FACULTES UNIVERSITAIRES
NOTRE-DAME DE LA PAIX NAMUR

**Institut d'Informatique**
**University of Namur**

# TRANSFORMATION-BASED DATABASE ENGINEERING

Jean-Luc Hainaut
professor in the Institute of Informatics
of the University of Namur

Address : Institut d'Informatique
rue Grandgagnage, 21
B-5000 Namur - Belgium

jlhainaut@info.fundp.ac.be
fax : +32 81/72.49.67

VLDB'95
Zürich (Switzerland) - September 1995

# Foreword

Significant parts of this tutorial derive from the material developed in the TRAMIS, PHENIX and DB-MAIN projects. Therefore, Olivier Marchand and Bernard Decuyper of the TRAMIS project, Catherine Tonneau, Muriel Chandelon and Michel Joris of the PHENIX project, and Didier Roland, Jean-Marc Hick, Jean Henrard and Vincent Englebert of the DB-MAIN project, can be considered as indirect coauthors of this tutorial.

DB-MAIN is a 47 man-year R & D, and Technology Transfer programme, including the projects DB-MAIN/Basic (*DB Applications Evolution*), DB-Process (*Software Process Modeling*), DB-MAIN/Objectif-1 (*Technology Transfer*) and InterDB (*Federated Systems*), TimeStamp (*Temporal databases*). It is supported by :

• the University of Namur

• an open international industrial consortium

ACEC-OSI
ARIANE-II
Banque UCL (Lux)
BBL
Centre de Recherche Public H. Tudor (Lux)
CGER
COCKERILL-SAMBRE
CONCIS (Fr)
D'Ieteren
DIGITAL
EDF (Fr)
Groupe S
IBM
OBLOG Software (Port)
ORIGIN
Ville de Namur
Institut de Criminalistique
WINTERTHUR
3 SUISSES
Euroviews Services
Région Bruxelles Capitale
*others pending*

• La Communauté Française de Belgique (*DB-PROCESS* project)

• The European Union (*DB-MAIN / Objectif-1* project)

• La Région Wallonne (*DB-MAIN / Objectif-1* project)

(*InterDB* project)

(*TimeStamp* project)

# About this tutorial

## Motivation

**Transformation-based** software engineering has long been considered as a major scientific approach to build reliable and efficient programs. According to this approach, abstract specifications can be converted into correct, compilable and efficient programs by applying selected, correctness-preserving operators called *transformations*.

In the database engineering realm, an increasing number of authors recognize the merits of such an approach, that can produce correct, compilable and efficient database structures from conceptual specifications. Transformations that are proved to preserve the correctness of the origin specifications have been proposed in practicaly all the activities related to schema engineering : schema normalization, DBMS schema translation, schema integration, views derivation, schema equivalence, data conversion, reverse engineering, schema optimization and others.

However, most authors propose either informal ad hoc restructuring techniques or, on the contrary, formal techniques that are out of scope of practitioner's competence. Little effort has been made (1) to rigourously define a fairly comprehensive toolset of orthogonal transformations, and (2) to translate these techniques into practical reasonings and tools which can help practitioners.

## The tutorial

The proposed tutorial is a contribution to the systematic study of both theoretical and practical aspects of database schema transformations. The concept of transformation is developed, together with its properties of semantics-preservation (or reversibility). Major database engineering activities are redefined in terms of transformation techniques, and the impact on CASE technology is discussed and demonstrated.

The material of this tutorial is based on a large experience in academic and industrial training programmes, and in methodologies and CASE tools development using the transformational approach. It is also based on the results of three database engineering R & D projects dedicated to database design (TRAMIS), database reverse engineering (PHENIX) and database applications evolution (DB-MAIN).

## The target audience

The tutorial is primarily dedicated to practitioners wishing to get a deeper and more rigourous analysis and development of database engineering activities. However, due to the originality and scope of the approach, it can be followed by researchers as well.

## Warning

Due to time limits, and to the immaturity of some parts of the material, this document must be considered as tentative. It will be revised, augmented and better illustrated. Contact us for further versions.

# Organization

1. INTRODUCTION
   - 1.1 Motivation
   - 1.2 An intuitive example
   - 1.3 Impact on software engineering
   - 1.4 State of the art
   - 1.5 Organization of the tutorial
2. THE CONCEPT OF DATABASE TRANSFORMATION
   - 2.1 Principles
   - 2.2 Generic/instantiated transformation
   - 2.3 Semantics preservation and reversibility
   - 2.4 Constraint propagation
   - 2.5 Non-redundant/redundant transformation
   - 2.6 Notations
3. BASIC TRANSFORMATIONS
   - 3.1 Motivation
   - 3.2 A N1NF relational model
   - 3.3 The Project/Join transformation
   - 3.4 The Denotation transformation
   - 3.5 The Extension transformation
   - 3.6 The Composition transformation
   - 3.7 The Nest/Unnest transformation
   - 3.8 The Aggregation/Disaggregation transformation
   - 3.9 Definitional transformations
4. The GER : a Generic ER/OR Model
   - 4.1 Objectives
   - 4.2 The Domains
   - 4.3 The Entity relation
   - 4.4 The Relationship relation
   - 4.5 The Constraints

# Part 1

## INTRODUCTION

**A transformation is a relation between two program schemes P and P' (***a program scheme is the [parametrized] 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'.**

[KRIEG,89]

**... the process of developing a *program* [can be] formalized as a set of correctness-preserving transformations [...] aimed to compilable and efficient *program* production.**

[BALZER,81], [FICKAS,85]

**the process of developing a *database* [can be] formalized as a set of correctness-preserving transformations [...] aimed to compilable and efficient *database structure* production.**

[anonymous, 20th century]

## Objective of this tutorial

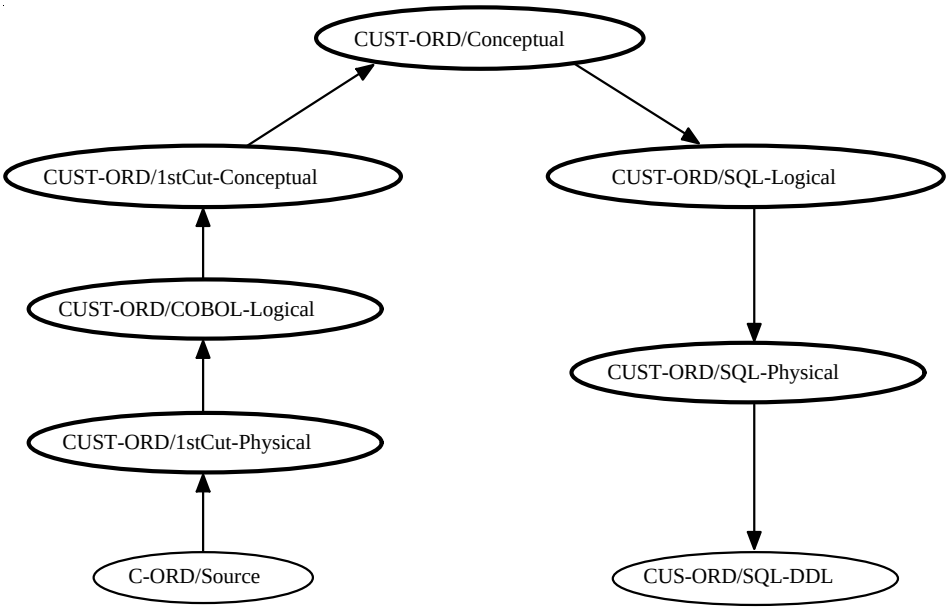
**to explore the applicability of the transformation paradigm in the most important database engineering processes.**

[Hainaut,1995]

## Objective

Converting the files of a COBOL application into relational structures. This example has been drawn from [HAINAUT,95c], Appendix. It has been developed with the DB-MAIN CASE tool (Section 8).

## The scenario (or history)



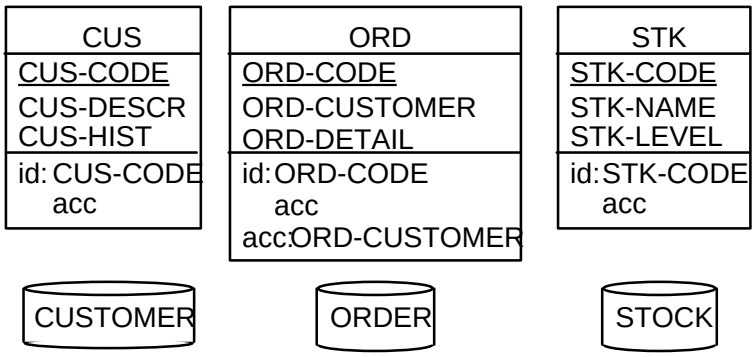
## 1. The source text (excerpts)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. C-ORD.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT CUSTOMER ASSIGN TO "CUSTOMER.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS CUS-CODE.
   SELECT ORDER ASSIGN TO "ORDER.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS ORD-CODE
      ALTERNATE RECORD KEY IS
      ORD-CUSTOMER WITH DUPLICATES.
   SELECT STOCK ASSIGN TO "STOCK.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS STK-CODE.

DATA DIVISION.
FILE SECTION.
FD CUSTOMER.
01 CUS.
   02 CUS-CODE PIC X(12).
   02 CUS-DESCR PIC X(80).
   02 CUS-HIST PIC X(1000).

FD ORDER.
01 ORD.
   02 ORD-CODE PIC 9(10).
   02 ORD-CUSTOMER PIC X(12).
   02 ORD-DETAIL PIC X(200).
```

```
FD STOCK.
01 STK.
   02 STK-CODE PIC 9(5).
   02 STK-NAME PIC X(100).
   02 STK-LEVEL PIC 9(5).

WORKING-STORAGE SECTION.
01 DESCRIPTION.
   02 NAME PIC X(20).
   02 ADDRESS PIC X(40).
   02 FUNCTION PIC X(10).
   02 REC-DATE PIC X(10).

01 LIST-PURCHASE.
   02 PURCH OCCURS 100 TIMES
         INDEXED BY IND.
      03 REF-PURCH-STK PIC 9(5).
      03 TOT PIC 9(5).

etc


PROCEDURE DIVISION.
MAIN.
   PERFORM INIT.
   PERFORM PROCESS UNTIL CHOICE = 0.
   PERFORM CLOSING.
   STOP RUN.

etc
```

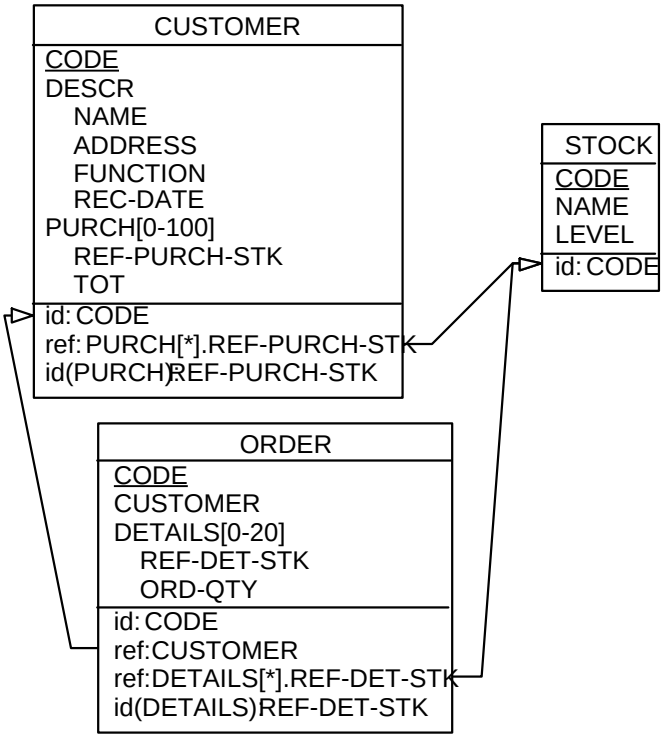
## 2. The abstract COBOL physical schema (first-cut)

This schema is the graphical expression of the file and record data structures explicitly declared in the program.



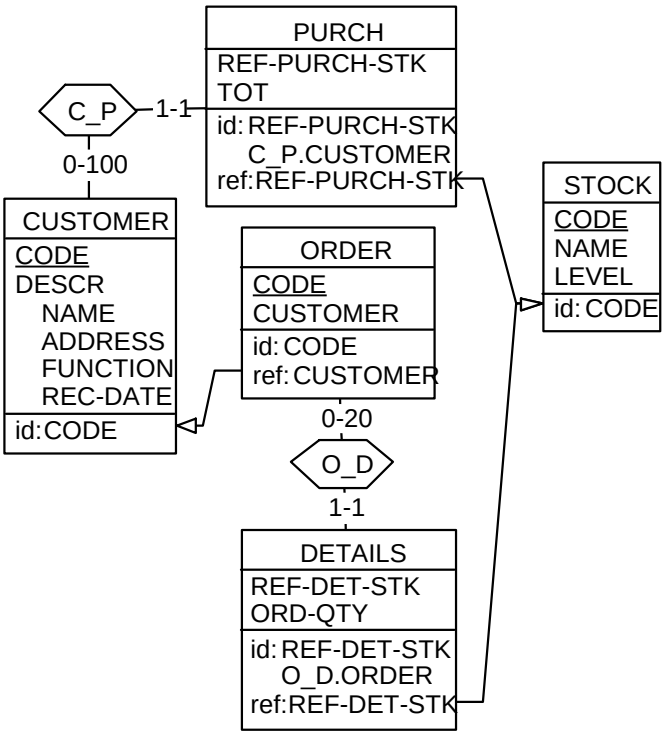
## 3. The COBOL logical schema

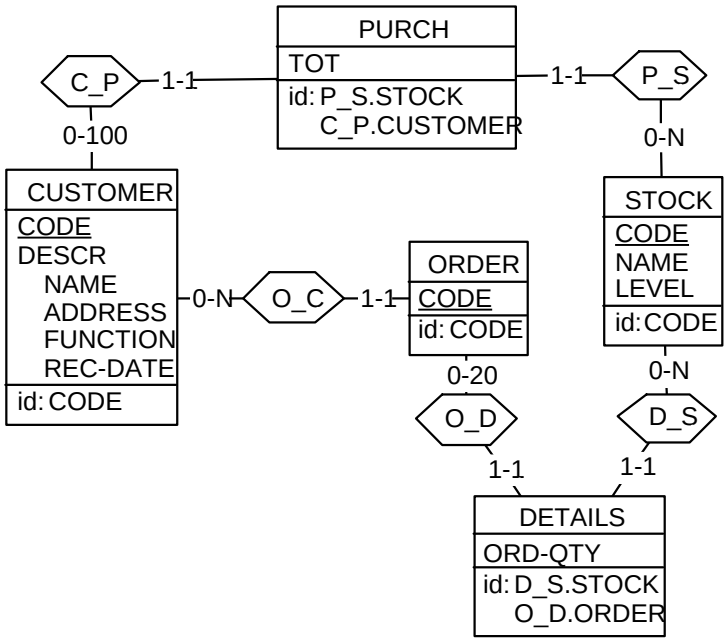
Through, a.o., procedural code and data analysis, the first-cut schema is refined, and a logical version is produced by discarding index and file specifications, and by **transforming** record and field names.



## 4. The conceptual schema (first-cut)

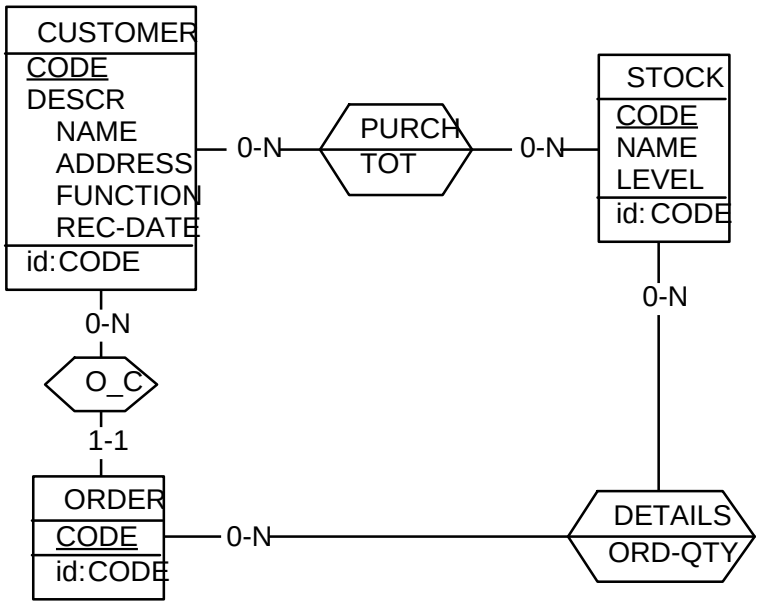
The logical schema is **transformed** step-by-step, to recover the conceptual structures. For instance, the compound-multivalued fields are **transformed** into entity types, and the foreign keys are **transformed** into relationship types.




PURCH
TOT
id: P_S.STOCK
C_P.CUSTOMER
C_P
1-1
1-1
P_S
0-100
0-N
CUSTOMER
CODE
DESCR
NAME
ADDRESS
FUNCTION
REC-DATE
id: CODE
0-N
O_C
1-1
ORDER
CODE
id: CODE
STOCK
CODE
NAME
LEVEL
id: CODE
0-20
0-N
O_D
D_S
1-1
1-1
DETAILS
ORD-QTY
id: D_S.STOCK
O_D.ORDER

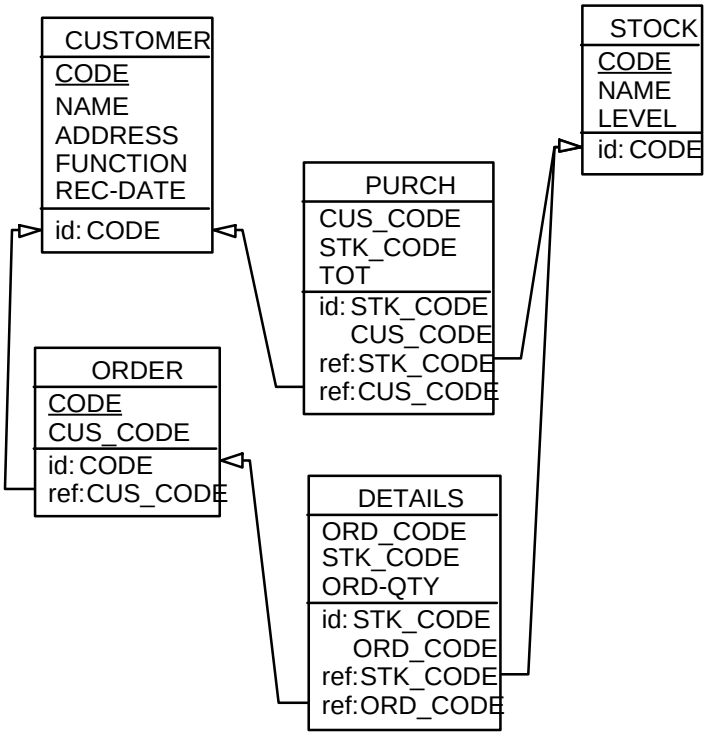
## 4. The conceptual schema (normalized)

This schema is given a cosmetic treatment in order to make it comply with the corporate methodological standards. For instance, some entity types are **transformed** into relationship types.



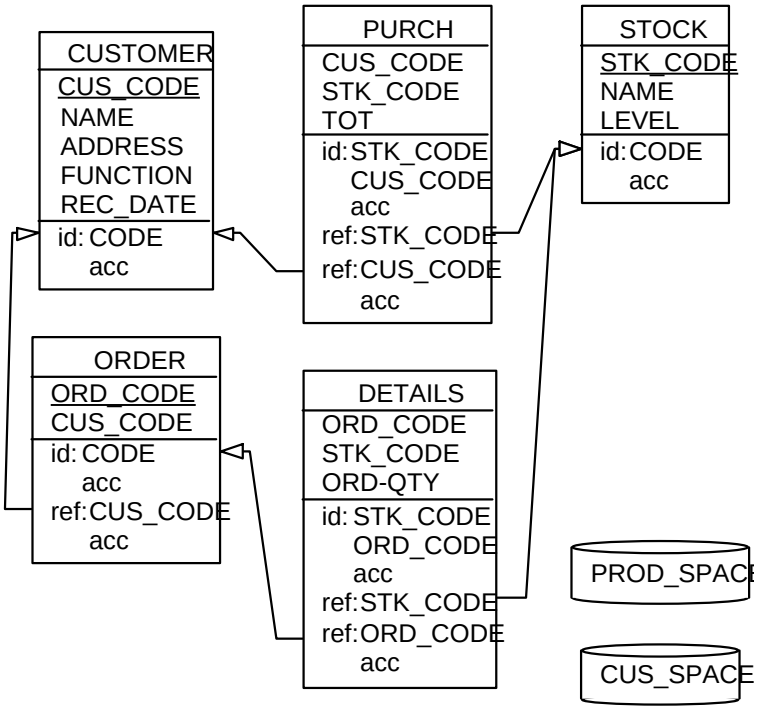
## 5. The SQL logical schema

Producing an SQL-compliant logical schema is fairly straighforward : the complex (not *one-to-many*) relationship types are **transformed** into entity types, then each one-to-many relationship type is **transformed** into a foreign key.



## 6. The SQL physical schema

Physical constructs are added : access keys (*indexes*) and files (*dbspaces*).



## 7. The SQL DDL script

```
create database CUS-ORD;

create dbspace PROD_SPACE;
create dbspace CUS_SPACE;

create table CUSTOMER (
   CUS_CODE char(12) not null,
   NAME char(20) not null,
   ADDRESS char(40) not null,
   FUNCTION char(10) not null,
   REC-DATE char(10) not null,
   primary key (CUS_CODE))
   in CUS_SPACE;

create table DETAILS (
   ORD_CODE numeric(10) not null,
   STK_CODE numeric(5) not null,
   ORD-QTY numeric(5) not null,
   primary key (STK_CODE,
      ORD_CODE))
   in CUS_SPACE;

create table ORDER (
   ORD_CODE numeric(10) not null,
   CUS_CODE char(12) not null,
   primary key (ORD_CODE))
   in CUS_SPACE;

create table PURCH (
   CUS_CODE char(12) not null,
   STK_CODE numeric(5) not null,
   TOT numeric(5) not null,
   primary key (STK_CODE,
       CUS_CODE))
   in CUS_SPACE;

create table STOCK (
   STK_CODE numeric(5) not null,
   NAME char(100) not null,
   LEVEL numeric(5) not null,
   primary key (STK_CODE))
   in PROD_SPACE;
```

