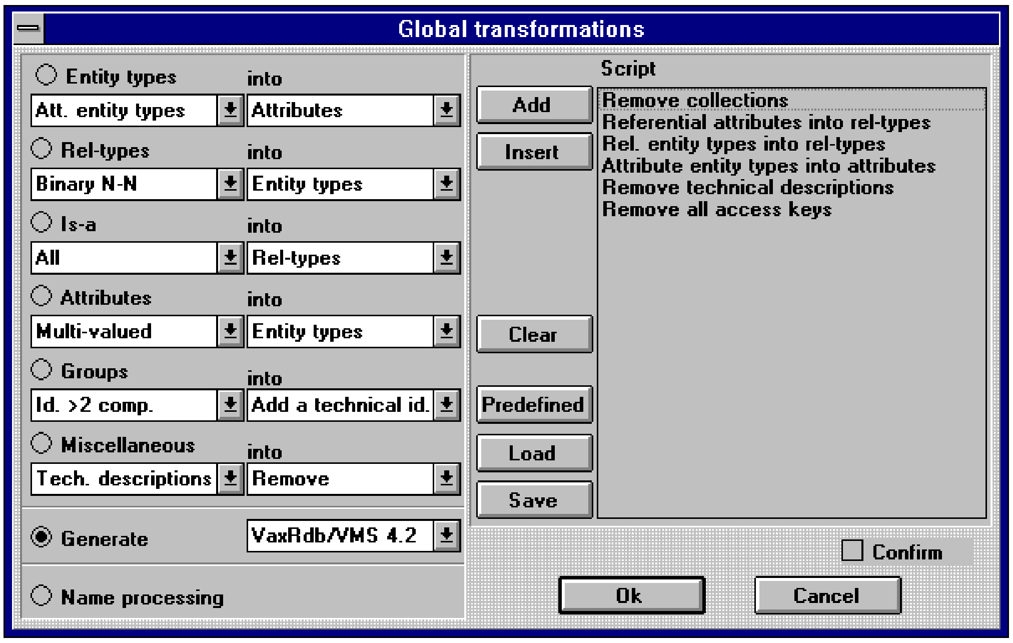
Fig. 31. A user-oriented transformation : split an entity type, merge two entity types, migrate components between entity types

6.4. Filtered transformations

A filtered, or global, transformation is defined by `g(Σ,p)`, where Σ is a transformation and `p` a structural predicate. It consists in applying Σ to all the objects in the current schema which satisfy the predicate `p`. More formally, if Σ = <`P,Q,t`>, the filtered transformation can be defined by `g` = <`p&P,Q,t`>. Such operations are controlled by the *Global Transformations* assistant. This expert processor follows a problem-solving architecture. It proposes a extendable list of standard problems, and for each of them, it suggests a list of transformations which can be used to solve the problem. We mention some representative filtered transformations :

- transform referential attributes [i.e. foreign keys] into relationship types,
- transform cyclic relationship types into entity types,
- transform cyclic relationship types into foreign keys,
- transform N-ary relationship types into entity types,
- transform relationship entity types [i.e. ET which seem to play the role of an RT] into relationship types,
- transform compound attributes into entity types,
- disaggregate compound attributes,
- make an access key with each identifier and each foreign key,
- remove all access keys,
- change the names in the schema according to a given translation table.

The control panel of the assistant, illustrated in *Fig. 32*, includes a column of problems, classified into object types, and a column of suggested transformations or actions. It also includes a script manager.



Entity types into
Att. entity types — Attributes

Rel-types into
Binary N-N — Entity types

Is-a into
All — Rel-types

Attributes into
Multi-valued — Entity types

Groups into
Id. >2 comp. — Add a technical id.

Miscellaneous into
Tech. descriptions — Remove

● Generate — VaxRdb/VMS 4.2

Name processing

Script
Remove collections
Referential attributes into rel-types
Rel. entity types into rel-types
Attribute entity types into attributes
Remove technical descriptions
Remove all access keys

☐ Confirm

Fig. 32. The control panel of the Global Transformation assistant. The problem-solving area is at the left-hand side, while a script manager is on the right half of the panel.

6.5. Model-driven transformations

A model-driven transformation applies on a schema. It can be defined by `m(M)`, where `M` is a submodel, i.e. a subset of the GER defining valid data structures in a specific process of the current methodology. It consists in applying the relevant transformations on the relevant objects in such a way that the final result complies with `M`. A model-based transformation is expressed as a *transformation plan* made up of a sequence of `<condition,action>` statements, where `condition` is a structural predicate and `action` is a transformation. Some of the most used transformation plans are built-in, (relational, CODASYL, COBOL logical design; normalization; reverse engineering; etc) but the user can write its own plans with the Global Transformation assistant (through its scripting facility), or as *Voyager 2* programs.