```
alter table DETAILS add constraint
FKDET_STO
     foreign key (STK_CODE)
     references STOCK;

alter table DETAILS add constraint
FKDET_ORD
     foreign key (ORD_CODE)
     references ORDER;

alter table ORDER add
constraint FKO_C
     foreign key (CUS_CODE)
     references CUSTOMER;

alter table PURCH add
constraint FKPUR_STO
     foreign key (STK_CODE)
     references STOCK;

alter table PURCH add
constraint FKPUR_CUS
     foreign key (CUS_CODE)
     references CUSTOMER;


create unique index CUS-CODE
 on CUSTOMER (CUS_CODE);

create unique index IDDETAILS
 on DETAILS (STK_CODE,ORD_CODE);

create index FKDET_ORD
 on DETAILS (ORD_CODE);

create unique index ORD-CODE
 on ORDER (ORD_CODE);

create index FKO_C
 on ORDER (CUS_CODE);

create unique index IDPURCH
 on PURCH (STK_CODE,CUS_CODE);

create index FKPUR_CUS
 on PURCH (CUS_CODE);

create unique index STK-CODE
 on STOCK (STK_CODE);
```

- towards more rigorous database engineering techniques
  - preserving the correctness of conceptual specifications
  - increasing the reliability of (and confidence in) operational databases
  - elaborating more formal (though more intuitive) methodologies, and formalizing current ones
- understanding the foundation of data models (power evaluation, comparison, equivalence)
- understanding reverse engineering (as the reverse of forward engineering)
- improving education and training in database engineering
- sound basis for more powerful and more flexible CASE technology

**Explicitly or implicitly used for more than 15 years**

[NAVATHE,80], [FAGIN,81] [HAINAUT,81]

**but emerged as a basic engineering paradigm of its own few years ago only**

[BATINI,92], [BATINI,93], [BERT,85], [DETROYER,93], [HAINAUT,91a], [HAINAUT,94a], [HALPIN,95], [JOHANNESSON,93], [KOBAYASHI,86], [KOZACZYNSKY,87], [NIJSSEN,89], [RAUH,95], [ROSENTHAL,88], [ROSENTHAL,94], [JONER,95], [VIDAL,95]

**its application domain is broadening**

- database development,
- reverse engineering,
- schema equivalence,
- database/schema/view integration,
- normalization,
- optimization,
- etc

**sound basis for rigourous**

- **reasoning** (schema equivalence, model equivalence, traceability)
- **engineering** (normalization, database production, reverse engineering, migration, etc)
- **CASE development** [HAINAUT,92a] [ROSENTHAL,94] [HAINAUT,95b]

**generally considered as ad hoc techniques, but some attempts to formalize are emerging**

e.g. [DETROYER,93], [HAINAUT,91a], [HALPIN,95], [KOBAYASHI,86], [RAUH,95]

**problems still to be solved**

- satisfying definition of the ***semantics-preserving*** property
- complete axiomatization of database transformations (kernel-based ?)
- constraint propagation and preservation
- integration of transformations in problem-solving reasoning; e.g. how to use the predicative definition in model-based transformation plans ?
- definition of a minimal set of practical transformations
- usage in training and education
- user-oriented presentation of transformations (e.g. in CASE tools)
- . . . and more

## The principles

The concept of database transformation is defined as a couple of mappings, one between database schemas (the *syntax*) and the other between database instances (the *semantics*). The properties related to semantics preservation, or reversibility, are studied (section 2). A set of basic techniques that are proved to be reversible are proposed. These techniques can form the kernel of a large family of practical transformations (section 3).

2. THE CONCEPT OF DATABASE TRANSFORMATION

3. BASIC TRANSFORMATIONS

## Practical transformations

From these basic transformations, one derives a structured catalog of some popular practical transformations that are applicable to most current conceptual and operational data models (e.g. ERA, NIAM, relational, OO, network, hierarchical, standard files). They are presented as large-scope, neutral techniques that can be used in many database engineering activities (section 5). They are expressed into a generic model defined as a generalization of the Entity-relationship model (section 4). Extended techniques also are discussed (section 6).

4. The GER : a Generic ER/OR Model

5. ER/OR SR-TRANSFORMATIONS

6. OTHER ER/OR TRANSFORMATIONS

## Methodological applications

The transformations are then used to solve various practical problems that occur in database engineering. Some of the main database engineering processes are discussed and expressed as transformation-based strategies.

7. APPLICATIONS

## CASE tools

Schema transformations form an ideal paradigm through which many design processes can be assisted or automated. The impact of these techniques on CASE technology is examined, in particularly for design traceability. The DB-MAIN prototype CASE tool based on transformation techniques is described and demonstrated.

8. TRANSFORMATIONS and CASE tools

## Case study

The transformation techniques and the DB-MAIN tool are applied to the reengineering of a legacy COBOL system into an SQL application.

9. CASE STUDY : COBOL to SQL schema conversion

## Bibliography (Section 10)

# Part 2

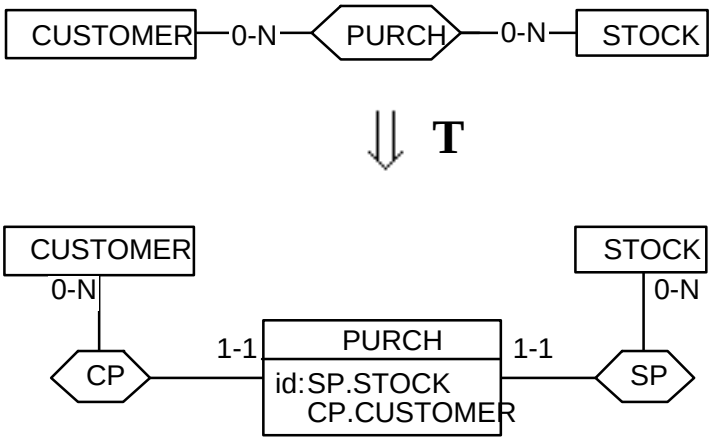
## THE CONCEPT OF DATABASE TRANSFORMATION

## Definition

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 : C' = T(C).

## Example

The relationship type PURCH (C) is transformed into entity type PURCH and rel-types CP and SP (C'). We *feel* that these schemas are *equivalent*., i.e. they have the same *semantics*, but we are unable to prove this (so far).




**Remark** : C or C' can be empty

- if C is empty, the transformation consists in adding C' to S : **S' = S ∪ C'**
- if C' is empty, the transformation consists in deleting C from S : **S' = S – C**

Though they can easily be integrated in this study, we will ignore these forms of transformations (see [BATINI,93] for instance).

## What about the semantics ?

This description specifies the types in each schemas, but it tells us nothing about the relation between the underlying semantics of these schemas. For instance, are they equivalent ? Is one of them more powerful, more general ?

**First approach**

Let us compare the *Universes of Discourse* (the real world objects and relationships) they each describe. Are they the same, do they overlap, is one of them included in the other one ?

Nice concept, but not really operational : how to reason, to build proofs on this basis ?

**Second approach (turn left 180°)**

Let us examine the *data instances* that populate these types at each instant. Are they the same, do they overlap, is one of them included in the other one ? How can we derive each one from the other one ?

Much better : data sets can be compared and manipulated much easier than real world objects.

**Is the question worth being discussed ?**

Of course it is.

Our first intuition about the example transformation is most probably that *each PURCH relationship is represented by a PURCH entity, and conversely*.

However, it is only one among thousands of other valid interpretations. Think of the following :

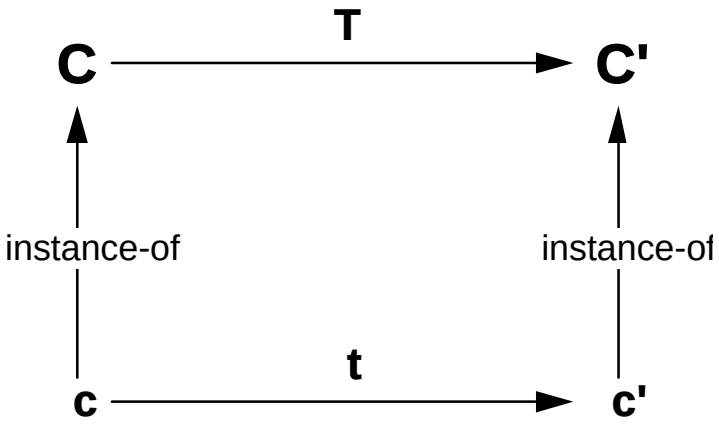
*whatever the instance of relationship type PURCH, the instance of entity type PURCH is made empty.*

or of this one :

*the first 100 PURCH relationships are represented by PURCH entities, and the other ones are ignored.*

Stupid, isn't it ? Stupid but quite valid; nothing prevents us from adopting such interpretations which fully satisfy the structures in source and target schemas.

Obviously, specifying a transformation requires specifying both inter-schema (**T**) and inter-instance (**t**) relations, otherwise, the operator is meaningless, or at least undefined.




**T = structural mapping (**the syntax of the transformation)

$$C' = T(C)$$

**t = instance mapping** (the semantics of the transformation)

$$c' = t(c)$$

**Transformation** $\Sigma = \langle T,t \rangle$

## Specification of a transformation

Let us discuss mappings **T** and **t** a bit further.

**How to define mapping T ?**

- **procedural specification**

  remove rel-type PURCH, insert entity type PURCH, insert rel-type CP, insert rel-type SP, insert identifier of PURCH consisting of ... .

  Intuitive, but weak support for reasoning [PARTSCH,83].

- **predicative specification**
  - minimal precondition P : *how must C look like in order to be a valid candidate ?*
  - maximal postcondition Q : *how will C' look like when the transformed has been carried out ?*

  In fact, P and Q include two kinds of statements : applicability constraints (in P) and properties (in Q) of the source and target schemas + identification of the involved components and structures (see 2.2).

  **Example** :

```
P =    entity-type(CUSTOMER)
     & entity-type(STOCK)
     & rel-type(PURCH)
     & role(PURCH,,CUSTOMER,[0-N])
     & role(PURCH,,STOCK,[0-N])
Q =    entity-type(CUSTOMER)
     & entity-type(STOCK)
     & entity type(PURCH)
     & rel-type(CP)
     & role(CP,,CUSTOMER,[0-N])
     & role(CP,,PURCH,[1-1])
     & rel-type(SP)
     & role(SP,,STOCK,[0-N])
     & role(SP,,PURCH,[1-1])
     & id(PURCH,{SP.STOCK,SP.CUSTOMER})
```

  However, we will most often use more concise and intuitive expressions, such as graphical and DDL.

**How to define mapping t ?**

**t** defines how to translate any instance c of C into instance c' of C'. Through any query or data manipulation language expression (algebra, calculus, procedural language, etc). See 2.6.

**Complete specification of a transformation**

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

**Additional notations**

$\mathbf{T}_\Sigma$ : the structural mapping of $\Sigma$

$\mathbf{P}_\Sigma$ : the precondition of $\Sigma$

$\mathbf{Q}_\Sigma$ : the postcondition of $\Sigma$

$\mathbf{t}_\Sigma$ : the instance mapping of $\Sigma$

## Generic transformation

Let us consider the following transformation. It specifies an operator through which a relation is replaced with its projections on two overlapping subsets of attributes whose union covers the relation attributes.

```
P: - R(U)
   - I∪J = U; I ≠ J; I∩J ≠ {}
Q: - R1(I), R2(J)
   - R1[I∩J] = R2[I∩J]
t: - R1 = R[I]
   - R2 = R[J]
```

Since R, I, J, etc, obviously are no actual names, but are some kind of **variables** which should be replaced with actual names, this transformation is called **generic**. It is intended to describe the characteristics of a large class of actual transformations.

## Instanciated transformation

Let us carry out the following substitutions :

R ← C
U ← {CUST#,CNAME,CITY,AMOUNT}
I ← {CUST#,CNAME,CITY}
J ← {CITY,AMOUNT}
R1 ← CC
R2 ← CA

We get a fully instanciated, or actual, transformation (all the variables have been assigned actual values) :

```
P: C(CUST#,CNAME,CITY,AMOUNT)
Q: CC(CUST#,CNAME,CITY)
   CA(CITY,AMOUNT)
   CC[CITY] = CA[CITY]
t: CC = C[CUST#,CNAME,CITY]
   CA = C[CITY,AMOUNT]
```

## Observation

The two kinds of statements in P and Q clearly appear in this example :

- structural declaration and naming : R(U), R1(I), R2(J), R1[I∩J] = R2[I∩J]; these statements are preserved (after substitution) in the instantiated transformation;
- applicability constraints : I∪J = U; I ≠ J; I∩J ≠{}; they control the substitution process, and are not translated; they disappear in the instantiation process.

## Semi-generic transformation

The substitution process can be incomplete. We still get a generic transformation, but it describes a smaller class of actual transformations :

```
P: C(CUST#,CNAME,CITY,AMOUNT)
   I∪{CITY,AMOUNT}={CUST#,CNAME,CITY,AMOUNT}
   I∩{CITY,AMOUNT} ≠ {}
Q: R1(I)
   CA(CITY,AMOUNT)
   R1[I∩{CITY,AMOUNT}] = R2[I∩{CITY,AMOUNT}]
t: R1 = R[I]
   CA = R[CITY,AMOUNT]
```

# 1. Preliminary discussion

***First case study***

Let us consider the transformation discussed in section 2.2, called Σ1 in this discussion :

```
P: - R(U)
   - I∪J = U;I ≠ J;I∩J ≠ {}
Q: - R1(I), R2(J)
   - R1[I∩J] = R2[I∩J]
t: - R1 = R[I]
   - R2 = R[J]
```

Once the current source instance of R has been transformed into instances of R1 and R2, how can we recover this source instance from the target instances ?

Answer : **in no way** !

Indeed, there are no algebraic, or procedural operators that could process the instances of R1 and R2 to yield the source instance of R in any situation. In other words, in general, Σ1 partially destroys the data contents of R, in such a way that we can claim that both schemas do not have the same semantics.

This transformation is **not semantics-preserving**, or is **not reversible**.

***Second case study***

Let us now consider a very popular transformation : the decomposition principle of the relational theory [DELOBEL,73] [FAGIN,77]. It can be (incompletely) paraphrased as follows :

```
P: R(U)
   I∪J∪K = U; I ≠ J ≠ K
   R: I →→ J|K
Q: R1(I,J)
   R2(I,K)
t: R1 = R[I,J]
   R2 = R[I,K]
```

The outstanding property of this transformation (called here Σ2) is that the instance of R can always be recovered by a natural join of the corresponding instances of R1 and R2.

In other words, there exists another transformation, called Σ3, which can **undo** the effect of this one. Its **t** part is clearly : `R = R1*R2`.

Now, this transformation is data-preserving or **semantics-preserving**. We also can call it a **reversible** transformation.

However, can we consider that this is a perfect transformation as far as reversibility is concerned ?

Not quite, unfortunately. It has a somewhat annoying drawback. Indeed, let us suppose that we have two arbitrary relations `R1(I,J)` and `R2(I,K)`. We can compute their join : `R(I,J,K)` = `R1*R2`. This operation (Σ3) seems to be the inverse of the decomposition transformation (Σ2). Unfortunately, Σ3 is not reversible, nor semantics-preserving : projecting the target instance R onto {I,J} and {I,K} does not yield the source instances of R1 and R2, since non-matching rows are lost.

**Amazing conclusions**

Σ3 is the inverse of Σ2

Σ2 is **not** the inverse of Σ3

Σ2 is reversible

Σ3 is **not** reversible

Σ2 definitely is a reversible transformation, but a second class one only !

***Third case study***

We can refine the decomposition principle by adding a derived property to its Q predicate, as follows (transformation Σ4):

```
P: R(U)
   I∪J∪K = U; I ≠ J ≠ K
   R: I →→ J|K
Q: R1(I,J)
   R2(I,K)
   R1[I]=R2[I]
t: R1 = R[I,J]
   R2 = R[I,K]
```

Σ4 too is reversible, but how about its inverse (Σ5) ? It should read, more or less, as follows :

```
P: R1(I,J)
   R2(I,K)
   R1[I]=R2[I]
Q: R(U)
   I∪J∪K = U; I ≠ J ≠ K
   R: I →→ J|K
t: R=R1*R2
```

Σ5 is reversible : given arbitrary instances of R1 and R2, such that `R1[I]=R2[I]`, these instances can be recovered by projecting their join. Therefore :

Σ5 is the inverse of Σ4

Σ4 is the inverse of Σ5

Σ4 is reversible

Σ5 is reversible

Σ4 is really a first class reversible transformation.

## Some preliminary conclusions

There are three classes of transformations :

- **non-reversible** transformations : they have no inverse
- **simply reversible** transformations : they have an inverse
- **symmetrically reversible** transformations : they have a reversible inverse

Let us analyse these phenomena in some detail.

## 2. Simple reversibility (R-transformations)

Transformation **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 instance c of C :

$$P1(C) \Rightarrow \; T2(T1(C))=C \text{ and } t2(t1(c))=c$$

**Σ2 is the inverse of Σ1, but not conversely**

Notation : **SCHEMA1 ⇒ SCHEMA2**

## 3. Symmetrical reversibility (SR-transformations)

Transformation **Σ1** = <T1,t1> = <P1,Q1,t1> is **symmetrically reversible** iff,

<T1,t1> is reversible and its inverse Σ2 is reversible

in other words,

$$P1(C) \Rightarrow [T2(T1(C))=C] \text{ and } [t2(t1(c))=c]$$
$$P2(C') \Rightarrow [T1(T2(C'))=C'] \text{ and } [t1(t2(c'))=c']$$

Notation : **SCHEMA1 ⇔SCHEMA2**

**Observation** : **Σ2 = <Q1,P1,t2>**

hence the concise notation for Σ1 + Σ2 : **Σ = <P,Q,t1,t2>**

## 4. Some (*slightly adjusted!*) quotations

... 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**.

[FAGIN,81]

... 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***.

[RISSANEN,77]

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 [...]*.

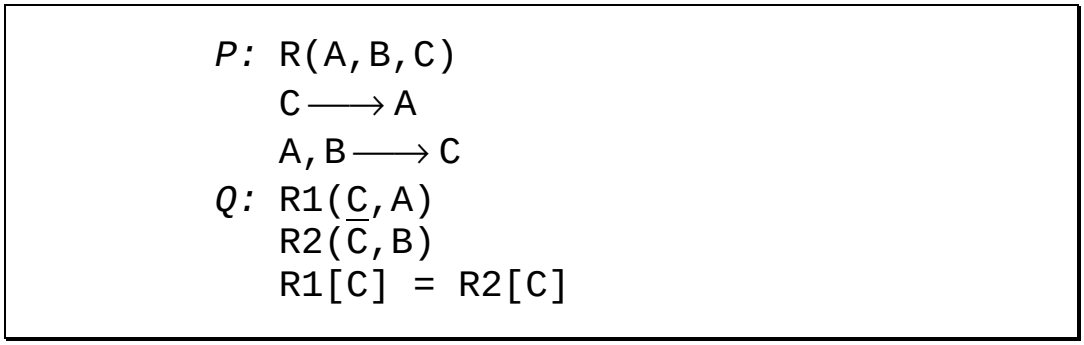
[CASANOVA,84] [ROSENTHAL,88]

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.

[BATINI,92]

## Situation

The source schema can include constraints such that the target schema is unable to support them. In the following example (limited to the **T** part), the FD A,B → C is lost :




## Problems

Let IC be a constraint (Keys, FD, MD, JD, ID, etc).

- *Constraint propagation*

  How should IC, holding in source schema S, be propagated to the target schema S' ?

- *Constraint preserving transformation*

  What is the minimal stronger precondition P', such that P' _ P, that garantees the propagation of IC ?

- *Lost constraints*

  How to express lost constraints ?
  Example : R1*R2:A,B ⟶ C

One of the most challenging problem still to be solved [KOBAYASHI,86].

Let us consider the following notation :

$$\mathbf{S' \leftarrow S(T(C) \leftarrow C)}$$

to be interpreted as : *S, renamed S', is replaced by S where C is replaced by T(C)* .

## Non-redundant transformation

$\Sigma \equiv <T,t>$ is a non-redundant transformation if T(C) completely replaces C in any schema S, i.e. if its application to $C \subseteq S$ has the following net effect :

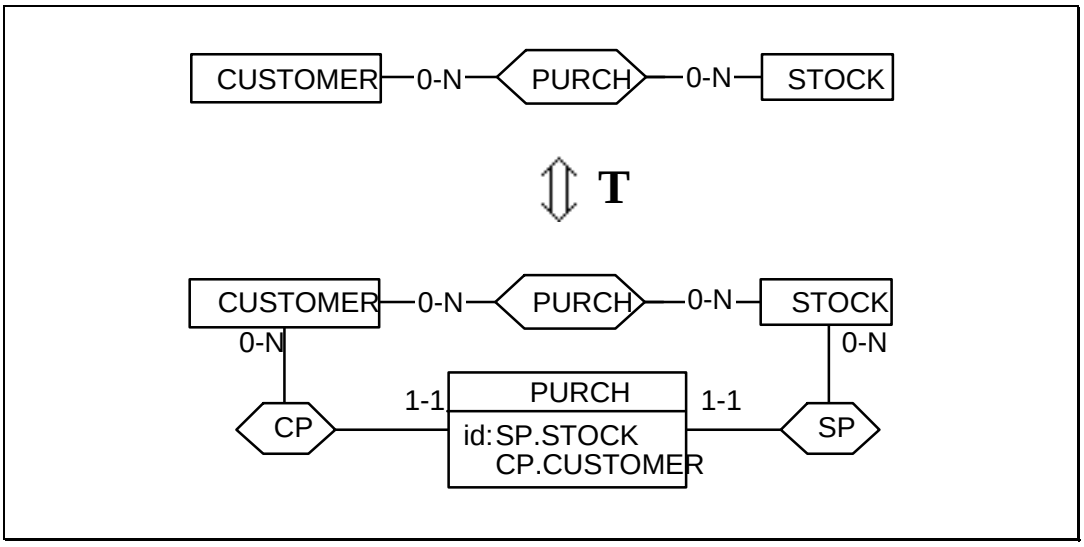
$$\mathbf{S' \leftarrow S(T(C) \leftarrow C)}$$

## Redundant transformation

$\Sigma \equiv <T,t>$ is a redundant transformation if some fragment of C coexists with T(C) in any schema S, i.e. if its application to $C \subseteq S$ has the following net effect :

$$\mathbf{S' \leftarrow S(C" \cup T(C) \leftarrow C)}$$
$$\mathbf{C" \subseteq C \ \&\ C" \neq \{\}}$$

Example of redundant transformation :



We need a precise notation to express the *definition* of transformations (P, Q, t) and (P, Q, t1, t2), as well as their *signature*.

## Definition

| | |
|---|---|
| *P:* | R(I,J,K)<br>R: I →→ J\|K |
| *Q:* | R1(I1,J)<br>R2(I2,K)<br>R1[I1]=R2[I2] |
| $t_1$*:* | let r be the current instance of R,<br>let r1 be an instance of R1,<br>let r2 be an instance of R2,<br>r1 = r[I,J]<br>r2 = r[I,K] |
| $t_2$*:* | let r1 be the current instance of R1,<br>let r2 be the current instance of R2,<br>let r be an instance of R,<br>r = r1(I1)*(I2)r2 |

## Signature

direct : (R1,R2) ⟵ **PJ**(R,I,J)

reverse : (R,I) ⟵ **PJ$^{-1}$**(R1,R2,I1,I2)

# Part 3

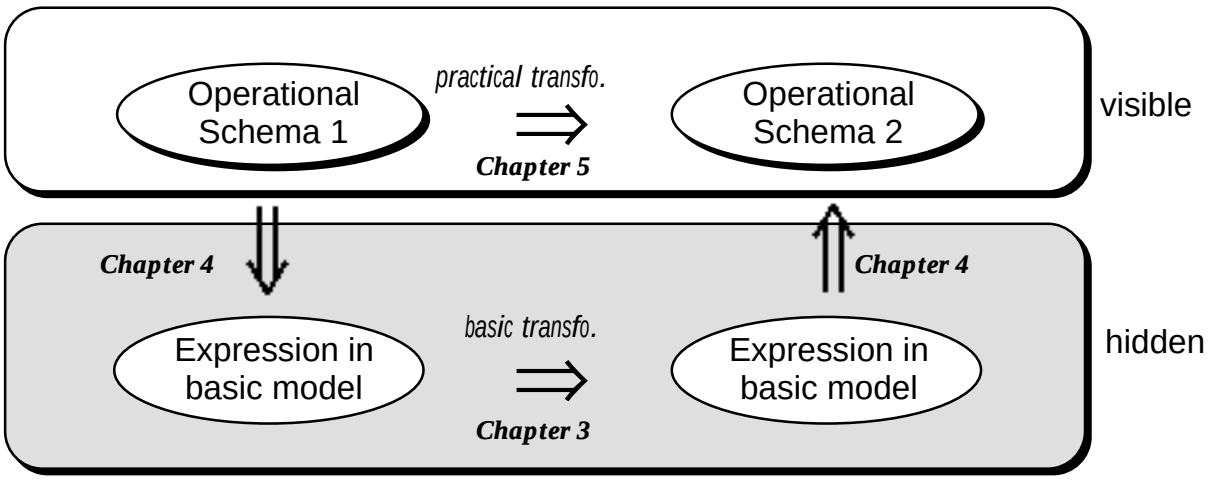
## BASIC TRANSFORMATIONS

## Observation

- A transformation is useful (or practical) if it can be used by practitioners.
- A practical transformation must be expressed in the operational data model(s) used by developers (ER, NIAM, OMT, UML, OO, SQL, CODASYL, IMS, standard files, etc).
- There exist hundreds of practical transformations, each in different variants according to the operational model.
- It is practically impossible to analyse each of them in a secure way (exact precondition and postcondition, reversibility, constraint propagation, etc).
- At first glance, there exist strong similarities between practical transformations.

## Proposal

- To define a minimal set of simple techniques (10 ?)
- To express them in a basic data model, which should be simple, generic and standard (e.g. some variant of the N1NF relational model)
- To express a bidirectional mapping between the operational model and the basic N1NF model



## Warning

- the basic transformations form a toolset for the **transformation engineer**
- **they are not aimed at the developer**
- they allow to generate, specify and analyse practical transformations
- they cover most important practical transformations, **but not all** (some work need to be done)

## (Over)optimistic references

[HAINAUT,81] 4 generic binary transformations [*covering most needs*] in model translation

[D'ATRI,84] [*all entity-preserving transformations are synthetized*] into 6 semantics-preserving graph restructuring operators

[KOBAYASHI,86] 4 generic transformations that [*cover most of the useful schema retructuring needs*]

The *basic model* is a variant of the N1NF (with multivalued, compound attributes) model(s) [SCHEK,86] [LEVENE,92] [ABITEBOUL,87], [HULL,87].

Why ?

- simpler than practical models
- more powerful than 1NF models (closer to practical models)
- semantics well defined
- theoretical tools available : algebrae, calculi, dependency structures, normal forms, etc

## Domains

- static domain : set of values
- dynamic domain : evolving set of values
- `index` : {1..N}; index attributes have no missing values
- included domains (= subset) :

  A : integer

  B : A

# Relations

### Syntax

```
employee(PID[1-1]:     integer,
         NAME[1-1]:    name,
         1ST-NAME[0-1]:name,
         PHONE[1-5]:   phone,
         ADDRESS[1-1]:(STREET[1-1]:name,
                       CITY[1-1]:name)  )
```

*an employee has one (from 1 to 1) personal ID, one name, from 0 to 1 christian name, from 1 to 5 phone numbers, and one address, which is made of one street and one city.*

| | |
|---|---|
| $A_k$[$i_k$-$j_k$]:$D_k$ | each $A_k$ value is made of a set of n $D_k$ values, with $i_k \le n \le j_k$ |
| $A_k$ | attribute $A_k$ |
| [$i_k$-$j_k$] | cardinality constraint of attribute $A_k$ |
| $D_k$ | domain of attribute $A_k$ |

### The attributes

| | |
|---|---|
| - single-valued attribute : | $j_k = 1$ |
| - multivalued attribute : | $j_k > 1$ |
| - mandatory attribute : | $i_k = 0$ |
| - optional attribute : | $j_k > 0$ |
| - atomic attribute : | PID, 1ST-NAME, PHONE, CITY |
| - compound attribute : | ADRRESS |

**!! The *null* value is replaced with the *empty set {}***

Extensions : non set-theoretic constructs

- bag : `R(A,B[I-J]bag:integer,C)`
- list : `R(A,B[I-J]list:integer,C)`

- unique list : `R(A,B[I-J]u-list:integer,C)`
- array : `R(A,B[I-J]array:integer,C)`
- unique array : `R(A,B[I-J]u-array:integer,C)`

**Shorthands**

1. Cardinality [1-1] is implicit :
   `person(PID[1-1]:integer,NAME[1-1]:name,PHONE[0-1]:name)`
   can be rewritten as :
   `person(PID:integer,NAME:name,PHONE[0-1]:name)`
2. If an attribute and its domain have the same name, the domain specification can be omitted :
   `R(A:A,B:B,C:C)`
   can be rewritten as
   `R(A,B,C)`

## Constraints

- Domain constraints
  `CAR` $\subseteq$ `VEHICLE`
  `owns[CAR] = CAR`
- Relation constraint
  *identifier (key)*
  `owns(CUSTOMER,VEHICLE); id(owns):VEHICLE`
  shorthand : `owns(CUSTOMER,VEHICLE)`
  *FD, MD, etc*

- Attribute constraint
  *cardinality (a customer owns from 0 to 5 vehicles)*
  `owns(CUSTOMER,VEHICLE)`
  `card(owns.CUSTOMER): [0-5]`
  *others*

- Examples
  domains `CUSTOMER,CAR,VEHICLE`
  relations `owns(CUSTOMER,CAR)`
  `cust(CUSTOMER,CID,NAME,PHONE[0-1])`
  constraints `CAR ⊆ VEHICLE`
  `owns[CAR] = CAR`
  `cust[CUSTOMER] = CUSTOMER`

## *The basic P/J transformation*

# Principles

A relation in which a multivalued dependency (e.g. a FD) holds can be decomposed into smaller fragments according to this dependency [DELOBEL,73], [FAGIN,77].

# Definition

| | |
|---|---|
| *P:* | R(I,J,K)<br>R: I →→ J\|K |
| *Q:* | R1(I1,J)<br>R2(I2,K)<br>R1[I1]=R2[I2] |
| $t_1$*:* | let r be the current instance of R,<br>let r1 be an instance of R1,<br>let r2 be an instance of R2,<br>r1 = r[I1,J]<br>r2 = r[I2,K] |
| $t_2$*:* | let r1 be the current instance of R1,<br>let r2 be the current instance of R2,<br>let r be an instance of R,<br>r = r1(I1)*(I2)r2 |

# Notation

direct : (R1,R2) ⟵ **PJ**(R,I,J)

reverse : (R,I) ⟵ **PJ**$^{-1}$(R1,R2,I1,I2)

# Properties

**PJ** is an SR-transformation (*Fagin's decomposition theorem*)

**PJ** and **PJ**$^{-1}$ preserve any FDs whose LHS is preserved

## Example

*Source schema*

```
works(who:EMP,in:PROJ,for:DEPART)
who ⟶ for
```

*Transformation*

```
(works-in,works-for) ⟵ PJ(works,{who},{in})
```

*Target schema*

```
works-in(who:EMP,in:PROJ)
works-for(who:EMP,for:DEPART)
works-in[who] = works-for[who]
```

# *Variants of the PJ transformation*

**1. I and K each comprise one attribute (A and C); C is optional**

| | |
|---|---|
| *P:* | R(A,J,C[0-1])<br>R: A →→ J\|C |
| *Q:* | R1(A1,J)<br>R2(A2,C)<br>R2[A2]⊆R1[A1] |
| $t_1$: | let r be the current instance of R,<br>let r1 be an instance of R1,<br>let r2 be an instance of R2,<br>r1 = r[A1,J]<br>r2 = r[A2,C] |
| $t_2$: | let r1 be the current instance of R1,<br>let r2 be the current instance of R2,<br>let r be an instance of R,<br>r = r1(A1)*>(A2)r2 *(right outer join)* |

**2. I and K each comprise one attribute (A and C); C is optional and multivalued**

| | |
|---|---|
| *P:* | R(A,J,C[0-$j_C$])<br>R: A →→ J\|C |
| *Q:* | R1(A1,J)<br>R2(A2,C[1-$j_C$])<br>R2[A2]⊆R1[A1] |
| $t_1$: | let r be the current instance of R,<br>let r1 be an instance of R1,<br>let r2 be an instance of R2,<br>r1 = r[A1,J]<br>r2 = r[A2,C](C≠{}) |
| $t_2$: | let r1 be the current instance of R1,<br>let r2 be the current instance of R2,<br>let r be an instance of R,<br>r = r1(A1)*>(A2)r2 *(right outer join)* |

**3. I and K each comprise one attribute (A and C); J is empty; C is optional and multivalued**

| | |
|---|---|
| *P:* | R(A,C[0-$j_C$])<br>R[A] = A |
| *Q:* | R2(A,C[1-$j_C$]) |
| $t_1$: | let r be the current instance of R,<br>let r2 be an instance of R2,<br>r1 = r[A,C](C≠{}) |
| $t_2$: | let r2 be the current instance of R2,<br>let r be an instance of R,<br>r = A*>r2 *(right outer join)* |

**4. I, J and K each comprise one attribute (A, B and C); B and C are optional and multivalued**

| | |
|---|---|
| *P:* | R(A,B[0-$j_B$],C[0-$j_C$])<br>R:A →→ B<br>R[A] = A |
| *Q:* | R1(A1,B[1-$j_B$])<br>R2(A2,C[1-$j_C$]]) |
| $t_1$: | let r be the current instance of R,<br>let r1 be an instance of R1,<br>let r2 be an instance of R2,<br>r1 = r[I,B](B≠{})<br>r2 = r[I,C](C≠{}) |
| $t_2$: | let r1 be the current instance of R1,<br>let r2 be the current instance of R2,<br>let r be an instance of R,<br>r = A*>(r1(A1)<*>(A2)r2) |

## Example (variant 4, reverse)

*Source schema* E-children(NE:EMP-NUM,CHILD[1-8])
E-phones(NE:EMP-NUM,PHONE[1-5])

*Transformation* (EMPLOYEE) ⟵ **PJ$^{-1}$**(E-children,E-phones,{NE},{NE})

*Target schema* EMPLOYEE(NE:EMP-NUM,CHILD[0-8],PHONE[0-5])
EMPLOYEE[NE] = EMP-NUM

## Principles

the result of a query (through any algebraic expression) is explicitly represented by a domain.

## Definition

| | |
|---|---|
| *P:* | schema S<br>algebraic expression **E** |
| *Q:* | schema S<br>domain X<br>*AE = attr(**E**)*<br>B(X,AE)<br>B[AE] = **E**<br>*X appears in B only* |
| $t_1$*:* | let s be the current instance of S,<br>let b be an instance of B,<br>generate arbitrary instance b of B such that b[AE] = **E** |
| $t_2$*:* | let b be the current instance of B,<br>drop b |

## Notation

direct : (X,B,{AE}) ⟵ **den**(S,**E**)

reverse : () ⟵ **den**$^{-1}$(S,X)

## Properties

**den** is an SR-transformation (add/delete redundant constructs)

## Example

*Source schema*

```
buys(CUST,PROD)
supplies(SUPPL,PROD)
```

*Transformation*

```
(DEAL,between,{CUST,SUPPL}) ⟵
den({buys,supplies},"(buys*supplies)[CUST,SUPPL]")
```

*Target schema*

```
buys(CUST,PROD)
supplies(SUPPL,PROD)
domain DEAL
between(DEAL,CUST,SUPPL)
between[CUST,SUPPL] = (buys*supplies)[CUST,SUPPL]
```

## *The basic extension transformation*

# Principles

The projection of a relation on some of its attributes is explicitly represented by a domain. This domain replaces these attributes in the relation [HAINAUT,90], [HAINAUT,91a].

# Definition

***First case*** : J is not empty :

| | |
|---|---|
| *P:* | R(I,J); *I,J not empty* |
| *Q:* | domain X<br>S(X,I)<br>T(X,J)<br>S[X] = T[X]<br>*X appears in S,T only* |
| $t_1$*:* | let r be the current instance of R,<br>let s be an instance of S,<br>let t be an instance of T;<br>generate arbitrary instance s of S such that s[I]=r[I]<br>t = (r*s)[X,J] |
| $t_2$*:* | let s be the current instance of S,<br>let t be the current instance of T,<br>let r be an instance of R;<br>r = (s*t)[I,J] |

***Second case*** : J is empty :

| | |
|---|---|
| *P:* | R(I) |
| *Q:* | domain X<br>S(X,I)<br>*X appears in S only* |
| $t_1$*:* | let r be the current instance of R,<br>let s be an instance of S,<br>generate arbitrary instance s of S such that s[I]=r |
| $t_2$*:* | let s be the current instance of S,<br>let r be an instance of R;<br>r = s[I] |

## Notation

direct : `(X,S,T)` ⟵ **`ext`**`(R,I)`

reverse : `R` ⟵ **`ext`**$^{-1}$`(X,S,T)`

*N.B : T is an optional parameter*

## Properties

**ext** is an SR-transformation (uses the **den** and **PJ**$^{-1}$ transformations)

## Example

*Source schema*

```
program(TEACHER,SUBJECT,DATE)
```

*Transformation*

```
(LECTURE,defined-as,program) ⟵
ext(program,{TEACHER,SUBJECT})
```

*Target schema*

```
domain LECTURE
program(LECTURE,DATE)
defined-as(LECTURE,TEACHER,SUBJECT)
defined-as[LECTURE] = program[LECTURE]
```

## *The extension-decomposition transformation*

***First case*** : J is not empty :

| | |
|---|---|
| *P:* | $R(I_1,..,I_m,J)$; $m \geq 1$<br>$I_i$ *not empty* $(i \in [1..m])$; *J not empty* |
| *Q:* | $S_i(\underline{X},I_i)$ $i \in [1..m]$<br>$T(X,J)$<br>$S_i[X] = T[X]$ $i \in [1..m]$<br>$(*S_i,i \in [1..m]):I_1,..,I_m \longrightarrow X$<br>*X appears in* $S_i$,*T only* |
| $t_1$: | let r be the current instance of R,<br>let $s_i$ be an instance of $S_i$, $i \in [1..m]$<br>let t be an instance of T;<br>generate arbitrary instances $s_i$ of $S_i$ such that :<br>$(*s_i,i \in [1..m])[I]=r[I]$<br>$t = (r*(*s_i,i \in [1..m]))[X,J]$ |
| $t_2$: | let $s_i$ be the current instance of $S_i$, $i \in [1..m]$<br>let t be the current instance of T,<br>let r be an instance of R;<br>$r = (r*(*s_i,i \in [1..m]))[I,J]$ |

***Second case*** : J is empty :

| | |
|---|---|
| *P:* | $R(I_1,..,I_m)$; $m > 1$<br>$I_i$ *not empty* $(i \in [1..m])$ |
| *Q:* | $S_i(\underline{X},I_i)$ $i \in [1..m]$<br>$S_i[X] = S_j[X]$ $i,j \in [1..m]$<br>$(*S_i,i \in [1..m]):I_1,..,I_m \longrightarrow X$<br>*X appears in* $S_i$ *only* |
| $t_1$: | let r be the current instance of R,<br>let $s_i$ be an instance of $S_i$, $i \in [1..m]$<br>generate arbitrary instances $s_i$ of $S_i$ such that :<br>$(*s_i,i \in [1..m])[I]=r$ |
| $t_2$: | let $s_i$ be the current instance of $S_i$, $i \in [1..m]$<br>let r be an instance of R;<br>$r = (*s_i,i \in [1..m])[I]$ |

**Notation**

direct : `(X,{S1,S2,..,Sm},T)` ⟵ **`ext-dec`**`(R,{I1,I2,..,Im})`
*N.B : T is an optional parameter*

reverse : `R` ⟵ **`ext-dec`**$^{-1}$`(X,{S1,S2,..,Sm})`

## *The basic composition transformation*

## Principles

A relation is replaced by its composition with another relation.

## Definition

| | |
|---|---|
| *P:* | R(I,K); *I,K not empty*<br>S(K',L); *K',L not empty*<br>S[K'] ⊆ R[K] |
| *Q:* | R(I,K);<br>T(I',L);<br>T[I'] ⊆ R[I]<br>R(I)*(I')T : K ⟶→ L\|I |
| $t_1$: | let r be the current instance of R,<br>let s be the current instance of S,<br>let t be an instance of T,<br>t = (r(K)*(K')s)[I,L] |
| $t_2$: | let r be the current instance of R,<br>let t be the current instance of T,<br>let s be an instance of S,<br>s = (r(I)*(I')t)[K,L] |

## Notation

direct : T ⟵ **comp**(R,S,K,K')

reverse : S ⟵ **comp**$^{-1}$(R,T,I,I')

## Properties

**comp** is an SR-transformation (uses the **PJ** and **PJ**$^{-1}$ transformations)

**comp** is symmetrical

## *Variant : R is bijective*

| |
|---|
| *P:* R(I,K);<br>S(K',L);<br>S[K'] ⊆ R[K] |
| *Q:* R(I,K);<br>T(I',L);<br>T[I'] ⊆ R[I] |
| $t_1$: let r be the current instance of R,<br>let s be the current instance of S,<br>let t be an instance of T,<br>t = (r(K)*(K')s)[I,L] |
| $t_2$: let r be the current instance of R,<br>let t be the current instance of T,<br>let s be an instance of S,<br>s = (r(I)*(I')t)[K,L] |

## Example

*Source schema*
manages(MANAGER,DEPART)
works-in(EMPLOYEE,DEPART)
works-in[DEPART] ⊆ manages[DEPART]

*Transformation*
works-for ⟵ **comp**(manages,works-in,{DEPART},{DEPART})

*Target schema*
manages(MANAGER,DEPART)
works-for(EMPLOYEE,MANAGER)
works-for[MANAGER] ⊆ manages[MANAGER]

## Note

This transformation can be generalized to *ternary* R relations :

| |
|---|
| *P:* R(I,K,J)<br>S(K',L)<br>S[K'] ⊆ R[K] |
| *Q:* R(I,K,J)<br>T(I',L)<br>T[I'] ⊆ R[I] |

# *The basic NEST/UNNEST transformation*

## Principles

A N1NF relation is replaced by its equivalent 1NF version [SCHEK,86] [LEVENE,92].

## Definition

| | |
|---|---|
| *P:* | R(I,B[1-N]) |
| *Q:* | R1(I,B) |
| $t_1$: | let r be the current instance of R,<br>let r1 be an instance of R1,<br>r1 = μB(r) |
| $t_2$: | let r1 be the current instance of R1,<br>let r be an instance of R,<br>r = νB(r1) |

## Notation

direct : R1 ⟵ **unnest**(R,B)

reverse : R ⟵ **unnest**$^{-1}$(R1,B)

## Properties

This version of **unnest** is an SR-transformation; indeed, according to, e.g., [DARWEN,93] : considering the relation R(A,B*,C), the application of the unnest relational operator on B* is (symmetrically) reversible *iff*

- *no tuple of R has an empty relation as its B value;*
- *B is functionally dependent on the set of all the other attributes of R.*

## Example

*Source schema* contacts(EMPLOYEE,PHONE[1-N])

*Transformation* contact ⟵ **unnest**(contacts,PHONE)

*Target schema* contact(EMPLOYEE,PHONE)

# *Variants of the NEST/UNNEST transformation*

**1. B is optional**

*Problem* : the tuples with an empty B value will be lost in the μ operation.

*Solution* : all the I values in R are known to form a reference set, defined by expression **E**.

| | |
|---|---|
| *P:* | R(I,B[0-N])<br>R[I] = **E** |
| *Q:* | R1(I,B)<br>R1[I] ⊆ **E** |
| $t_1$: | let r be the current instance of R,<br>let r1 be an instance of R1,<br>r1 = $\mu_B$(r) |
| $t_2$: | let r1 be the current instance of R1,<br>let r be an instance of S,<br>r = result(**E**)*>$\nu_B$(r1) |

Example of reference set : R[I] = I, for I comprising one attribute only

**2. B is optional, I is made of a single-valued attribute (A)**

Every value of domain A appears in some R tuple (**E** = domain A).

| | |
|---|---|
| *P:* | R(A,B[0-N])<br>R[A] = A |
| *Q:* | R1(A,B) |
| $t_1$: | let r be the current instance of R,<br>let r1 be an instance of R1,<br>r1 = $\mu_B$(r) |
| $t_2$: | let r1 be the current instance of R1,<br>let r be an instance of S,<br>r = A*>$\nu_B$(r1) |

## Example

*Source schema*
```
descr(EMPLOYEE,CHILD[0-N])
descr(EMPLOYEE) = EMPLOYEE
```

*Transformation*
```
children ⟵ unnest(descr,CHILD)
```

*Target schema*
```
children(EMPLOYEE,CHILD)
```

## Principles

A mandatory, single-valued, **compound** attribute is replaced by its components.

## Definition

| | |
|---|---|
| *P:* | R(I,B(J)) |
| *Q:* | R1(I,J) |
| $t_1$: | let r be the current instance of R,<br>let r1 be an instance of R1,<br>r1 = {(i,j): ∃ρ∈r:(i=ρ[I] & j=(ρ[B])[J]))} |
| $t_2$: | let r1 be the current instance of R1,<br>let r be an instance of R,<br>r = {(i,(j)): ∃σ∈r1: (i=σ[I] & j=σ[J])} |

## Notation

direct : R1 ⟵ **disag**(R,B)

reverse : (R,B) ⟵ **disag**$^{-1}$(R1,J)

## Properties

**disag** is an SR-transformation (it does not change the contents of r)

## Example

*Source schema* emp(EMPID,STREET,CITY)

*Transformation* (empl,ADDRESS) ⟵ **disag**$^{-1}$(emp,{STREET,CITY})

*Target schema* empl(EMPID,ADDRESS(STREET,CITY))

## Principles

Non purely set-theoretic constructs (bag, list, etc) are defined by equivalent set-theoretic structures. These definitions can be considered as SR-transformations.

## Definitions (<P,Q> part only)

***List : ordered sequence of (non necessarily distinct) values***

| |
|---|
| *P:* R(A,B[1-N]*list*) |
| *Q:* R'(A,BB[1-N]:(I:sequence,B)) |

***List-set : ordered sequence of distinct values***

| |
|---|
| *P:* R(A,B[1-N]*u-list*) |
| *Q:* R'(A,BB[1-N]:(I:sequence,B)) |

***Bag : unordered collection of (non necessarily distinct) values***

| |
|---|
| *P:* R(A,B[1-N]*bag*) |
| *Q:* R'(A,BB[1-N]:(B,N:{1..N})) |

***Array : addressable sequence of cells***

| |
|---|
| *P:* R(A,B[0-J]*array*) |
| *Q:* R'(A,BB[J-J]:(I:index,B[0-1])) |

# Part 4

## The GER : a Generic ER/OR Model

## Objective

To propose a framework that allows reasoning rigorously about, e.g., ER schema properties.

## Principle

The Generic Entity-Relationship (GER) model is a subset and a specialization of the N1NF model developed in Section 3.2, such that there exists a one-to-one mapping between the GER constructs and those of entity/object-based models :

- each GER construct has an ER/OR interpretation,
- each ER/OR construct has a GER expression.

## Application domain

The Entity/Object-based models, i.e. all the models that include the concept of self-identifying object, and of inter-object relationship, i.e. the Entity-Relationship (ER) or Object-Relationship (OR) models :

- all the variants of the Entity-Relationship model (including ORM);
- the binary conceptual models (e.g. NIAM)
- the Object-oriented models (including OMT, UML);
- the operational record-based DB models (CODASYL, IMS, TOTAL, etc);
- the file structure models (COBOL files and the like);
- the SQL models (easily included into ER/OR models)

## Immediate application

The basic transformations (section 3) can be given a straighforward interpretation in ER-like models.

We consider a specific dynamic domain, called **`entities`**. It designates all the objects or entities of interest at the present time.

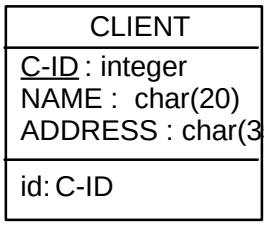
An entity type is expressed as a domain defined as a subset (= subtype) of this basic domain, or of another entity type :

```
PERSON : entities
ORDER : entities
CUSTOMER : PERSON
```

Primarily, an entity relation describes the attributes of an entity type. General form for entity type E :

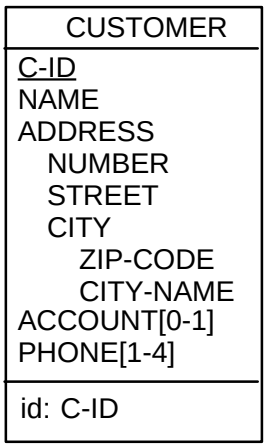
desc-of-E(E, *list of attributes of E*)

**Example 1** :