7. Conclusions

The concept of schema transformation has proved an outstanding aid both at the theoretical and practical levels. As a formal expression of common practices, it provides a rigorous specification and allows reasoning on these operators, for instance to derive properties such as those related to constraint propagation and design specifications preservation. Studying schema transformations with the two-level analytical approach proposed in this paper induces a great economy of concepts. Indeed, only a small number of basic operators need be completely formalized. Since these operators are expressed in some form of N1NF models, i.e. the abstract expression of the GER model, proving their reversibility is fairly easy. Interpreting these basic transformations at the operational level, i.e. as concrete expressions of the GER model, allows the derivation of a very large number of practical operators.

At the practical level, schema transformations can improve the reliability and the scope of Information Systems development methodologies by providing intuitive but rigorous ways of reasoning. For instance, in an educational context, both in the university and in the industry, we have observed that using this paradigm leads to a much better and faster understanding of complex processes such as schema translation, schema optimization, schema normalization, and reverse engineering. In addition, implementing schema transformations as CASE tools functions increases both the reliability and the power of these tools. In every process in which a schema is derived from another one, the tool can garantee that both schemas express the same semantics, particularly in large and complex information systems. The functional interpretation of transformations (through mapping T) also provides a concise notation to record user or automated actions on the specifications. These records form the history of the engineering processes. Due to the formal specification of these actions, this history can be analyzed, exploited and processed. One of the most impressive application of history processing is called *design recovery*, i.e. the reconstruction of a hypothetical, but plausible, design history of a legacy system, the documentation of which has been lost [19].

The second contribution of the paper is the extension of the Entity-relationship model to the statistical description of the data. Though exploiting this information is most often relevant when implementing and optimizing the database and the programs, we have proposed to collect it at the highest level. Indeed, schema transformation can also be extended to cope with this statistical model, and to propagate without loss these specifications.

This statistical model is not comprehensive, and considering the complexity of the problem, will never be so. However, let us mention some problems still to be discussed and some natural extensions.

- correlations between statistics cannot be described; besides the analytical relations between statistics presented in section 2.4, it is impossible to declare real world relationships between statistics. For instance, unmarried persons have not the same statistics about their children than married ones.
- data usage statistics must also be taken into account in order to estimate the access time of queries and application. Update and data usage statistics are strongly linked to the statistical model. In addition, this information provides a precise decision support for physical design (see [11] for instance).
- update statistics must be taken into account as well. In particular, create/delete/modify statistics are strongly linked to the temporal aspects of the statistical model presented here. These statistics can be either specified explicitly, or they can be inferred from knowledge on the access strategies and frequencies of the update queries and applications.
- the model is based on the idea that all the objects of a schema are described by all the statistics of their type. For instance, all attributes are supposed to be associated with the five statistics evoked in *Fig. 4*. This is not realistic. The user (i.e. the designer) must be allowed to decide what are the quantities (s)he needs. The concept of consistent and computable system of statistics must be revised accordingly.

Ensuring the propagation of the statistics model of the data should be an important by-product of computer-aided transformations. The rules controlling this propagation have been implemented into TRAMIS, the previous version of DB-MAIN [16]. Incorporating this function into the current version is under investigation.

Acknowledgements

I would like to thank Jean Henrard, Vincent Englebert, Didier Roland and Jean-Marc Hick, who designed and implemented the DB-MAIN tool.

References

[1] Balzer, R., Transformational implementation : An example, *IEEE TSE*, Vol. SE-7, No. 1, (1981)

[2] Batini, C., Ceri, S., Navathe, S., B., *Conceptual Database Design*, (Benjamin/ Cummings, 1992)

[3] Batini, C., Di Battista, G., Santucci, G., Structuring Primitives for a Dictionary of Entity Relationship Data Schemas, *IEEE TSE*, Vol. 19, No. 4, (1993)

[4] Bolois, G., Robillard, P., Transformations in Reengineering Techniques, *Proc. of the 4th Reengineering Forum "Reengineering in Practice*", Victoria, Canada, (1994)

[5] Casanova, M., A., Amaral De Sa, Mapping uninterpreted Schemes into Entity-Relationship diagrams : two applications to conceptual schema design, *IBM J. Res. & Develop*., Vol. 28, No 1, January, (1984)

[6] D'Atri, A., Sacca, D., Equivalence and Mapping of Database Schemes, *Proc. 10th VLDB conf*., Singapore, (1984)

[7] De Troyer, O., *On data schema transformation*, PhD Thesis, University of Tilburg, Tilburg, The Netherlands (1993)

[8] Fagin, R., Multivalued dependencies and a new normal form for relational databases, *ACM TODS*, Vol. 2, N°3, (1977)

[9] Fagin, R., Normal Form for Relational Databases Bases on Domains and Keys, *ACM TODS*, Vol. 6, N°. 3, September (1981)

[10] Fikas, S., F., Automating the transformational development of software, *IEEE TSE*, Vol. SE-11, pp1268-1277, (1985)

[11] Hainaut, J-L., Some Tools for Data Independence in Multilevel Data Base Systems, *Proc of the IFIP TC2/WC on Modelling in Data Base Management Systems*, Nice,(North-Holland, 1977)