```
desc-of-CLIENT(CLIENT,C-ID:integer,NAME:char(20),ADDRESS:char(35))
```

**Example 2** (domains ignored) :

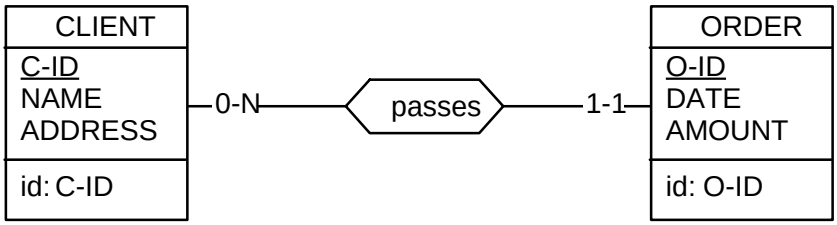



```
desc-of-CUSTOMER( CUSTOMER,
                  C-ID,
                  NAME,
                  ADDRESS:(
                     NUMBER,
                     STREET,
                     CITY:(
                         ZIP-CODE
                         CITY-NAME),)
                  ACCOUNT[0-1],
                  PHONE[1-4])
```

A relationship relation describes the roles and the attributes of a relationship type. General form for rel-type R :

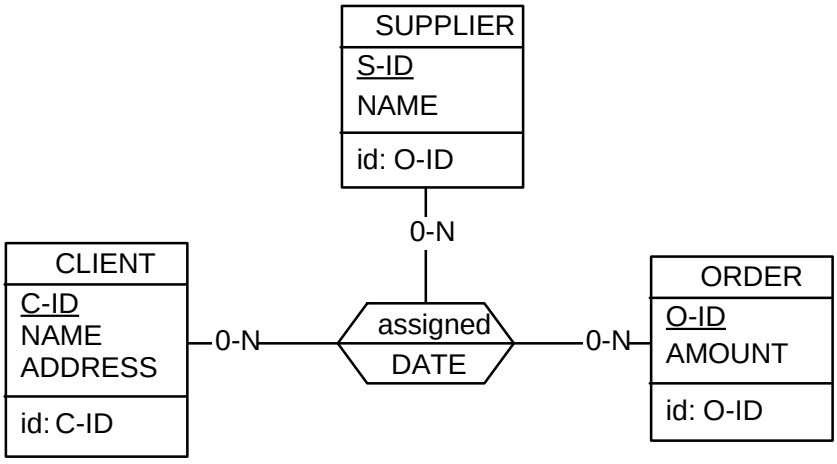
R (*list of roles of R,list of attributes of R*)

Example 1 :




passes(CLIENT,ORDER)

Example 2 :




assigned(CLIENT,ORDER,SUPPLIER,DATE)

## Identifier constraint

```
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT)
id(desc-of-ORDER):ORDER
id(desc-of-ORDER):O-ID
```

or, more concisely :

```
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT)
```

an id can be defined on a relational expression (the orders of a client have distinct O-ID) :

```
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT)
passes(CLIENT,ORDER)
id(desc-of-ORDER*passes): CLIENT,O-ID
```

## Domain constraint

```
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT)
passes(CLIENT,ORDER)
desc-of-ORDER[ORDER] = ORDER
passes[ORDER] = ORDER
```

## Inclusion constraint

```
orders(CUSTOMER,ITEM,SUPPLIER,DATE,QTY)
supplies(SUPPLIER,ITEM,PRICE)
orders[SUPPLIER,ITEM] ⊆ supplies[SUPPLIER,ITEM]
```

## Cardinality constraint

```
passes(CLIENT,ORDER)
card(passes.CLIENT): [0-20]
```

can express other constraints (Identifier and domain) :

```
passes(CLIENT,ORDER)
passes[ORDER] = ORDER
```

is equivalent to :

```
passes(CLIENT,ORDER)
card(desc-of-ORDER.ORDER): [1-1]
```

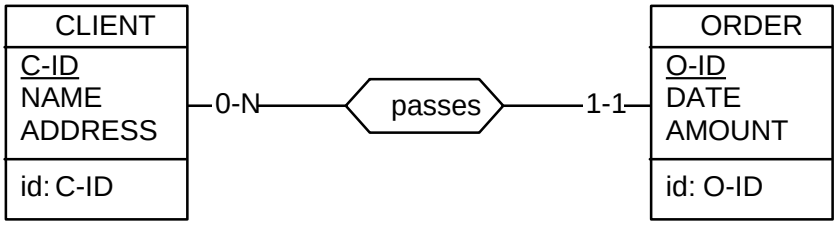
## Others

any set-theoretic assertion or relational constraint (FD, MD, JD, etc)

## Note on rel-type representation (1)

A concise form is proposed for *one-to-many* rel-types. It can be better to define some integrity constraints, such as identifiers and FD, that include roles.

*Standard form* :




```
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT)
passes(CLIENT,ORDER)
desc-of-ORDER[ORDER] = ORDER
passes[ORDER] = ORDER
```

*Concise form* :

```
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT,passes:CLIENT)
desc-of-ORDER[ORDER] = ORDER
```

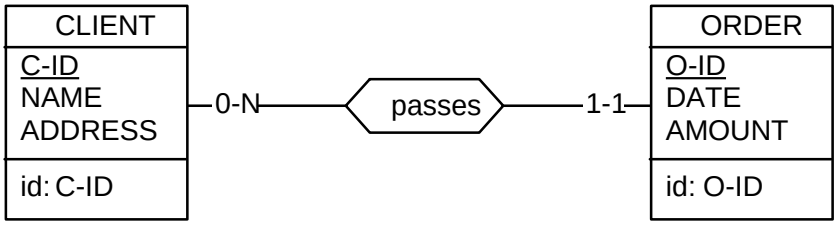
Proof of equivalence :

through **PJ**$^{-1}$ basic transformation.

## Note on rel-type representation (2)

Following the same rule, a more general form is proposed for any *binary* rel-type. It provides an adequate way to represent *Object-oriented* structures.

*Standard form* :




```
CLIENT : entities
ORDER : entities
desc-of-CLIENT(CLIENT,C-ID,NAME,ADDRESS)
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT)
passes(CLIENT,ORDER)
desc-of-CLIENT[CLIENT] = CLIENT
desc-of-ORDER[ORDER] = ORDER
passes[ORDER] = ORDER
```

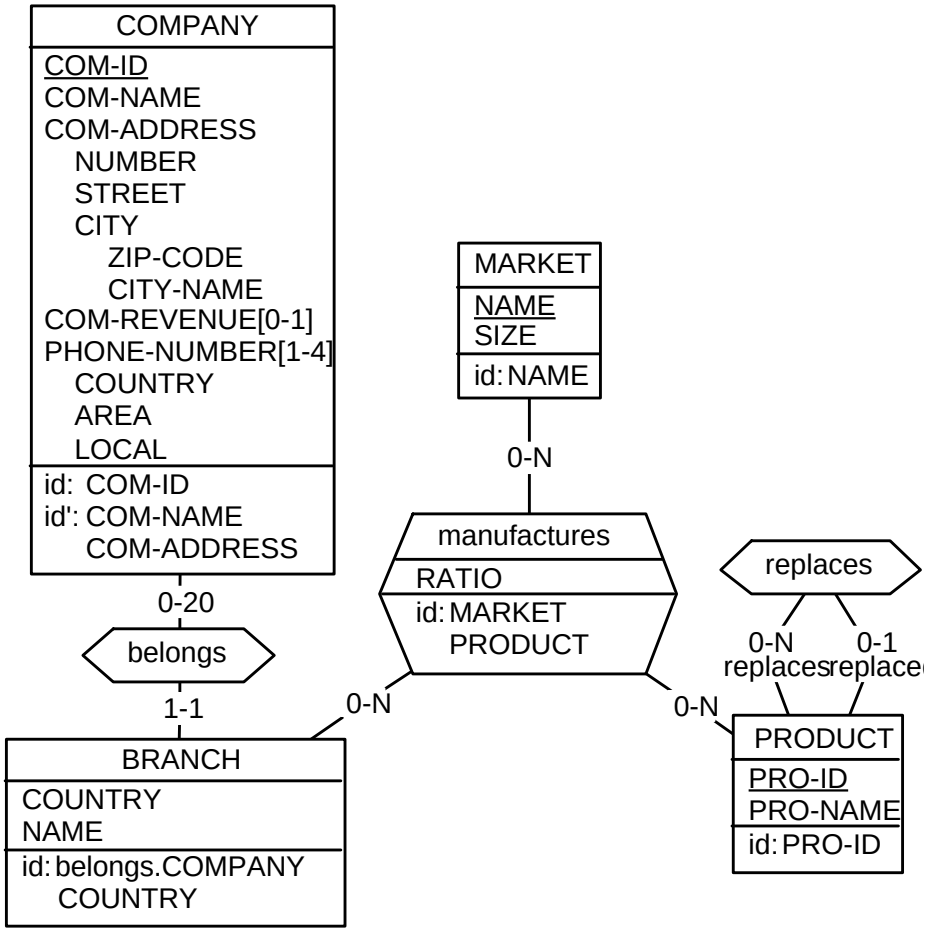
*Concise form* :

```
CLIENT : entities
ORDER : entities
desc-of-CLIENT(CLIENT,C-ID,NAME,ADDRESS,passes[0-
N]:ORDER)
desc-of-ORDER(ORDER,O-ID,DATE,AMOUNT)
desc-of-CLIENT[CLIENT] = CLIENT
∪desc-of-CLIENT[passes] = ORDER
desc-of-ORDER[ORDER] = ORDER
```

Proof of equivalence :

through **PJ**$^{-1}$ and **unnest** basic transformations.

# An ER schema


COMPANY
COM-ID
COM-NAME
COM-ADDRESS
NUMBER
STREET
CITY
ZIP-CODE
CITY-NAME
COM-REVENUE[0-1]
PHONE-NUMBER[1-4]
COUNTRY
AREA
LOCAL
id: COM-ID
id': COM-NAME
COM-ADDRESS
MARKET
NAME
SIZE
id: NAME
0-N
manufactures
RATIO
id: MARKET
PRODUCT
replaces
0-N
replaces
0-1
replace
0-20
belongs
1-1
0-N
0-N
BRANCH
COUNTRY
NAME
id: belongs.COMPANY
COUNTRY
PRODUCT
PRO-ID
PRO-NAME
id: PRO-ID

# ... and its GER expression

**Entity types**

```
COMPANY : entities
BRANCH :  entities
MARKET :  entities
PRODUCT : entities
```

**Entity type description** (domains of atomic attributes ignored)

```
desc-of-COMPANY(COMPANY,
          COM-ID,
          COM-NAME,COM-ADDRESS:(NUMBER,STREET,
                                CITY:(ZIP-CODE,CITY-NAME),
          COM-REVENUE[0-1],PHONE-NUMBER[1-4]:
                               (COUNTRY,AREA,LOCAL))

desc-of-BRANCH(BRANCH,
          COUNTRY,belongs:COMPANY,
          NAME)

desc-of-MARKET(MARKET,
          NAME,
          SIZE)

desc-of-PRODUCT(PRODUCT,
          PRO-ID,
          PRO-NAME)
```

**Relationship types**

```
manufactures(
          BRANCH,
          PRODUCT,MARKET,
          RATIO)

replaces( replaced:PRODUCT,
          replaces:PRODUCT)
```

**Constraints**

```
desc-of-COMPANY[COMPANY] = COMPANY
desc-of-BRANCH[BRANCH]   = BRANCH
desc-of-MARKET[MARKET]   = MARKET
desc-of-PRODUCT[PRODUCT] = PRODUCT
card(desc-of-BRANCH.belongs): [0-20]
```

# Part 5

## ER/OR SR-TRANSFORMATIONS

There exist several hundreds of practical transformations. Building a comprehensive catalog of such operators would be particularly tedious and would be endless anyway. Indeed, every analyst and developer will, one day or another, invent a new technique to represent some conceptual construct.

Therefore, the objective of this section is different. It describes, sometimes in detail, some of the most representative techniques that can be found in practical methodologies, in lectures, in text books and in CASE tools. In particular, the reader will find all the techniques that will be used in the following sections.

The practitioner will be given guidelines on how to define new transformations, and on how to prove the properties of a practical transformation. Of course, to make this material (tentatively) readable, the treatment is a bit informal. Nevertheless, the researcher will also find some hints on how to develop more precise techniques and demonstrations.

The organization is as follows :

- **Informal description**

  Short textual description of the principles of the transformation; some references.

- **General pattern**

  A generic graphical example describing the main aspects of the technique; the signature.

- **Analysis / GER Analysis**

  Either a discussion on the reversibility or on other properties, of the transformation. Sometimes, a precise demonstration of the reversibility is given through GER expressions and basic transformations.

- **Variants**

  Some specializations are given for selected transformations.

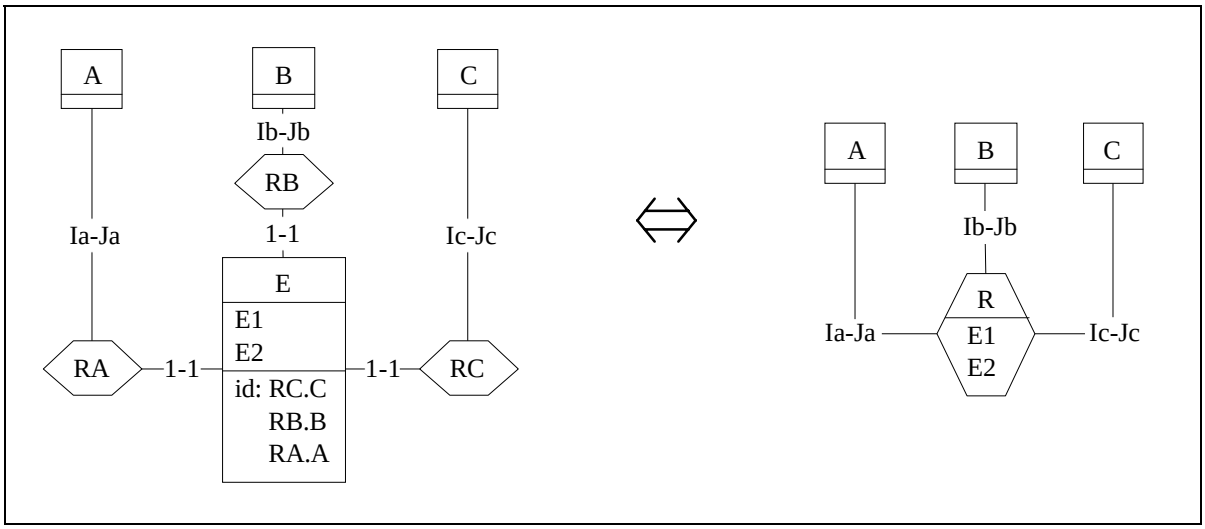
# 1. Transforming an Entity type into a Rel-type

## Informal description

Under certain conditions, an entity type E can be transformed into rel-type R. In particular, it must have an identifier, be linked to at least two other entity types, with binary *one-to-many* rel-types.

Reference : [HAINAUT,91a]

## General pattern




R ← **ET-to-RT**(E)

## Analysis

**ET-to-RT** = **RT-to-ET**$^{-1}$

*Conclusion* : according to 5.3, the transformation is symmetrically reversible

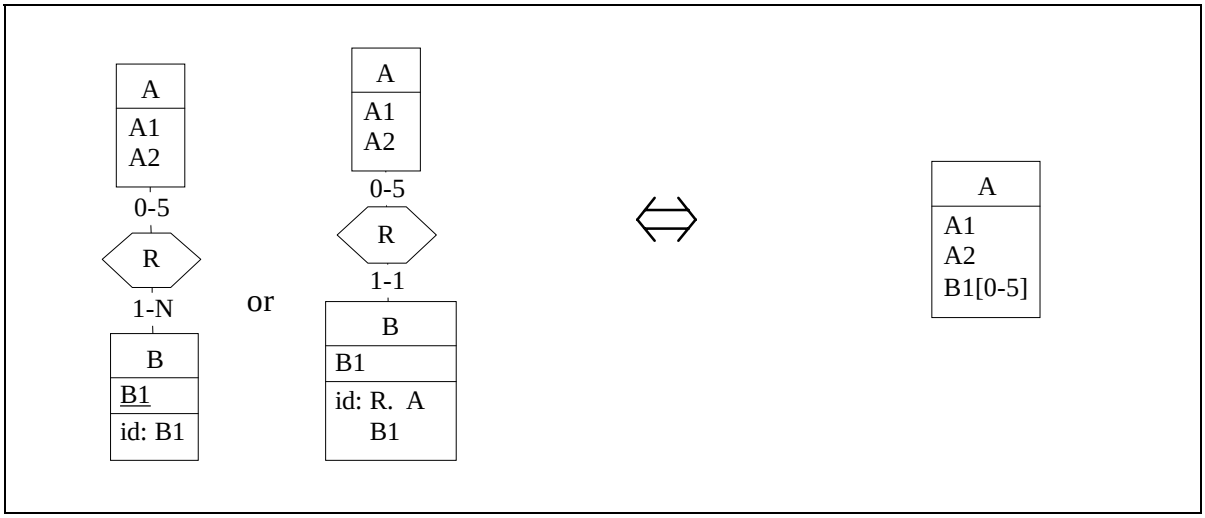
# 2. Transforming an Entity type into an Attribute

## Informal description

Under certain conditions, an entity type B can be transformed into attribute B1. In particular, it must play one and only role (in rel-type R), have one and only one identifier, and its attributes must belong to this identifier. There are two variants, according to whether R is *binary* or *N-ary*.

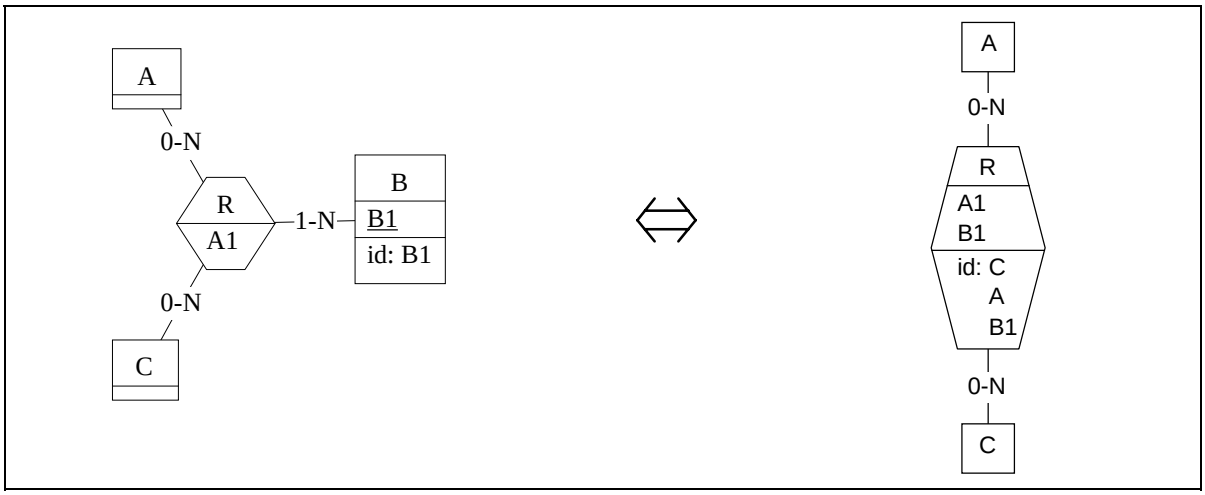
Reference : [HAINAUT,91a]

## General pattern (R is binary)




B1 ← **ET-to-Att**(B)

## General pattern (R is N-ary)




B1 ← **ET-to-Att**(B)

## Analysis

**ET-to-Att** = **Att-to-ET/Value**$^{-1}$ or **Att-to-ET/Instance**$^{-1}$

*Conclusion* : according to 5.4, the transformation is symmetrically reversible.

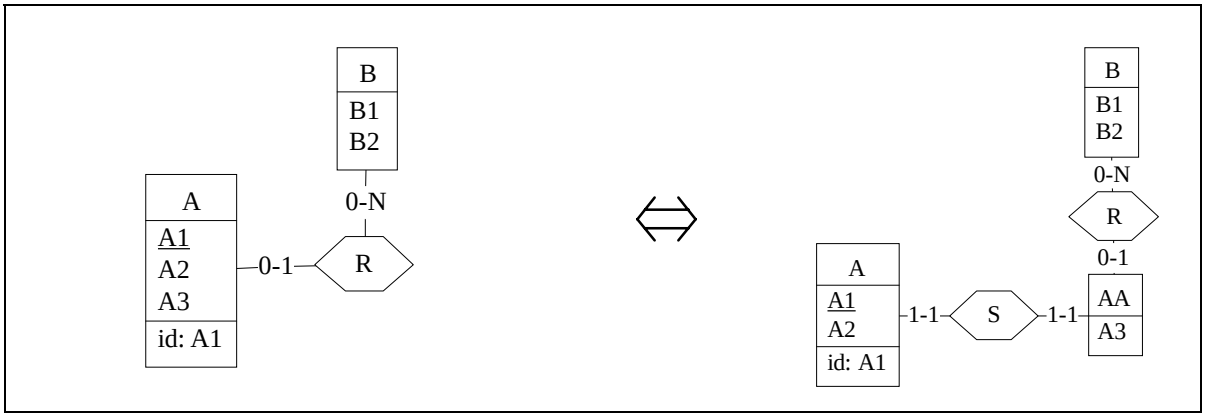
# 3. Splitting an Entity type

## Informal description

Some components (attributes and/or roles) of entity type A are extracted and are packed together into new entity type AA. This is an SR-transformation.

Reference : [HAINAUT,91a]

## General pattern




(AA,S) ← **Split-ET**(A,{A3,R.B})

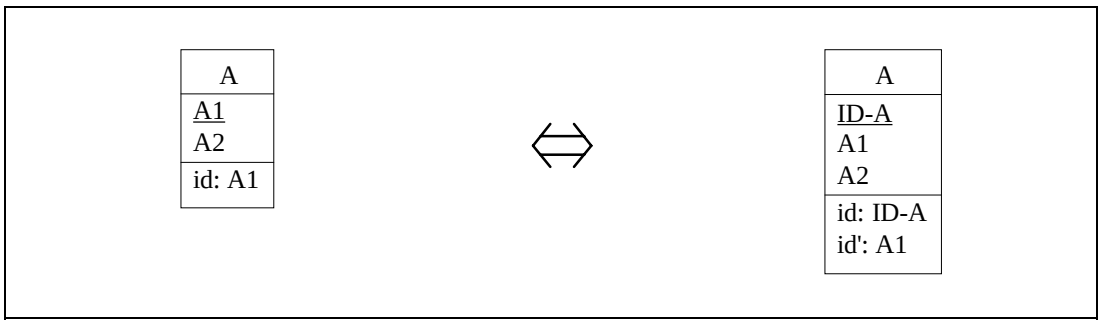
{A3,R.B} ← **Merge-ET**(AA,S)

# 4. Add a technical Identifier

## Informal description

Entity type A is given new (semantic-less) attribute ID-A, which is made its primary ID. If a primary ID already existed, it is made secondary. Adds (removes) a non-semantic construct : trivial SR-transformation.

## General pattern



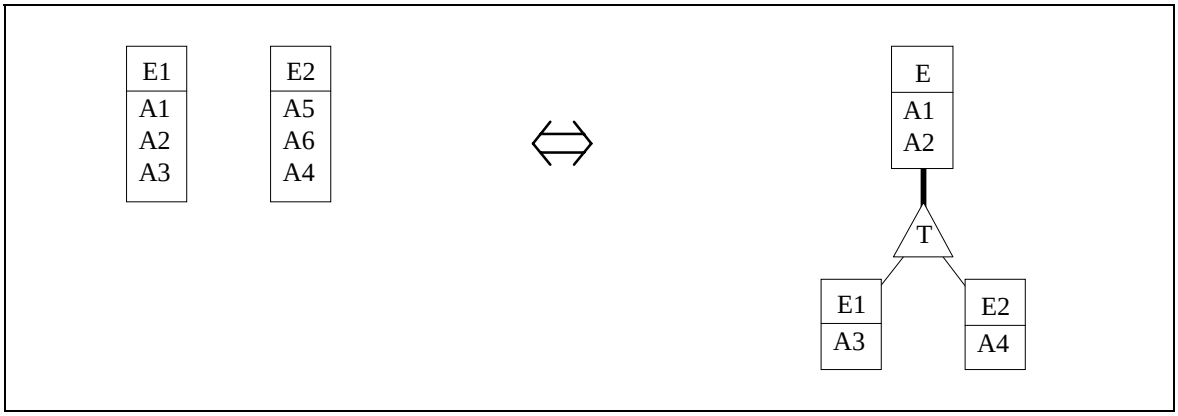

`ID-A` ← **`Tech-ID`**`(A)`

# 5. Make a supertype

## Informal description

Entity types E1 and E2 are given common supertype E. The common components (attributes and/or roles) are extracted and moved into E. Symmetrically reversible.

## General pattern




`(E,{A1,A2})` ← **`Make-Supertype`**`({(E1,{A1,A2}),(E2,{A5,A6})})`

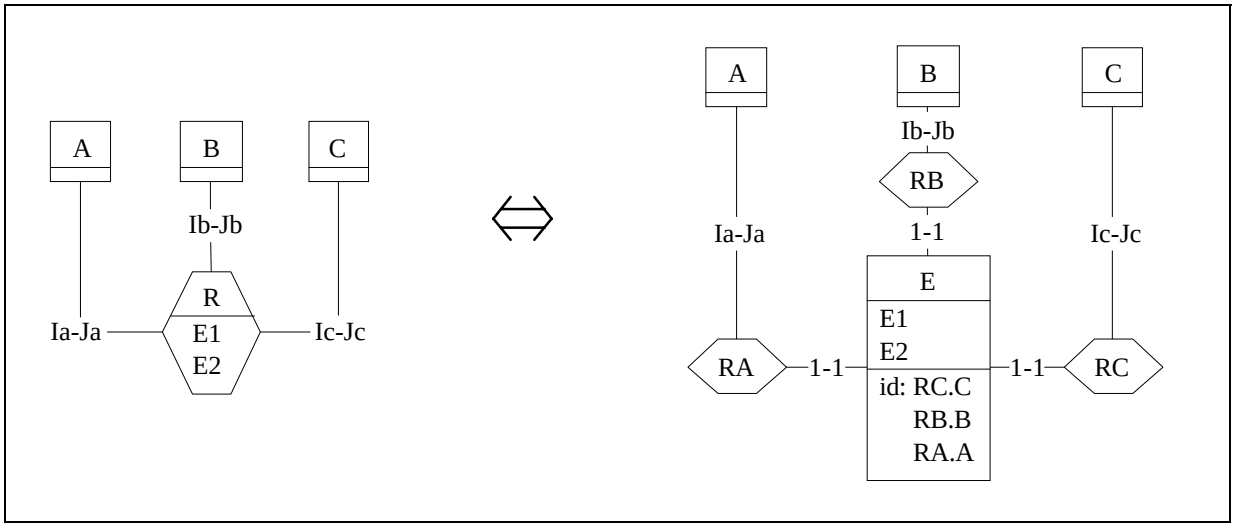
# 1. Transforming a Rel-type into an Entity type

## Informal description

A rel-type R can always be transformed into an entity type E and into rel-types that link E to the former roles of R.

Reference : [HAINAUT,91a], [BATINI,92], [JONIER,95]

## General pattern




(E,{(A,RA),(B,RB),(C,RC)}) ← **RT-to-ET**(R)

## GER analysis

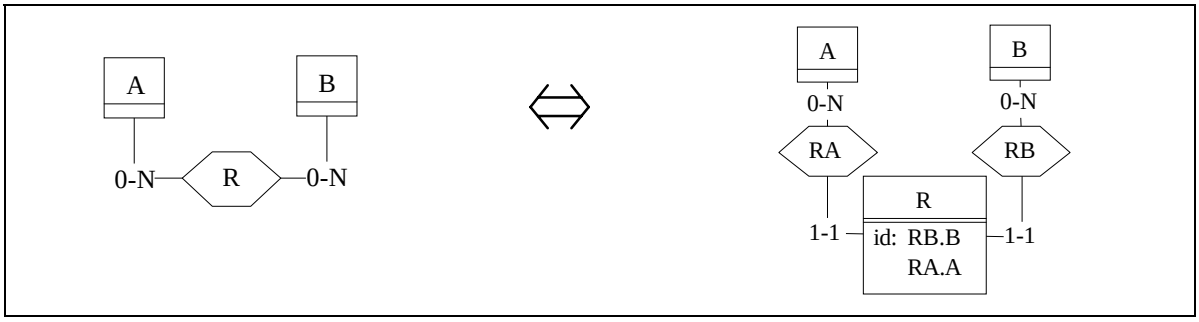
```
R(A,B,C,E1,E2)
```

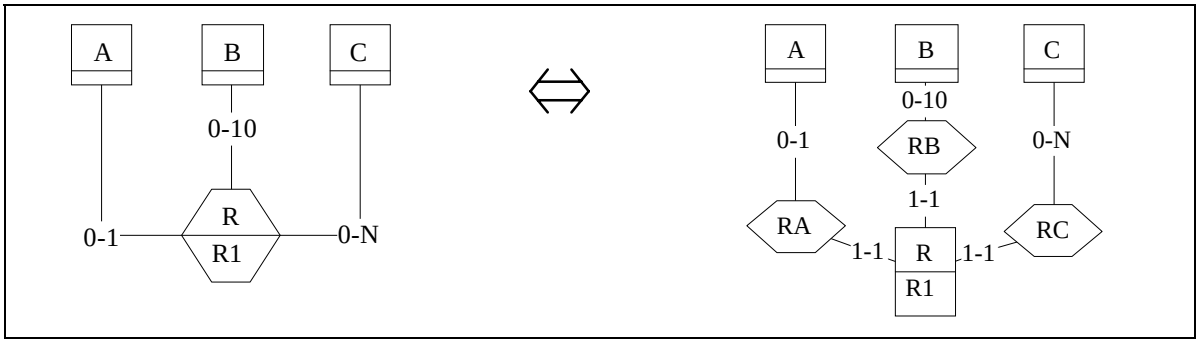
⇕ (E,{RA,RB,RC},desc-of-E) ← **ext-dec**(R,{{A},{B},{C}})

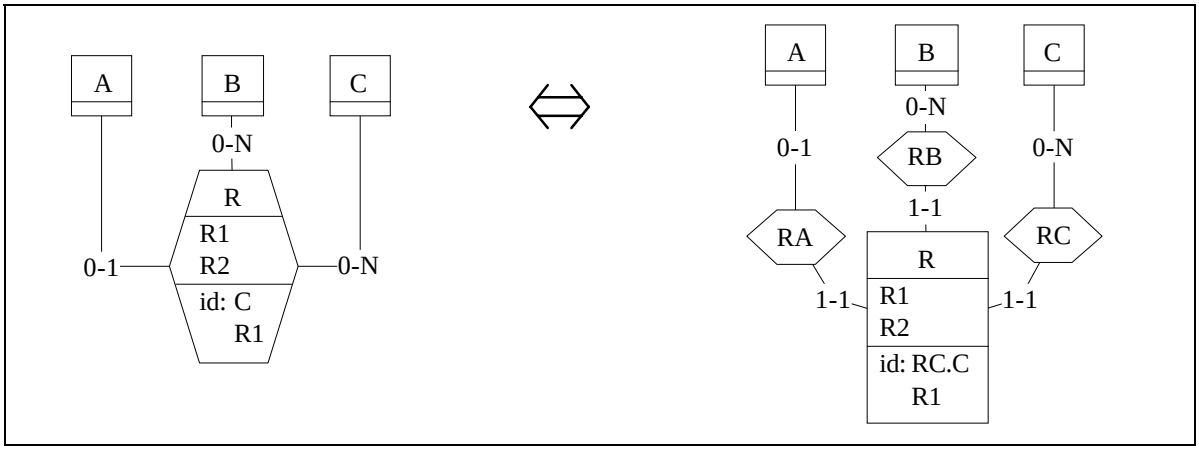
```
E:entities
RA(E,A)
RB(E,B)
RC(E,C)
desc-of-E(E,E1,E2)
RA*RB*RC:A,B,C → E
RA[E]=RB[E]=RC[E]=desc-of-R[E]=E
```

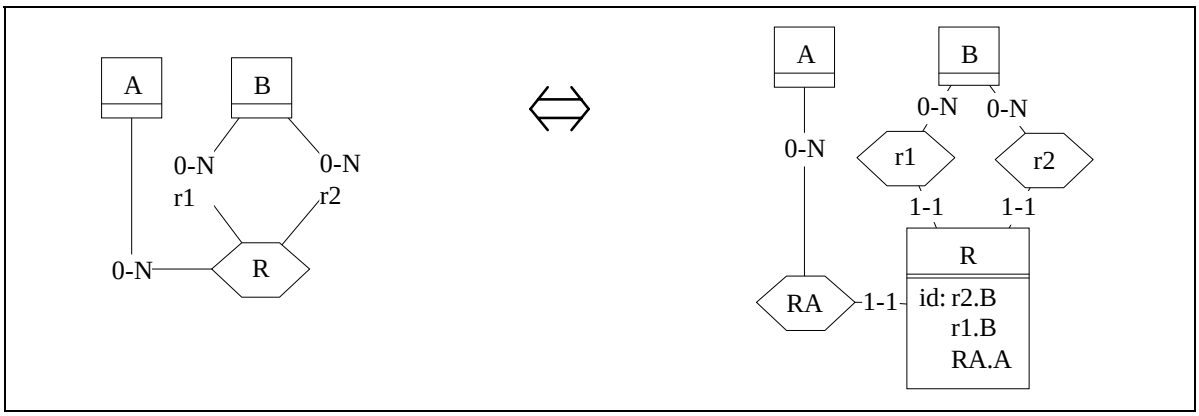
*Conclusion* : the **RT-to-ET** transformation is *symmetrically reversible*

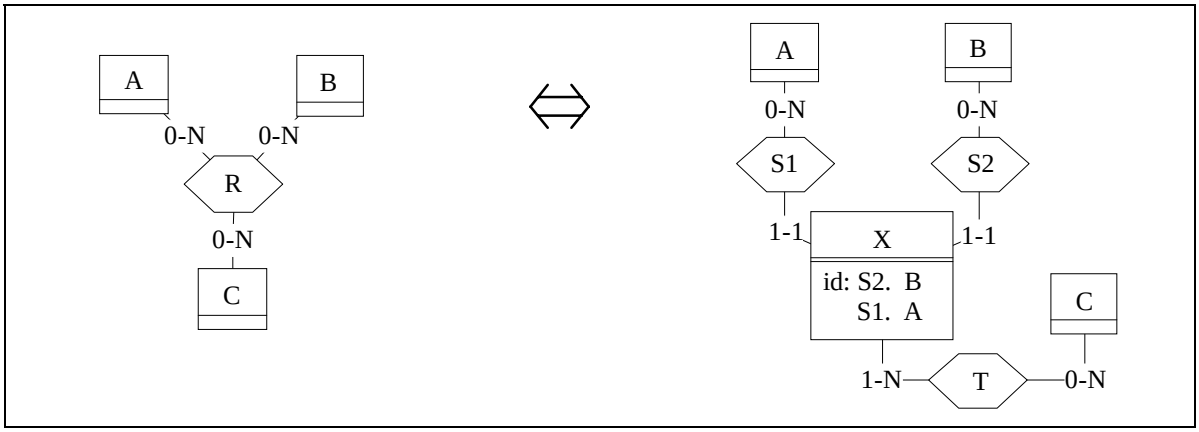
## Variants




A
B
C
0-10
R
R1
0-1
0-N
RB
RA
RC
1-1


A
B
C
0-N
R
R1
R2
id: C
id: RC.C
0-1
0-N
RB
RA
RC
1-1


A
B
0-N
r1
r2
R
id: r2.B
r1.B
RA.A
RA
1-1




*Analysis* : can be expressed by an extension-decomposition transformation where J (here entity type C) is not empty :

`(X,{S1,S2},T)` ← **`ext-dec`**`(R,{{A},{B}})`

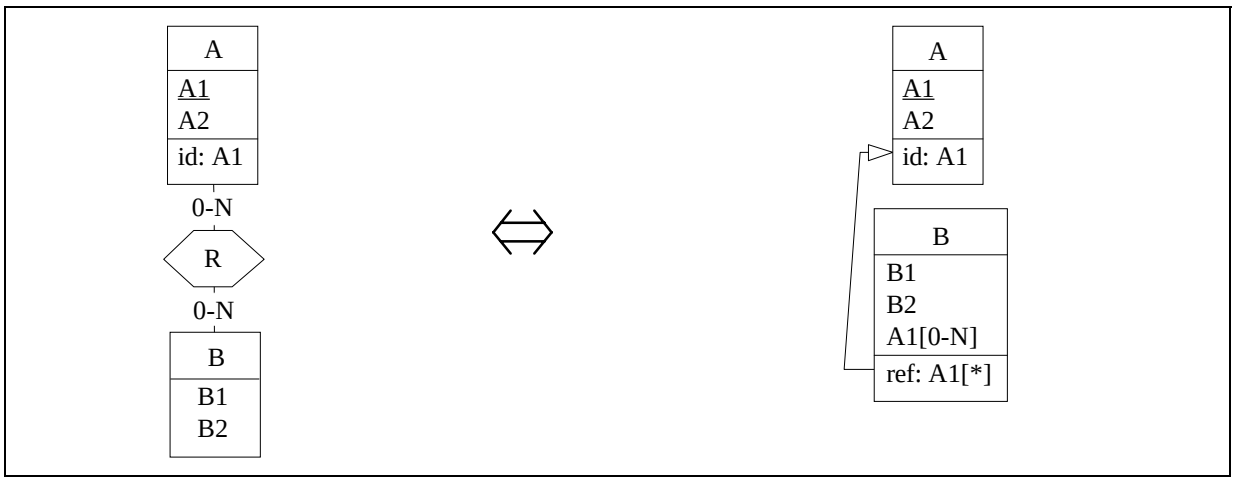
# 2. Transforming a Rel-type into a Foreign key

## Informal description

A binary rel-type R is transformed into a foreign key associated to entity type B.

Reference : [HAINAUT,90]

## General pattern




{A1} ← **RT-to-FK**(R,B)

## GER analysis

```
desc-of-A(A,A1,A2)
desc-of-B(B,B1,B2)
R(A,B)
desc-of-A[A]=A
desc-of-B[B]=B
```

⇕ R' ← **comp**(desc-of-A,R,{A1},{A})

```
desc-of-A(A,A1,A2)
desc-of-B(B,B1,B2)
R'(A1,B)
desc-of-A[A]=A
desc-of-B[B]=B
R'[A1] ⊆ desc-of-A[A1]
```

⇕ R" ← **unnest**$^{-1}$(R',A1)

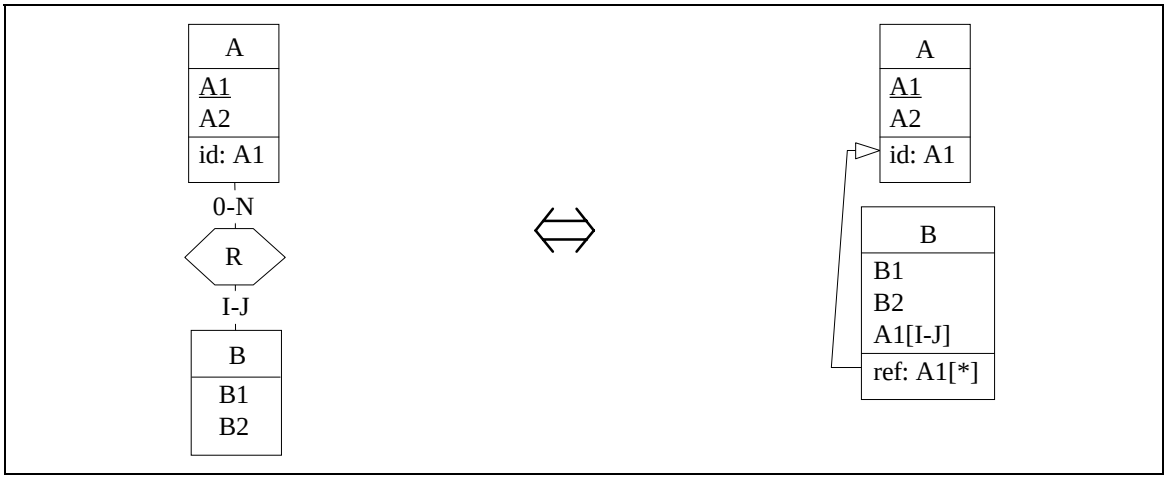
```
desc-of-A(A,A1,A2)
desc-of-B(B,B1,B2)
R"(A1[1-N],B)
desc-of-A[A]=A
desc-of-B[B]=B
∪R"[A1] ⊆ desc-of-A[A1]
```

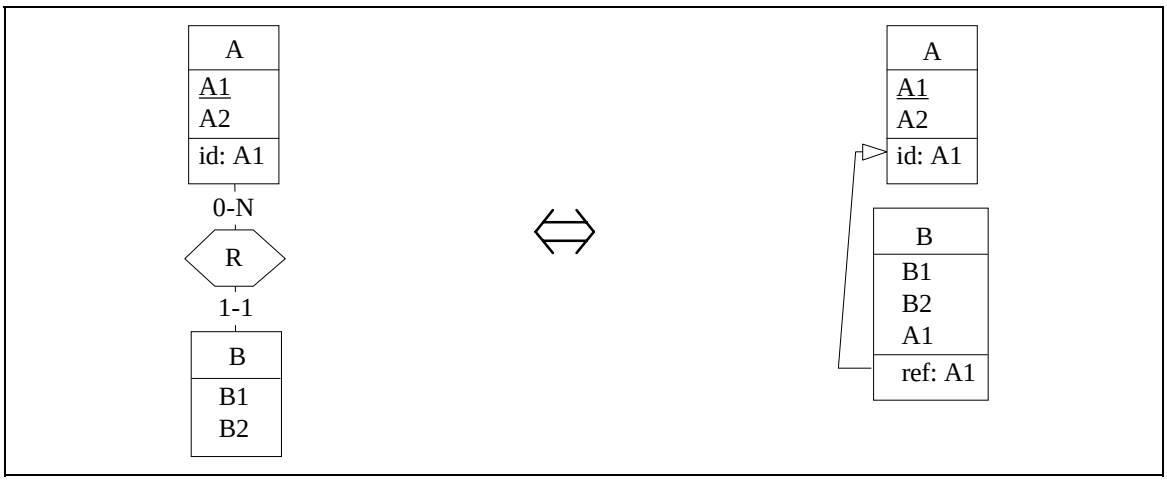
⇕ (desc-of-B',B) ← **PJ**$^{-1}$(desc-of-B,R",{B},{B)}

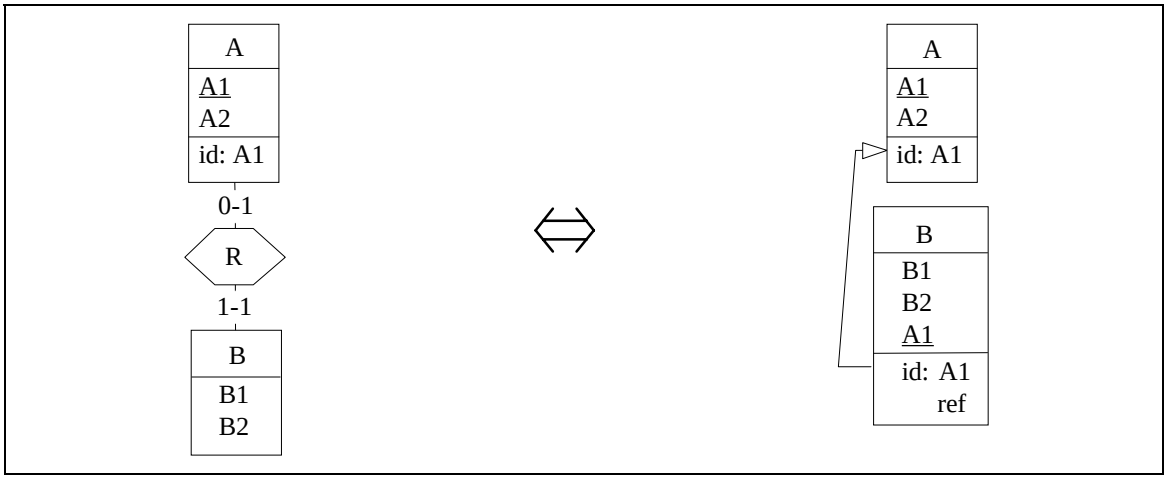
```
desc-of-A(A,A1,A2)
desc-of-B'(B,B1,B2,A1[0-N])
desc-of-A[A]=A
desc-of-B'[B]=B
∪desc-of-B'[A1] ⊆ desc-of-A[A1]
```

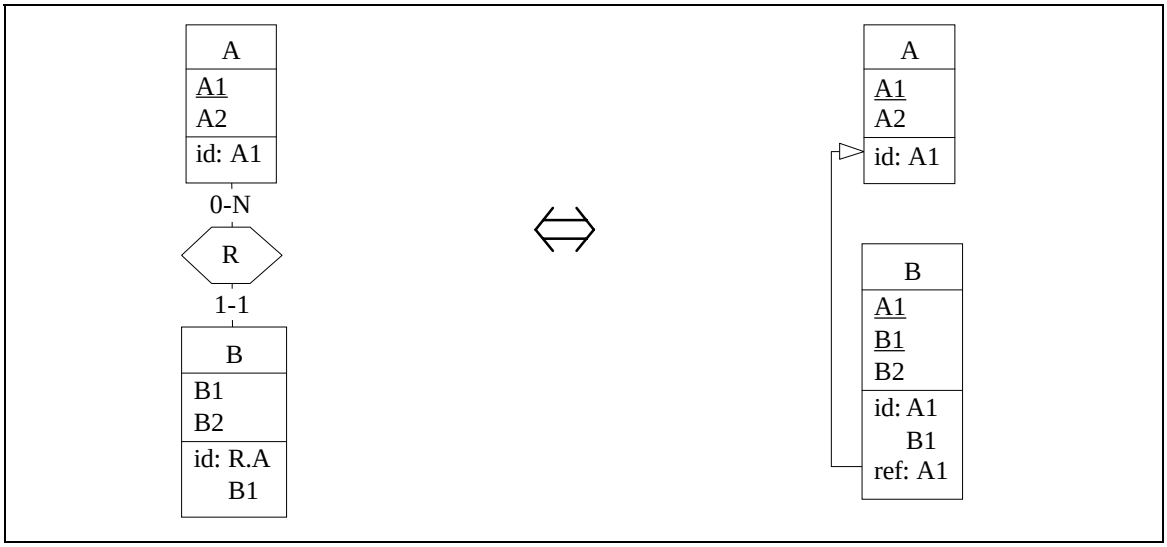
Conclusion : the **RT-to-FK** transformation is *symmetrically reversible.*

# Variants


A
A1
A2
id: A1
0-N
R
I-J
B
B1
B2
A
A1
A2
id: A1
B
B1
B2
A1[I-J]
ref: A1[*]


A
A1
A2
id: A1
0-N
R
1-1
B
B1
B2
A
A1
A2
id: A1
B
B1
B2
A1
ref: A1


A
A1
A2
id: A1
0-1
R
1-1
B
B1
B2
A
A1
A2
id: A1
B
B1
B2
A1
id: A1
ref


A
A1
A2
id: A1
0-N
R
1-1
B
B1
B2
id: R.A
B1
A
A1
A2
id: A1
B
A1
B1
B2
id: A1
B1
ref: A1

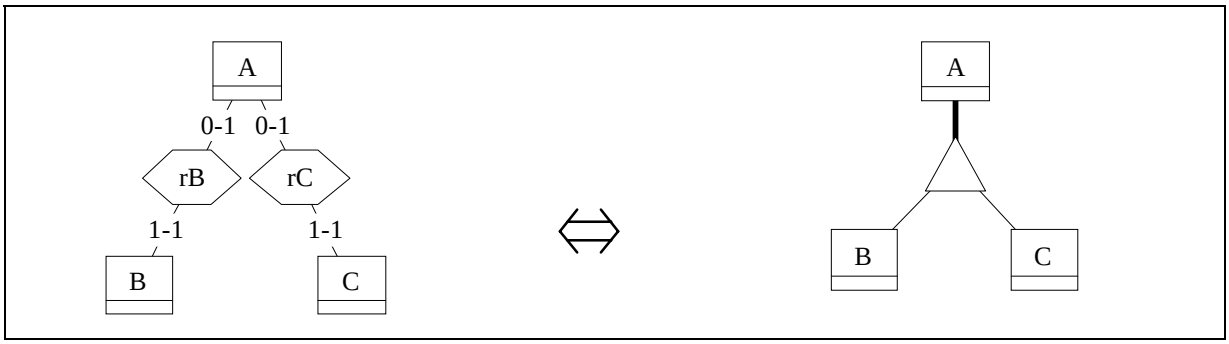
# 3. Transforming one-to-one Rel-types into IS-A relations

## Informal description

The common member (A) of a set of one-to-one rel-types (rB and rC) is transformed into a supertype of the other members (B,C). In short, the rel-types are transformed into IS-A relations;

Reference : [BATINI,92], [HAINAUT,94a]

## General pattern




Remark : must be completed according to the constraint (exclusive, at-least-one) in which rB, rC are involved.