[12] Hainaut, J-L., Theoretical and practical tools for data base design, *Proc. of the Very Large Databases Conf*., pp. 216-224, September, (IEEE Computer Society Press,1981)

[13] Hainaut, J.-L., A Generic Entity-Relationship Model, *Proc. of the IFIP WG 8.1 Conf. on Information System Concepts: an in-depth analysis*, (North-Holland, 1989)

[14] Hainaut, J-L., Entity-generating Schema Transformation for Entity-Relationship Models, *Proc. of the 10th Entity-Relationship Approach*, San Mateo (CA), 1991, (North-Holland, 1992)

[15] Hainaut, J-L., A Temporal Statistical Model for Entity-Relationship Schemas, *Proc. of the 11th Conf. on the Entity-Relationship Approach*, Karlsruhe, Oct. 1992, LNCS, (Springer-Verlag, 1992)

[16] Hainaut, J-L., Cadelli, M., Decuyper, B., Marchand, O., Database CASE Tool Architecture : Principles for Flexible Design Strategies, *Proc. of the 4th Int. Conf. on Advanced Information System Engineering* (CAiSE-92), Manchester, May 1992, LNCS, (Springer-Verlag, 1992)

[17] Hainaut, J-L, Chandelon M., Tonneau C., Joris M., Transformation-based database reverse engineering, *Proc. of the 12th Int. Conf. on ER Approach*, Arlington-Dallas, LNCS, (Springer-Verlag, 1994)

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

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

[20] Hainaut, J-L., Englebert, V., Henrard, J., Hick, J-M., Roland, D., *Database Reverse Engineering - Problems, Techniques and CASE tools*, Tutorial notes, CAiSE•95 Conference, Jÿvaskÿla (Finland), (June 1995) <available at jlh@info.fundp.ac.be>

[21] Hainaut, J-L, Englebert, V., Henrard, J., Hick J-M., Roland, D., Requirements for Information System Reverse Engineering Support, *Proc. of the 2nd IEEE WC on Reverse Engineering*, Toronto, (IEEE Computer Society Press, 1995); an extended version has been published as Database Reverse Engineering: from Requirements to CARE tools, *Journal of Automated Software Engineering*, Vol. 3 No. 2 (1996)

[22] Hainaut, J-L., *Transformation-based Database Engineering*, Tutorial notes, 21st VLDB Conf., (Zürich, September 1995) <available at jlh@info.fundp.ac.be>

[23] Halpin, T., A., Proper, H., A., Database schema transformation and optimization, *Proc. of the 14th Int. Conf. on ER/OO Modelling* (ERA), (1995)

[24] Jajodia, S., Ng, P., A., Springsteel, F., N., The problem of Equivalence for Entity-Relationship Diagrams, *IEEE Trans. on Soft. Eng*., SE-9, 5, (1983)

[25] Kobayashi, I., Losslessness and Semantic Correctness of Database Schema Transformation : another look of Schema Equivalence, *Information Systems*, Vol. 11, No 1, pp. 41-59, (1986)

[26] Lien, Y., E., On the equivalence of database models, *JACM*, 29, 2, (1982)

[27] Navathe, S., B., Schema Analysis for Database Restructuring, *ACM TODS*, Vol.5, No.2, (1980)

[28] Partsch, H., Steinbrüggen, R., Program Transformation Systems, *Computing Surveys*, Vol. 15, No. 3, (1983)

[29] Rauh, O., Stickel, E., Standard Transformations for the Normalization of ER Schemata, *Proc. of the CAiSE•95 Conf.*, Jyväskylä, Finland, LNCS, (Springer-Verlag, 1995)

[30] Rissanen, J., Independent components of relations, *ACM TODS*, Vol. 2, N°4, (1977)

[31] Rosenthal, A., Reiner, D., Theoretically sound transformations for Practical Database Design, *Proc. of the 6th Int. Conf. on Entity-Relationship Approach*, March (Ed.), (North-Holland, 1988)

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

[33] Vidal, V., Winslett, M., A Rigorous Approach to Schema Restructuring, *Proc. of the 14th Int. Conf. on ER/OO Modelling* (ERA), (1995)

Jean-Luc Hainaut is a professor in Information Systems and Databases at the Institute of Informatics of the University of Namur, Belgium. Since 1971, his interests comprise database technology, database methodology, reverse engineering, system evolution and CASE technology. He is heading the multinational DB-MAIN R&D project (47 person-year), the purpose of which is to develop general methodologies and CASE tools to assist practitioners in tackling distributed database applications evolution and maintenance problems. He is the author of the books (in French) *Computer Aided Application development - Database Design*, and *Databases and Computing Models - Tools and Methods for Users*, and of over 25 recent papers (11 of which deal with schema transformations). Recently, he gave several tutorials on Database Reverse Engineering and on Transformational Approach to Database Engineering in Entity-Relationship, INFORSID, CAiSE, VLDB and Software Engineering conferences.