```
() ← RT-to-ISA({(B,rB),(C,rC)})
```

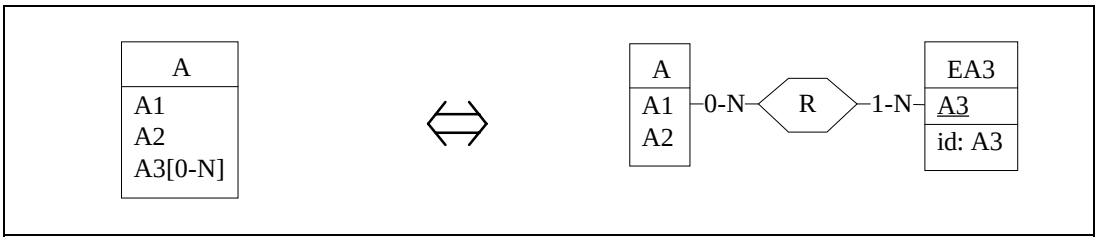
# 1. Transforming an Attribute into an Entity type

## Informal description

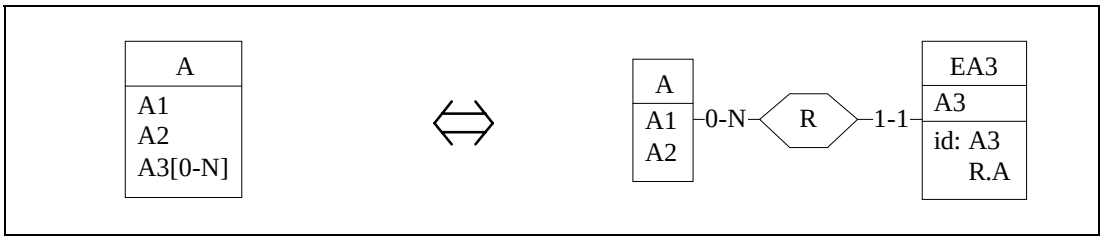
An attribute is expressed as an independent entity type. There are two basic variants, according to whether each new entity represents a distinct value of the attribute (*Value representation*), or an instance of it (*Instance representation*).

Reference : [HAINAUT,91a]

## General pattern




(EA3,R) ← **Att-to-ET/Value**(A,{A3})




(EA3,R) ← **Att-to-ET/Instance**(A,{A3})

## GER analysis (common)

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

⇕ (desc-of-A',R) ← **PJ**(desc-of-A,{A},{A3})

```
desc-of-A'(A,A1,A2)
R(A,A3[1-N])
desc-of-A'[A] = A
```

⇕ R' ← **unnest**(R,A3)

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

## GER analysis (Value representation)

⇳ (EA3,{desc-of-EA3},R") ← **ext-dec**(R',{A3})

```
EA3 : entities
desc-of-A'(A,A1,A2)
desc-of-EA3(EA3,A3)
R"(A,EA3)
desc-of-A'[A] = A
desc-of-EA3[EA3] = R"[EA3] = EA3
```

*Conclusion* : **Att-to-ET/Value** is an SR-transformation.

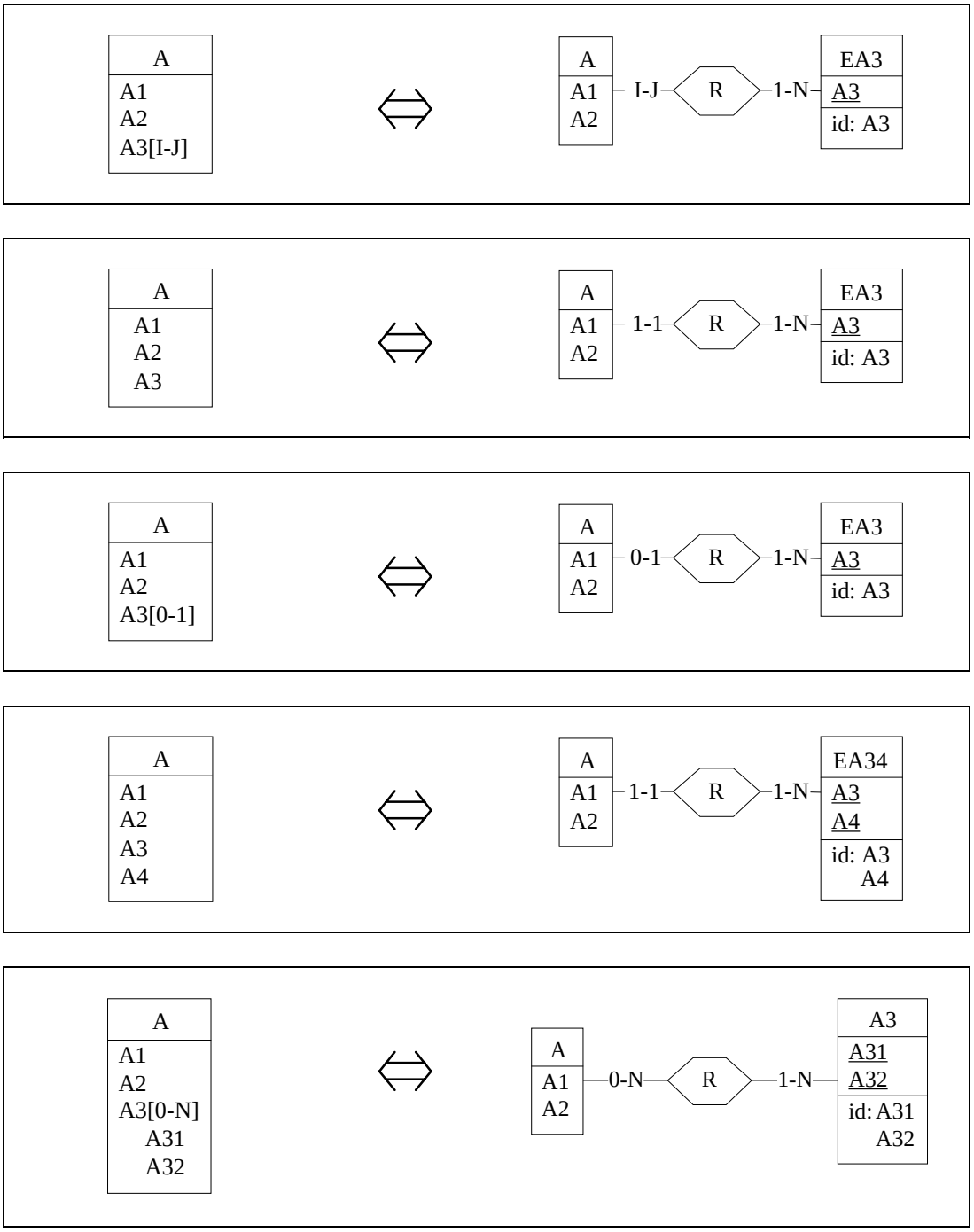
## GER analysis (Instance representation)

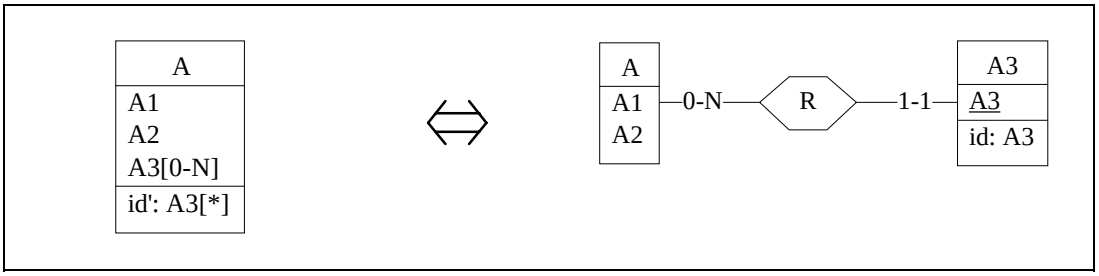
⇳ (EA3,{desc-of-EA3,R"}) ← **ext-dec**(R',{A,A3})

```
EA3 : entities
desc-of-A'(A,A1,A2)
desc-of-EA3(EA3,A3)
R"(EA3,A)
desc-of-A'[A] = A
desc-of-EA3[EA3] = EA3
R"*desc-of-EA3:A,A3 → EA3
```

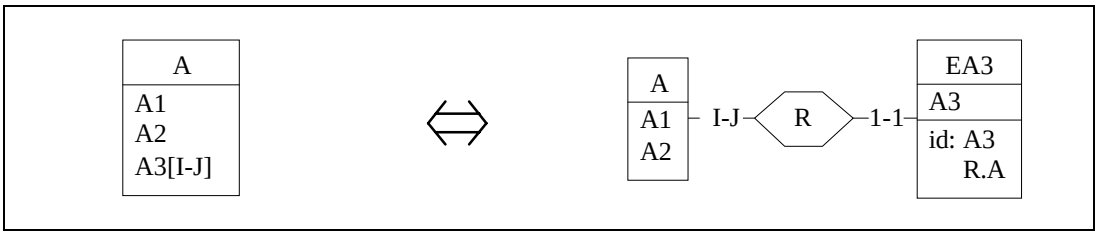
*Conclusion* : **Att-to-ET/Instance** is an SR-transformation.

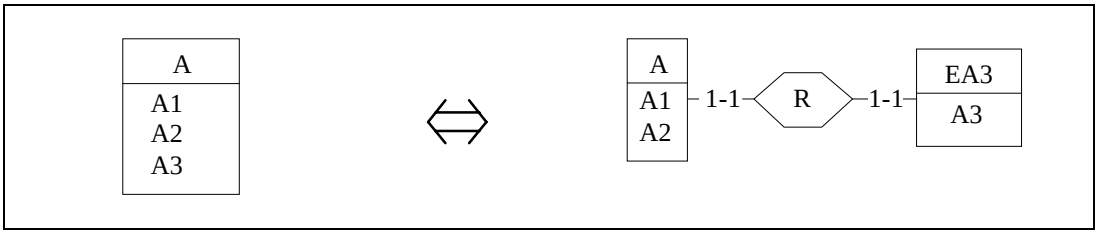
# Variants (Value representation)

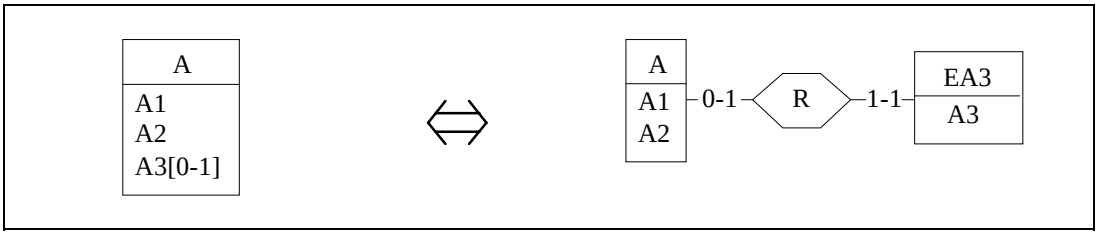


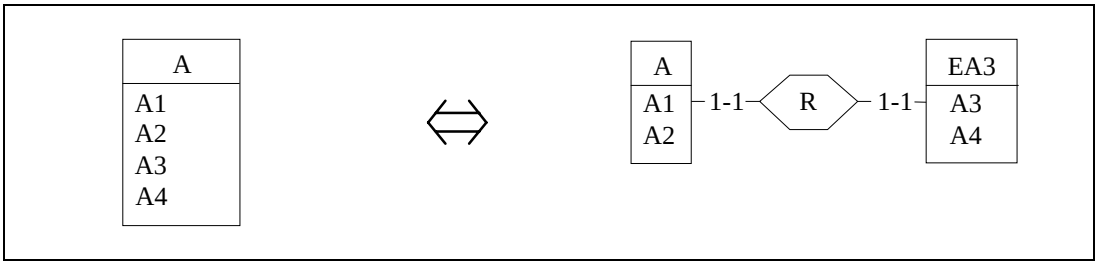


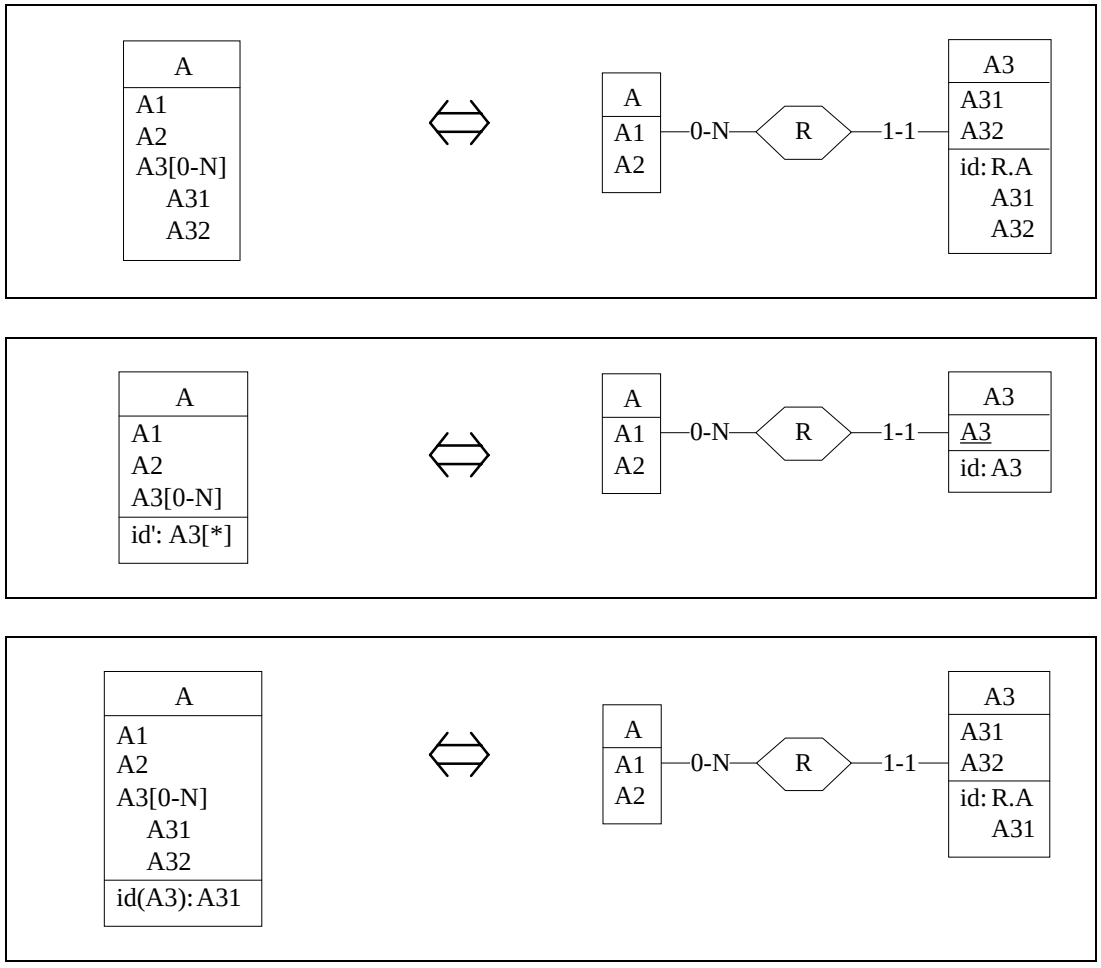

## Variants (Instance representation)



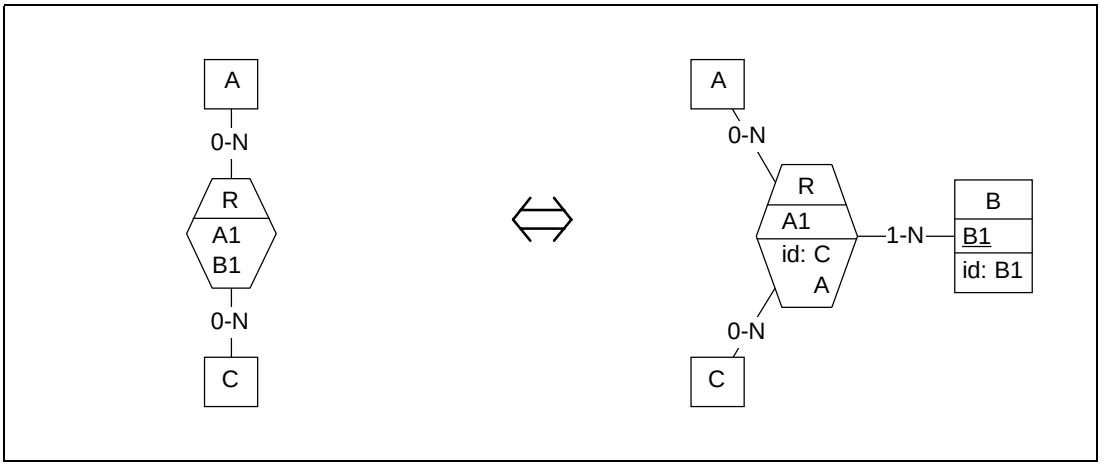




A
A1
A2
A3[0-N]
A31
A32
A
A1
A2
0-N
R
1-1
A3
A31
A32
id: R.A
A31
A32
A
A1
A2
A3[0-N]
id': A3[*]
A
A1
A2
0-N
R
1-1
A3
A3
id: A3
A
A1
A2
A3[0-N]
A31
A32
id(A3): A31
A
A1
A2
0-N
R
1-1
A3
A31
A32
id: R.A
A31

## Extension : rel-type attribute

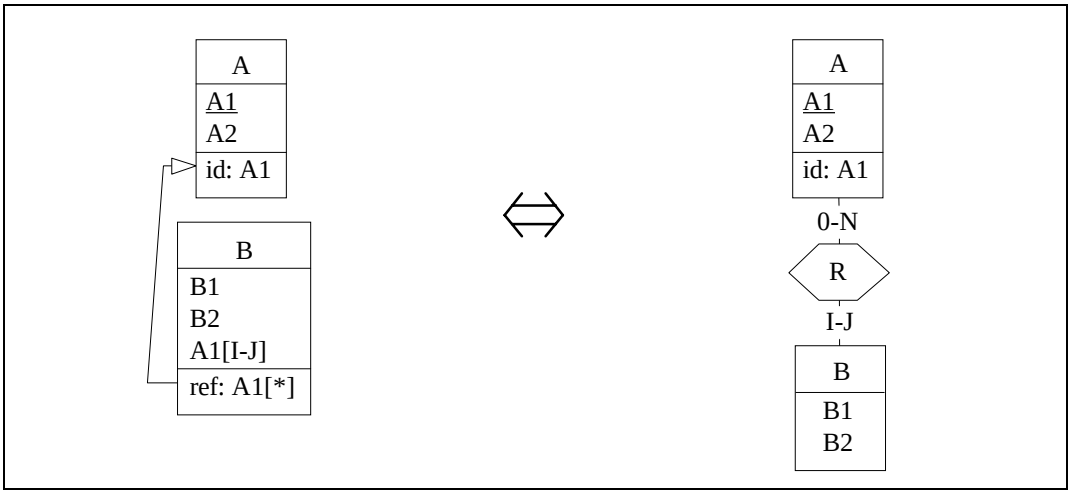


# 2. Transforming a Foreign key into a Rel-type

## Informal description

A foreign key {A1} from B to A is transformed into rel-type R between A and B;

## General pattern




R ← **FK-to-RT**(B,{A},A)

{R,role-of-A,role-of-B} ← **FK-to-RT**(B,{A},A)

## Analysis

**FK-to-RT** = **RT-to-FK**$^{-1}$

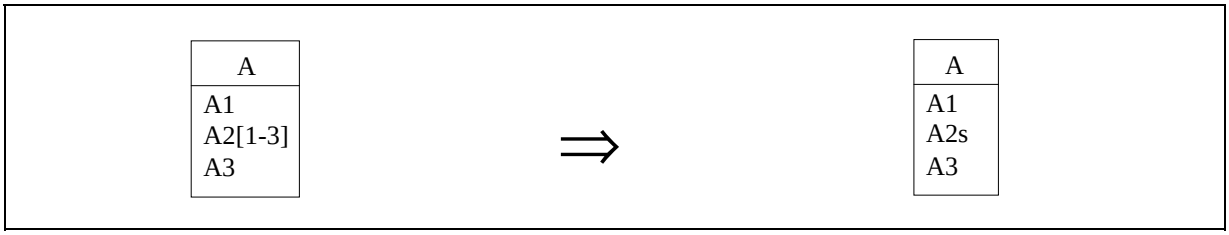
*Conclusion* : according to 5.3, the transformation is symmetrically reversible

# 6. Concatenating a Multivalued Attribute

## Informal description

Multivalued attribute A2 is replaced by an atomic attribute each value of which being made of the concatenation of the values of the origin A2 attribute.

## General pattern




A2s ← **MultAtt-to-SingleAtt**(A,A2)

## Analysis

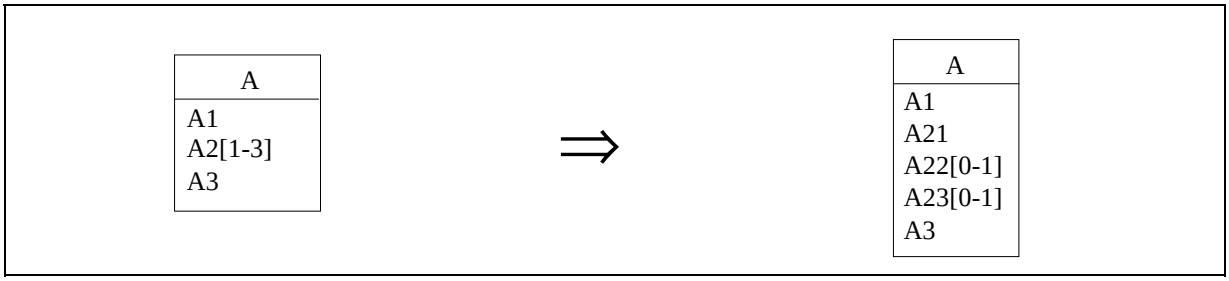
This is a pragmatic transformation which is not fully reversible. Indeed, the concatenated attribute induced an ordering relation on the A2 values which did not exist in the origin schema. The transformation is simply reversible from left to right.

# 3. Transforming a Multivalued Attribute into Serial Attributes

## Informal description

Multivalued attribute A2 is replaced by a series of single-valued attributes A21, A22, A23, etc.

## General pattern




{A21,A22,A23} ← **MultiAtt-to-SerialAtt**(A,A2)

## GER analysis

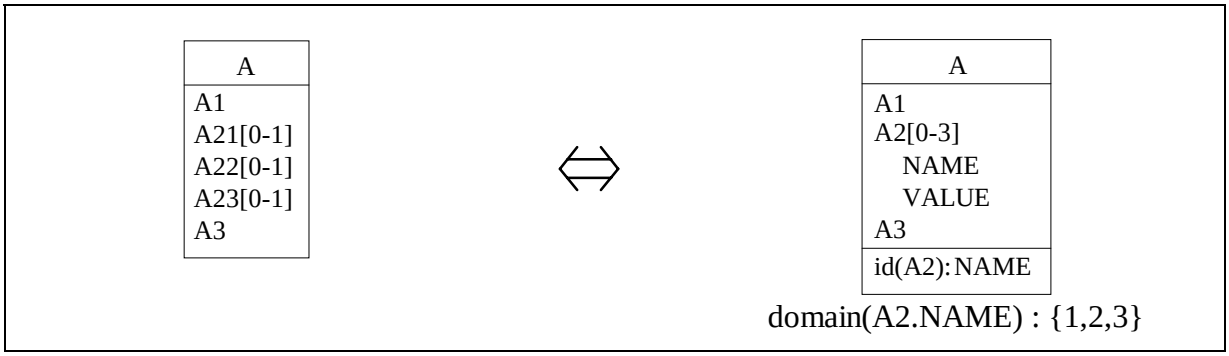
Still a pragmatic transformation which is not fully reversible. Indeed, the serial attributes define, through their names, a distinct role for each A2 value which did not exist in the origin schema. In addition, the uniqueness constraint on the A2 values is lost in the final schema. The transformation is simply reversible from left to right.

# 4. Transforming Serial Attributes into a Multivalued Attribute

## Informal description

The serial attributes {A21, A22, A23} are transformed into multivalued attribute A2; each value of the latter includes a value, and a distinct name for it.

## General pattern




(A2,NAME,VALUE) ← **SerialAtt-to-MultiAtt**(A,{A21,A22,A23})

## Analysis

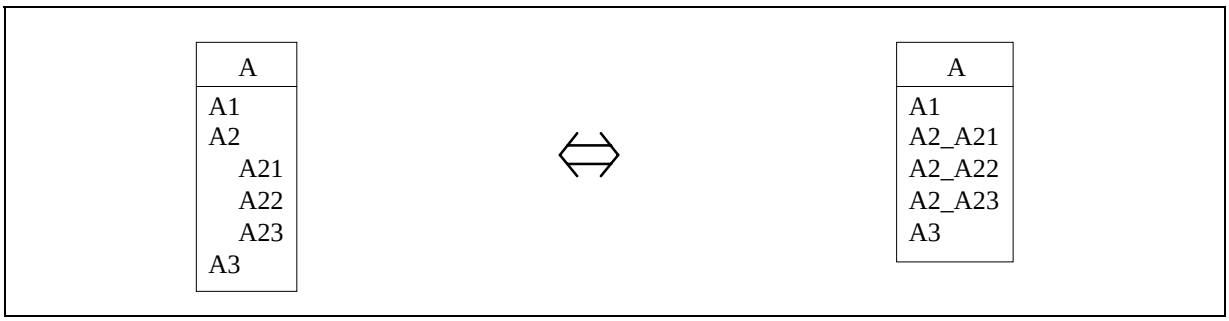
The serial attributes are considered as a ***list*** *multivalued attribute*, and transformed according to the corresponding definition. It is an SR-transformation.

# 5. Disaggregating a Compound Attribute

## Informal description

Compound attribute A2 is replaced by its components. It is an SR-transformation.

## General pattern



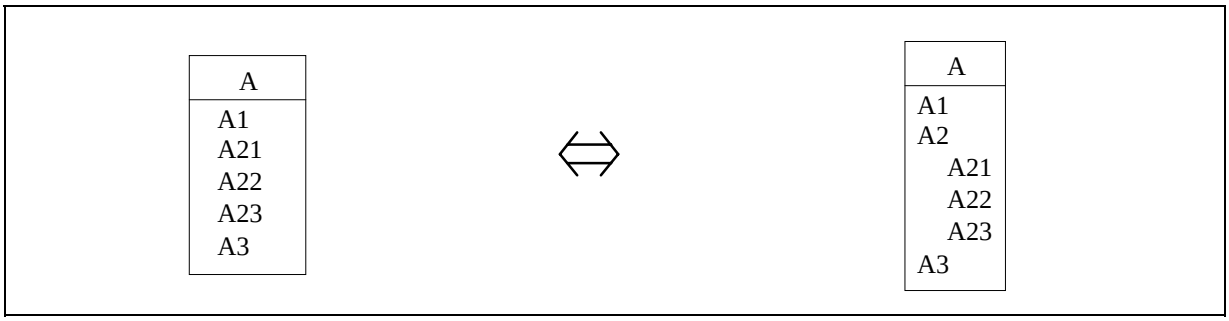

{A2_A21,A2_A22,A2_A23} ← **Disaggregate**(A,A2)

# 6. Aggregating a list of Attributes

## Informal description

A list of attributes are grouped into a new compound attribute. The attributes must have the same parent object. It is an SR-transformation.

## General pattern



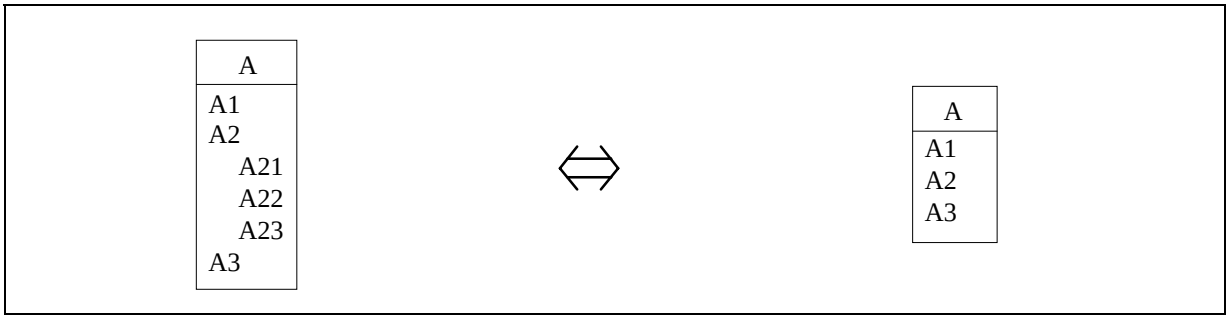

A2 ← **Aggregate**(A,{A21,A22,A23})

# 7. Concatenating a Compound Attribute

## Informal description

Compound attribute A2 is replaced by the concatenation of its components. It is an SR-transformation.

## General pattern




() ← **CompConcat**(A,A2)

## Analysis

An interesting transformation : though symmetrically reversible, it produces a schema that obviously is less informative than its origin.

# Part 6

## OTHER ER/OR TRANSFORMATIONS

**Elementary**, **semantics-preserving** transformations form the most respectable members of their class. However, these characteristics are not always possible, nor sometimes desirable.

We will examine some other categories of transformations, namely,

- **simply reversible** transformations, which have no reversible inverse,
- **non-reversible** transformations, such as *delete* and *modify*,
- **compound** transformations, formed as a chain of other transformations
- **redundant** transformations, which leave some trace of the source construct in the target schema
- **transformation plans**, which are complex arrangements of transformations, in order to make the source schema satisfy some complex objectives or requirements.

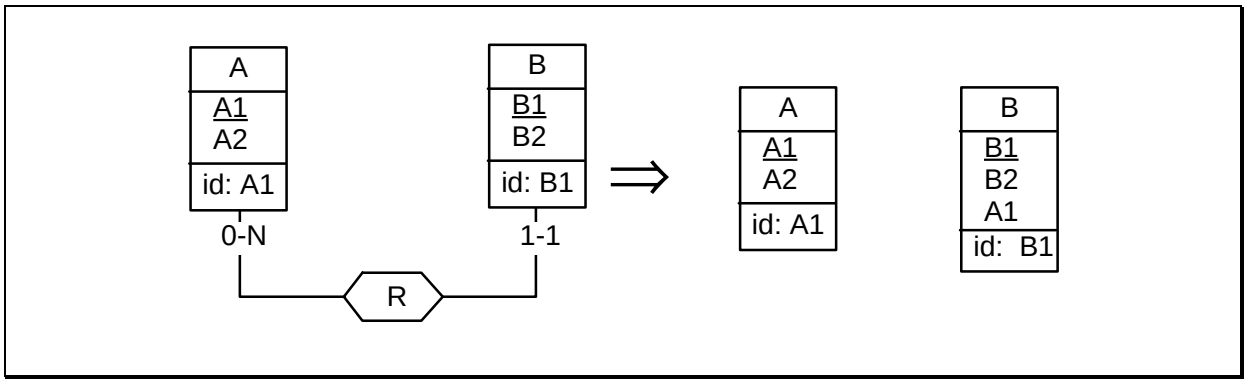
As observed in section 2.3 (case studies), an R-transformation is most often an incomplete SR-transformation. Generally, a constraint of the postcondition has been discarded.

Such transformations are observed in typical degenerated situations :

- poor methodologies and practices;
- simplification (some constraints are fairly complex);
- the target DBMS does not support such constraints;
- the target DBMS construct used to translate the source structure is not fully adequate (e.g. implementing a multivalued attribute with an array does not garantee value uniqueness, but introduces an unneeded ordering relation);
- checking such constraints is resource-consuming.

Note that **inserting a new object** in a schema typically is an R-transformation : it is always possible to delete the created object, but not conversely (at least through a generic transformation).

**Example :**

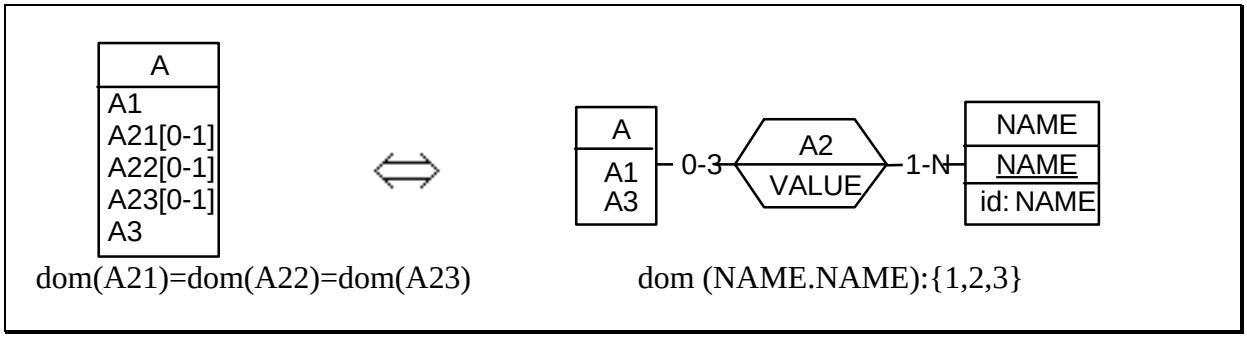


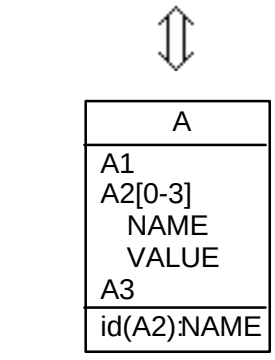
A non-reversible transformation has no inverse. This is the case for all schema manipulation actions that **delete** or **modify** an existing object : entity type, rel-type, atribute, domain, constraint, etc.

This class also includes composite operations (compound transformations and transformation plans) that include at least one non-reversible action.

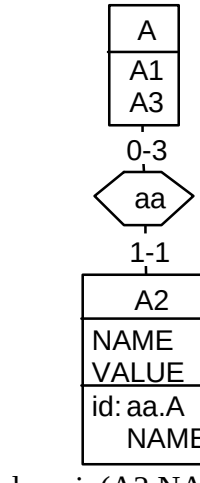
By chaining several transformations, complex operators can be defined :

## Example 1

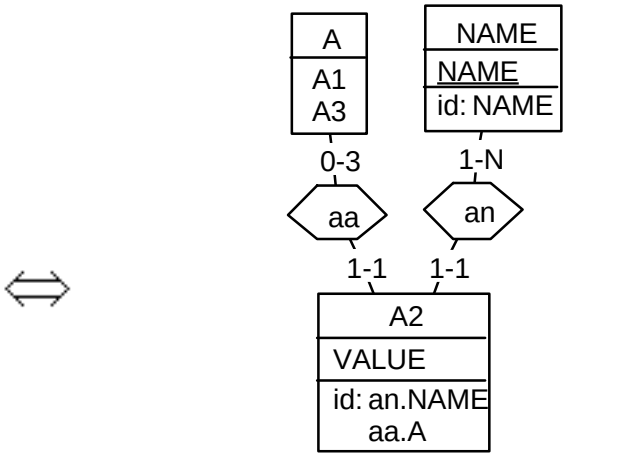



domain(A.A2.NAME):{1,2,3}

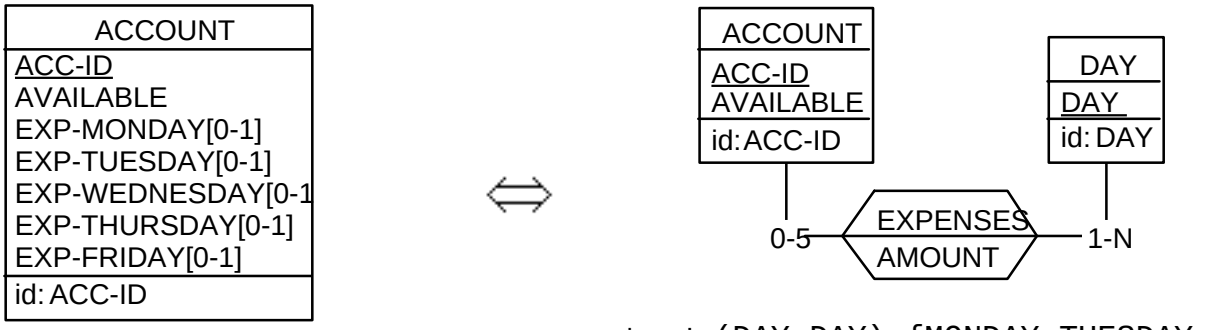



domain(A2.NAME):{1,2,3}



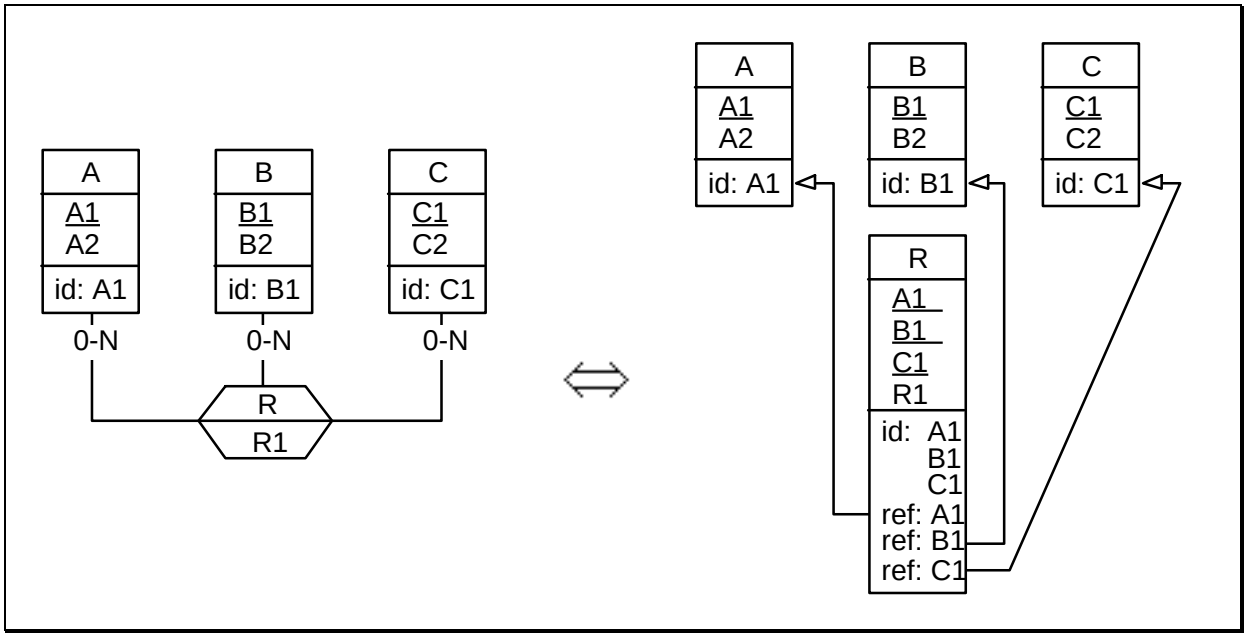

domain(NAME.NAME):{1,2,3}

# Application 1




```
domain(DAY.DAY):{MONDAY,TUESDAY,
     WEDNESDAY,THURSDAY,FRIDAY}
```

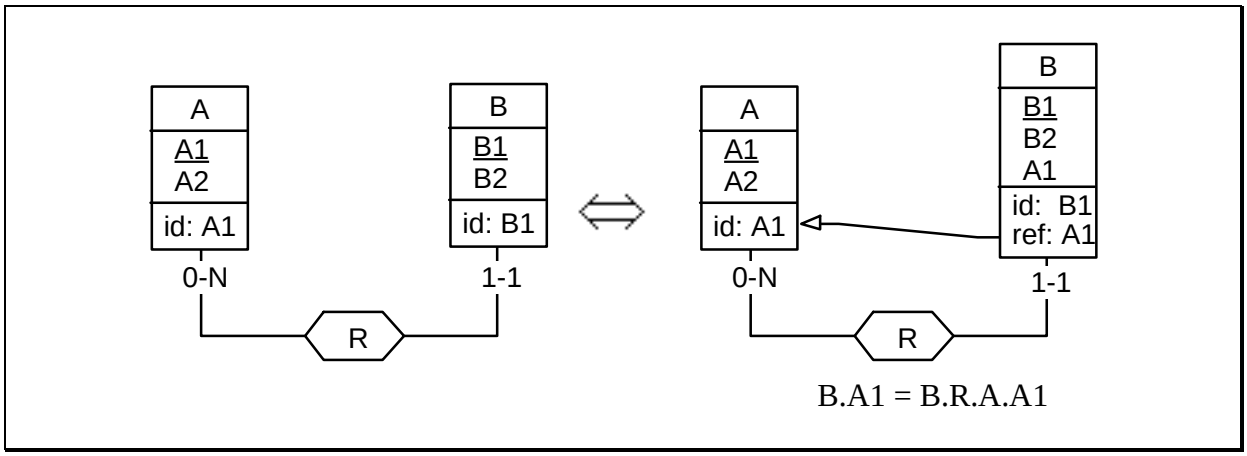
# Example 2



In a redundant transformation, the source construct, or part of it, is maintained in the target schema (section 2.5). This is a special case of structural redundancy, a popular practice when optimizing a schema [HAINAUT,93b], [HAINAUT,94a].

A redundant transformation must generate a (too often) duplication, or derivation, integrity constraint.

## Example



The transformations discussed so far are basic tools only, even when they are chained to form compound transformations (section 6.4). Completely solving complex problems raise the questions of *what transformations to apply*, *on what objects* and *in what order*.

This defines a **transformation plan**, i.e. an algorithm composed of steps of the following form (O is an object type and P is a predicate) :

$$\texttt{for each } o \in O, \texttt{ such that } P(o), \texttt{ do } T_{\Sigma i}(o);$$

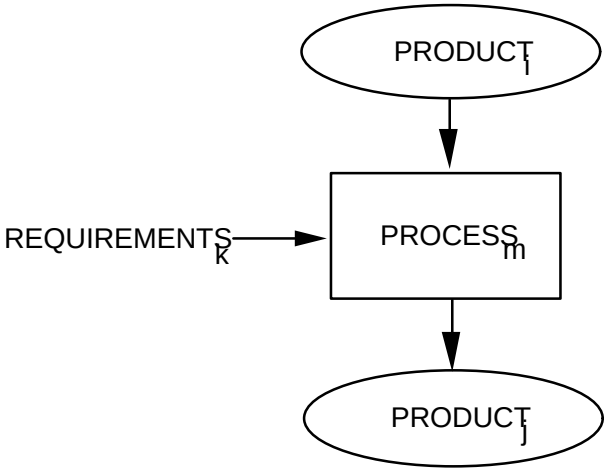
The following **transformation script** describes a transformation plan that *produces relational schemas equivalent to source ER schemas* [HAINAUT,92a].

- Let S be the current schema;
- for each rel-type R such that ((R is N-ary) or (R has attributes)) do
  `(...) ← RT-to-ET(R)`
- for each rel-type R such that ((R is binary) and (R is many-to-many)) do
  `(...)← RT-to-ET(R)`
- do
  for each attribute A such that ((A is at level 1 in E) and (R is compound)) do
  `(...)← Disaggregate(E,A)`
  for each attribute A such that ((A is at level 1 in E) and (R is multivalued)) do
  `(EA,RA) ← Att-to-ET/Instance(E,A)`
  until there is no more compound or multivalued attributes
- do
  for each rel-type R(E1,E2) such that (R is one-to-many) do
  `(...)← RT-to-FK(R,E2)`
  until no rel-types have been transformed
- for each entity type E such that ((there exists R(E,E2)) and (R is one-to-many) and (E has no identifier)) do
  `E-ID ← Tech-ID(E)`
- for each rel-type R(E1,E2) such that (R is one-to-many) do
  `(...)← RT-to-FK(R,E2)`

## Further discussion

A transformation plan implements an engineering process such as those described in section 7 : Normalization, DBMS translation, Optimization, Reverse engineering for instance.

Each such **process** can be perceived as a transformation process which produces **target products** (generally schemas or texts) from **source products**. This transformation is most often guided by a specific set of objectives, or **requirements** that the source products do not necessarily satisfy, but that the target products have to satisfy [HAINAUT,94b] :




Very often, the requirements can be defined as syntactic or structural rules, and therefore can be expressed by structural predicates.

To analyse a bit further the role of the transformations in engineering processes, and the reasoning at the basis of transformation plan development, let us consider the following limited context [HAINAUT,92a] :

- R is the set of requirements of process P,
- S is the input schema of the process,
- C is a construct of schema S,
- r is a rule of R such that : $\neg\, r(C)$
- Σ is the set of available transformations.

C is a construct of schema S that does not satisfy requirements R (i.e. $\neg R(C)$), and that must be transformed into C' such that r(C').

An obvious elementary strategy is as follows :

(1) select a transformation Σ of Σ such that $P_\Sigma(C)$ & $(Q_\Sigma \Rightarrow r)$
(2) replace C with $T_\Sigma(C)$ in S

Potential problems may arise that require more sophisticated strategies. Let's examine some of them.

***P1 : Construct C may violate more than one rule.***

*Strategy* : Let R' be the set of rules that C doesn't satisfy. Choose a rule r in R' such that there exists a transformation Σ in **T** such that : $T_\Sigma(C)$ violates as few rules of R as possible. This strategy may generate a set of solutions.

***P2 : More than one transformation satisfies : P(C) & (Q Þ r)***

*Strategy* : the selection of Σ can be done either arbitrarily, or according to other rules. In the latter case, P generates a set of solutions. The final selection will be done according to other kinds of requirements, i.e. in another engineering process.

***P3 : No transformations satisfy P(C)***

*Diagnostic* : either the transformation set Σ is not powerful enough or the requirement cannot be satisfied.
*Strategies* : extend the set of transformations, keep construct C as it is or discard C.

***P4 : No transformations satisfy : Q Þ r***

*Strategy* : choose a transformation $\Sigma$ such that $P_{\Sigma}(C)$, then select another transformation $\Sigma'$ such that : $(Q_{\Sigma} \Rightarrow P_{\Sigma'})$ & $(Q_{\Sigma'} \Rightarrow r)$; if the latter cannot be satisfied, iterate the process. Example : RE-to-RT + RT-to-FK in RDBMS-translation. This strategy may generate a set of solutions.

***P5 : T(C) violates another rule that C satisfies***

This problem can be local, i.e. it concerns the termination of the process, or it can be global, such as when the current process may destroy the effect (i.e. the satisfaction of a set of requirements) of a former process. This problem still has no general solution.

These problems may induce the production of a large solution space. In such a situation, the concept of *target products* must be replaced by that of *set of equivalent target products* that must be explored according to other criteria. A higher-level strategy must be defined to manage this space and reduce it to one solution.

Anyway, for most engineering processes, the proposed transformation plan can be asked to satisfy some requirements :

- **termination** : the plan must terminate for any arbitrary schema
- **equivalence** : the target schema must be equivalent to the source schema
  - **completeness** : all the semantics (or other properties) must be preserved
  - **minimality** : ... and only it.
- **compliance** : the target schema must satisfy the process requirements
- **idempotence** : applying the plan on the target schema does not change the latter

# Part 7

# APPLICATIONS

Schema transformations can prove useful formal and practical tools in almost every database engineering activity.

In addition, this naturally leads to greater consistency and interrelationship of all these activities : it shows that, for instance, such activities as database design, database reverse engineering, schema integration, database conversion, federated databases share common problems, reasonings and solving techniques.

In each application domain, the transformations play an important role, but are in no way the only technique allowing to solve the problem.

**Structure**

- Objective/description of the application; some references
- The script transformation plan to solve the problem (where relevant)
- A graphical example
- The transformation script history which solved the example

**Objective**

(Loosely specified) process which tries to give a conceptual schema definite qualities such as readability, conciseness, minimality, expressiveness, normality (in the relational meaning), compliancy to corporate standards, etc.

Will appear in Conceptual Database Design and in Database Reverse Engineering.

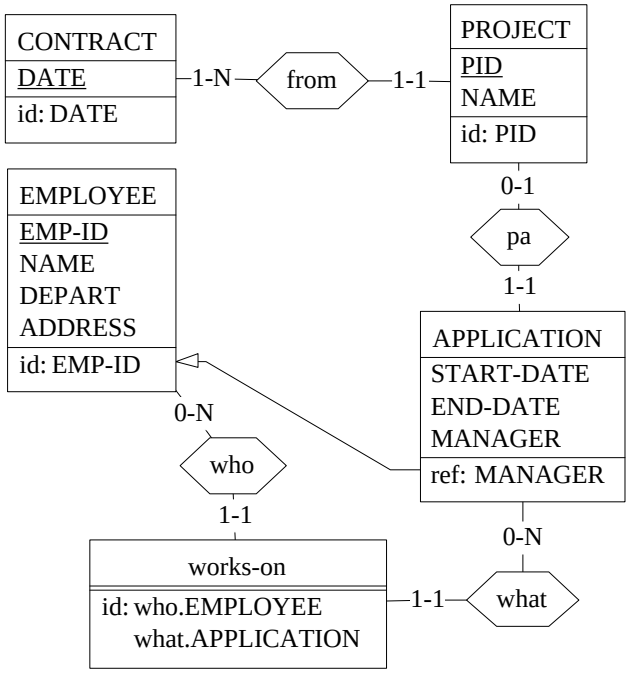
References : [LING,85], [BATINI,92], [HAINAUT,94a], [TEOREY,94], [RAUH,95]

**Script**

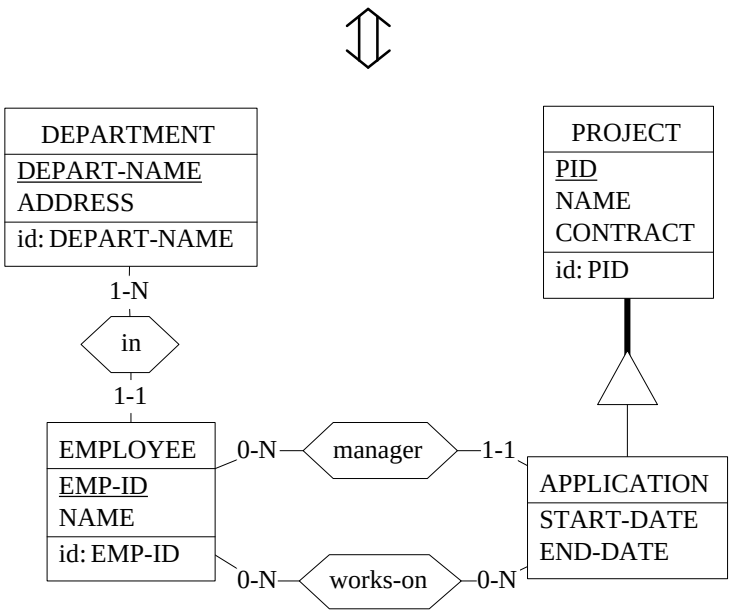
1. Transform foreign keys into Rel-types
2. Transform Relationship Entity types into Rel-types
3. Transform Attribute Entity types into attributes
4. Transform the components of non-Key-FD of Entity types into Entity types
5. Decompose Rel-types in which non-Key-FD hold
6. Express one-to-one Rel-types as IS-A relations (where pertinent)

Etc

## Example




EMPLOYEE: DEPART → ADDRESS



**Script history of the example**

manager ← FK-to-RT(APPLICATION,{MANAGER},EMPLOYEE)

works-on ← ET-to-RT(works-on)

CONTRACT ← ET-to-Att(CONTRACT)

(DEPARTMENT,in) ← Att-to-ET/Value(EMPLOYEE,{DEPART,ADDRESS})

() ← RT-to-ISA({(APPLICATION,pa)})

**Objective**

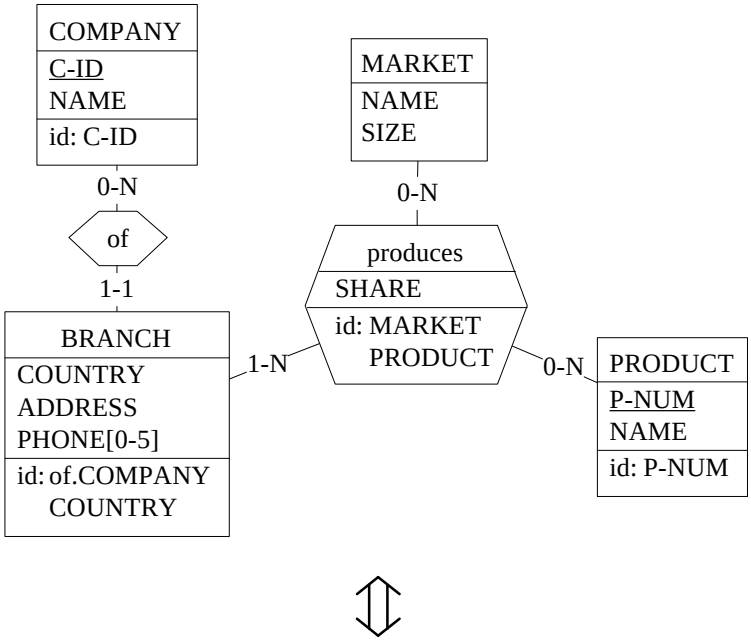
Process through which a conceptual schema is translated into a logical schema according to a specific DMS model. Appears in Logical Database Design.

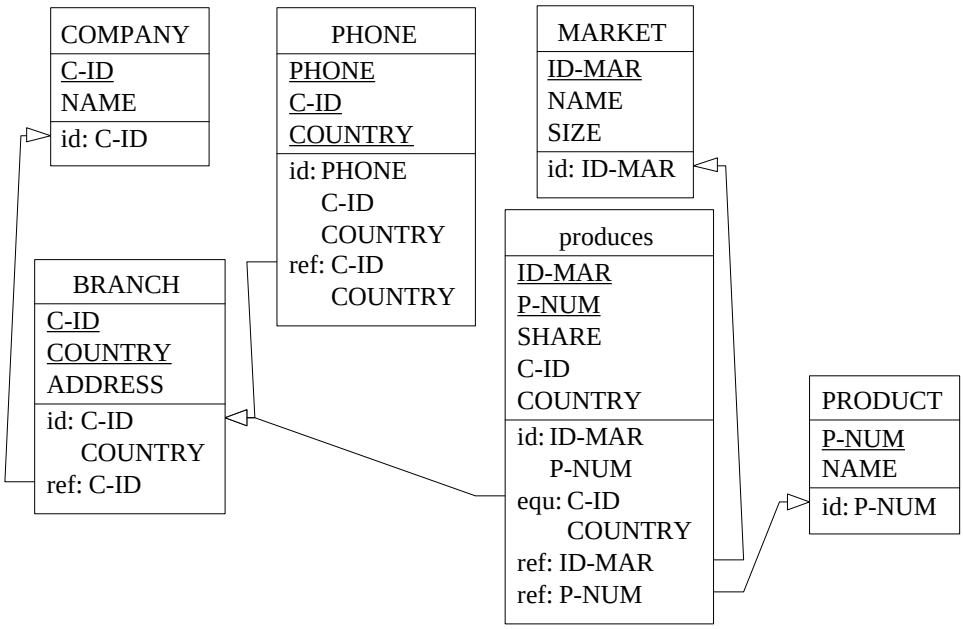
References : [HAINAUT,81], [BATINI,92], [HAINAUT,92a], [HAINAUT,94a]

**Script (for SQL, simplified)**

1. Transform each IS-A relation into a one-to-one Rel-type
2. Transform each complex Rel-type into an Entity-type
3. Transform each level-1 Multivalued Attribute into an Entity type (Instance repres.)
4. Disaggregate each level-1 Compound Attribute
5. Repeat steps 3 and 4 until there are no Multivalued and Compound Attributes left
6. Transform each Rel-type into a foreign key
7. Add a Technical Id to each Entity type which has prevented step 6 to operate successfully
8. Transform each Rel-type into a foreign key

## Example


COMPANY
C-ID
NAME
id: C-ID
0-N
of
1-1
BRANCH
COUNTRY
ADDRESS
PHONE[0-5]
id: of.COMPANY
COUNTRY
1-N
MARKET
NAME
SIZE
0-N
produces
SHARE
id: MARKET
PRODUCT
0-N
PRODUCT
P-NUM
NAME
id: P-NUM


COMPANY
C-ID
NAME
id: C-ID
PHONE
PHONE
C-ID
COUNTRY
id: PHONE
C-ID
COUNTRY
ref: C-ID
COUNTRY
MARKET
ID-MAR
NAME
SIZE
id: ID-MAR
BRANCH
C-ID
COUNTRY
ADDRESS
id: C-ID
COUNTRY
ref: C-ID
produces
ID-MAR
P-NUM
SHARE
C-ID
COUNTRY
id: ID-MAR
P-NUM
equ: C-ID
COUNTRY
ref: ID-MAR
ref: P-NUM
PRODUCT
P-NUM
NAME
id: P-NUM

**Script history of the example**

(produces,{(BRANCH,r1),(MARKET,r2),(PRODUCT,r3)})
← RT-to-ET(produces)

(PHONE,r4) ← Att-to-ET/Instance(BRANCH,{PHONE})

{C-ID} ← RT-to-FK(of,BRANCH)

{C-ID,COUNTRY} ← RT-to-FK(r1,produced)

*failure* ← RT-to-FK(r2,produced)

{P-NUM} ← RT-to-FK(r3,produced)

{C-ID,COUNTRY} ← RT-to-FK(r4,PHONE)

ID-MAR ← Tech-ID(MARKET)

{ID-MAR} ← RT-to-FK(r2,produced)

**Objective**

Merely carrying out physical tuning on a logical schema does not always produce satisfying performance. The designer must often restructure either the conceptual or the logical schema to gain better performance. This **optimization** process appears in Logical Database Design.

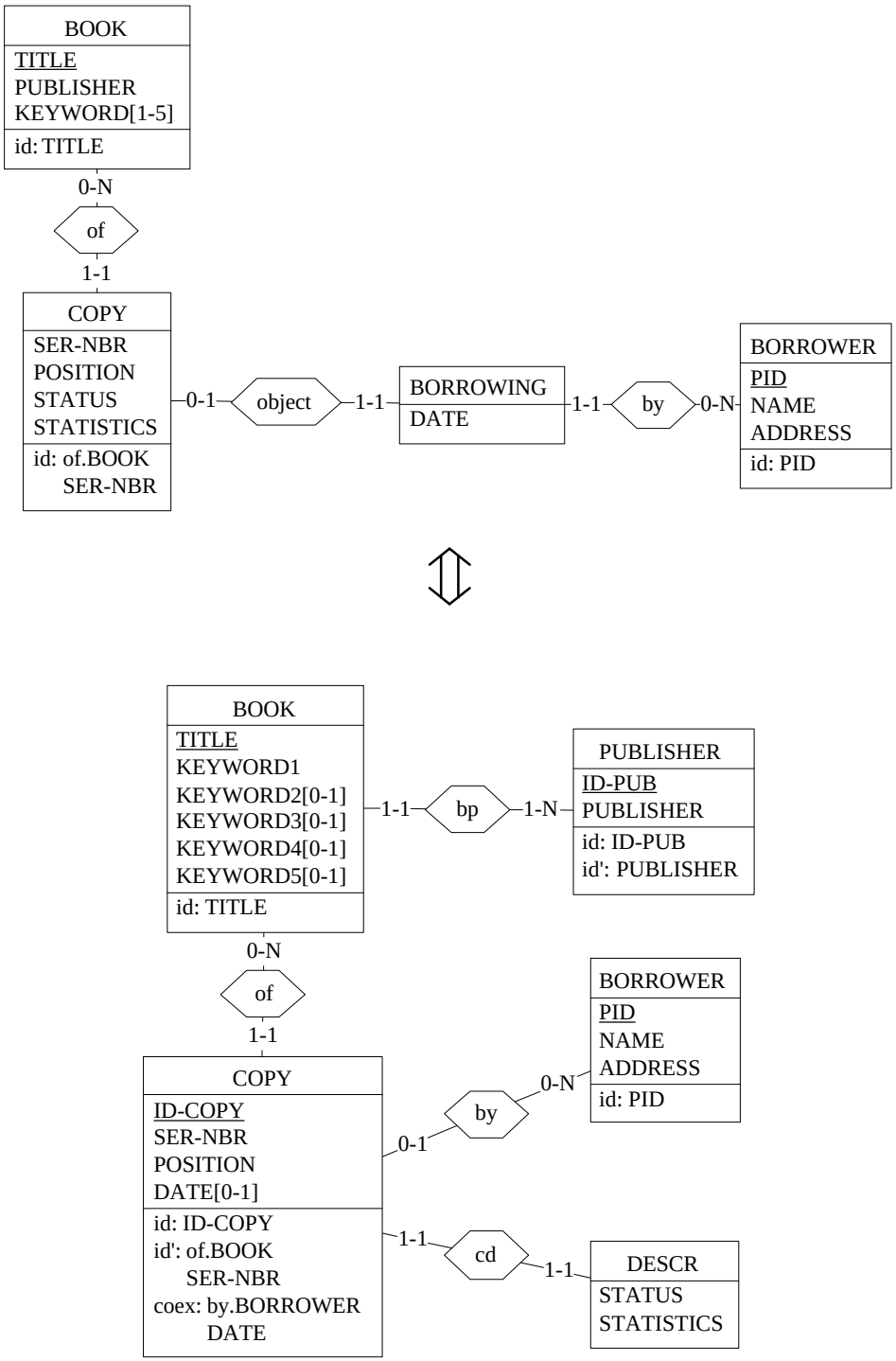
References : [SHASHA,92], [BATINI,92], [HAINAUT,92a], [HAINAUT,94a], [HALPIN,95]

**Script (simplified)**

1. Transform some Multivalued Attributes into Serial attributes (list of similar single-valued Attributes)
2. Transform some long Attributes into Entity types
3. Split some long Entity types
4. Merge some logically related Entity types
5. Replace long, complex or unstable primary identifiers by short, meaningless technical identifiers
6. Denormalize

Etc

## Example


BOOK
TITLE
PUBLISHER
KEYWORD[1-5]
id: TITLE
0-N
of
1-1
COPY
SER-NBR
POSITION
STATUS
STATISTICS
id: of.BOOK
SER-NBR
0-1
object
1-1
BORROWING
DATE
1-1
by
0-N
BORROWER
PID
NAME
ADDRESS
id: PID
BOOK
TITLE
KEYWORD1
KEYWORD2[0-1]
KEYWORD3[0-1]
KEYWORD4[0-1]
KEYWORD5[0-1]
id: TITLE
1-1
bp
1-N
PUBLISHER
ID-PUB
PUBLISHER
id: ID-PUB
id': PUBLISHER
0-N
of
1-1
BORROWER
PID
NAME
ADDRESS
id: PID
COPY
ID-COPY
SER-NBR
POSITION
DATE[0-1]
id: ID-COPY
id': of.BOOK
SER-NBR
coex: by.BORROWER
DATE
by
0-N
0-1
1-1
cd
1-1
DESCR
STATUS
STATISTICS

**Script history of the example**

{KEYWORD1,KEYWORD2,KEYWORD3,KEYWORD4,KEYWORD5} ← MultiAtt-to-SerialAtt(BOOK,KEYWORD)

(PUBLISHER,bp) ← Att-to-ET/Value(BOOK,{PUBLISHER})

(DESCR,cd) ← Split-ET(COPY,{STATUS,STATISTICS})

{DATE} ← Merge-ET(COPY,object)

ID-PUB ← Tech-ID(PUBLISHER)

ID-COPY ← Tech-ID(COPY)

**Objective**

Reverse engineering a database consists in recovering its conceptual schema. According to a general methodology proposed in [HAINAUT,93], this complex activity can be carried out in two processes : Data Structure Extraction (recovering the physical/logical schema) and Data Structure Conceptualization (converting this schema into conceptual specifications). This latter process, in turn, comprises several sub-processes that strongly rely on transformational techniques :

- Schema Untranslation (reversing the forward *DBMS translation* process)
- De-optimization (reversing the forward *Optimization* process)
- Conceptual Normalization (same as 7.2)

References : [HAINAUT,93], [HAINAUT,95a], [HAINAUT,95b]

see bibliography

**Script for Conceptualization (SQL structures, simplified)**

*1. Untranslation*

Transform all Foreign keys by Rel-types

etc

*2. Deoptimization*

Transform Serial attributes into Multivalued attributes

Normalize non-BCNF (or other non-higher-order forms) structures

etc

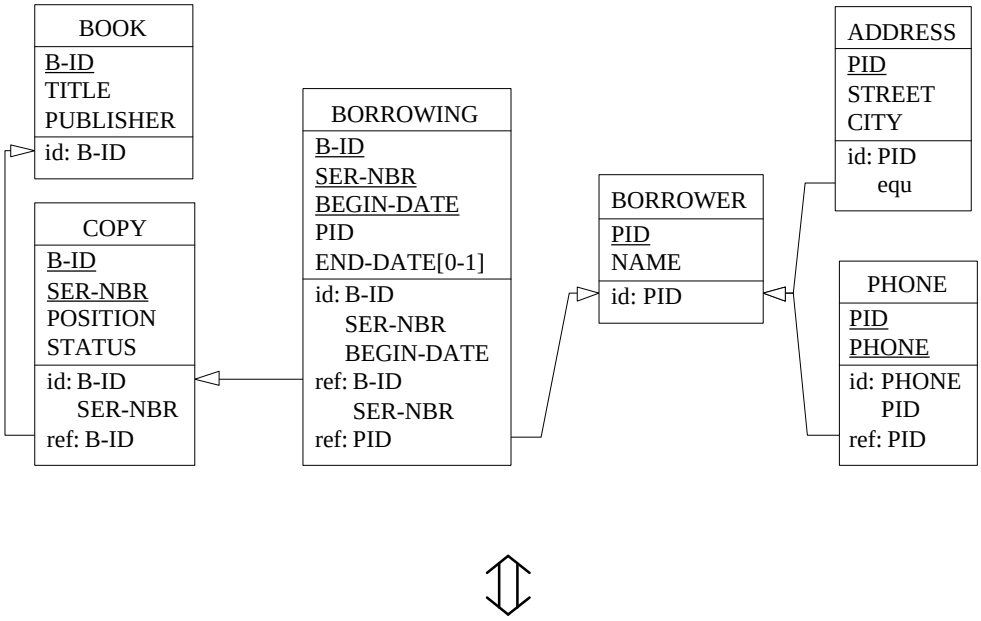
*3. Normalization*

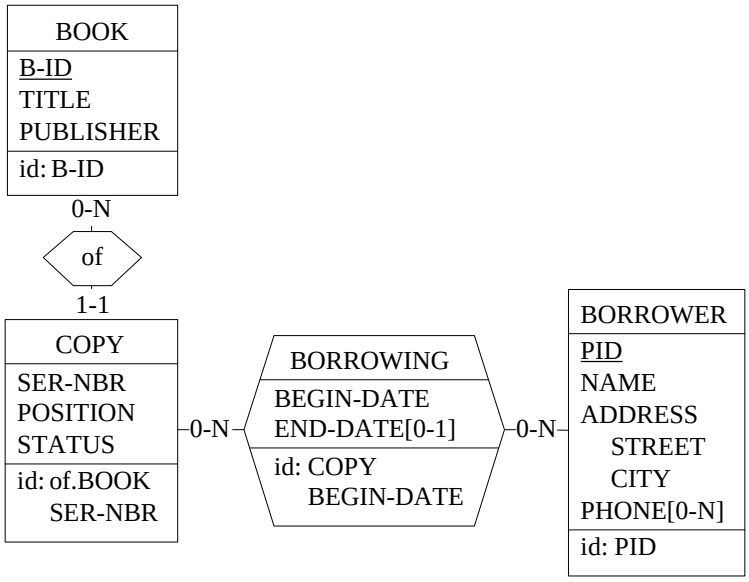
Transform Relationship Entity types into Rel-types

Transform Attribute Entity types into attributes

Etc

## Example


BOOK
B-ID
TITLE
PUBLISHER
id: B-ID
COPY
B-ID
SER-NBR
POSITION
STATUS
id: B-ID
SER-NBR
ref: B-ID
BORROWING
B-ID
SER-NBR
BEGIN-DATE
PID
END-DATE[0-1]
id: B-ID
SER-NBR
BEGIN-DATE
ref: B-ID
SER-NBR
ref: PID
BORROWER
PID
NAME
id: PID
ADDRESS
PID
STREET
CITY
id: PID
equ
PHONE
PID
PHONE
id: PHONE
PID
ref: PID


BOOK
B-ID
TITLE
PUBLISHER
id: B-ID
0-N
of
1-1
COPY
SER-NBR
POSITION
STATUS
id: of.BOOK
SER-NBR
0-N
BORROWING
BEGIN-DATE
END-DATE[0-1]
id: COPY
BEGIN-DATE
0-N
BORROWER
PID
NAME
ADDRESS
STREET
CITY
PHONE[0-N]
id: PID

**Script history of the example**

*Untranslation*

of ← FK-to-RT(COPY,{B-ID},BOOK)

cb ← FK-to-RT(BORROWING,{B-ID,SER-NBR},COPY)

bb ← FK-to-RT(BORROWING,{PID,},BORROWER)

ab ← FK-to-RT(ADDRESS,{PID,},BORROWER)

pb ← FK-to-RT(PHONE,{PID,},BORROWER)

*Normalization*

BORROWING ← ET-to-RT(BORROWING)

ADDRESS ← ET-to-Att(ADDRESS)

PHONE ← ET-to-Att(PHONE)

**Objective**

Schemas S1 and S2 are semantically equivalent if *there exists a chain of SR-transformations that transform S1 into S2 (or conversely)*;

or

Schemas S1 and S2 are semantically equivalent if *there exists, for each of them, a chain of SR-transformations that transform them into the same schema*.

N.B. Other definitions exist for this concept.

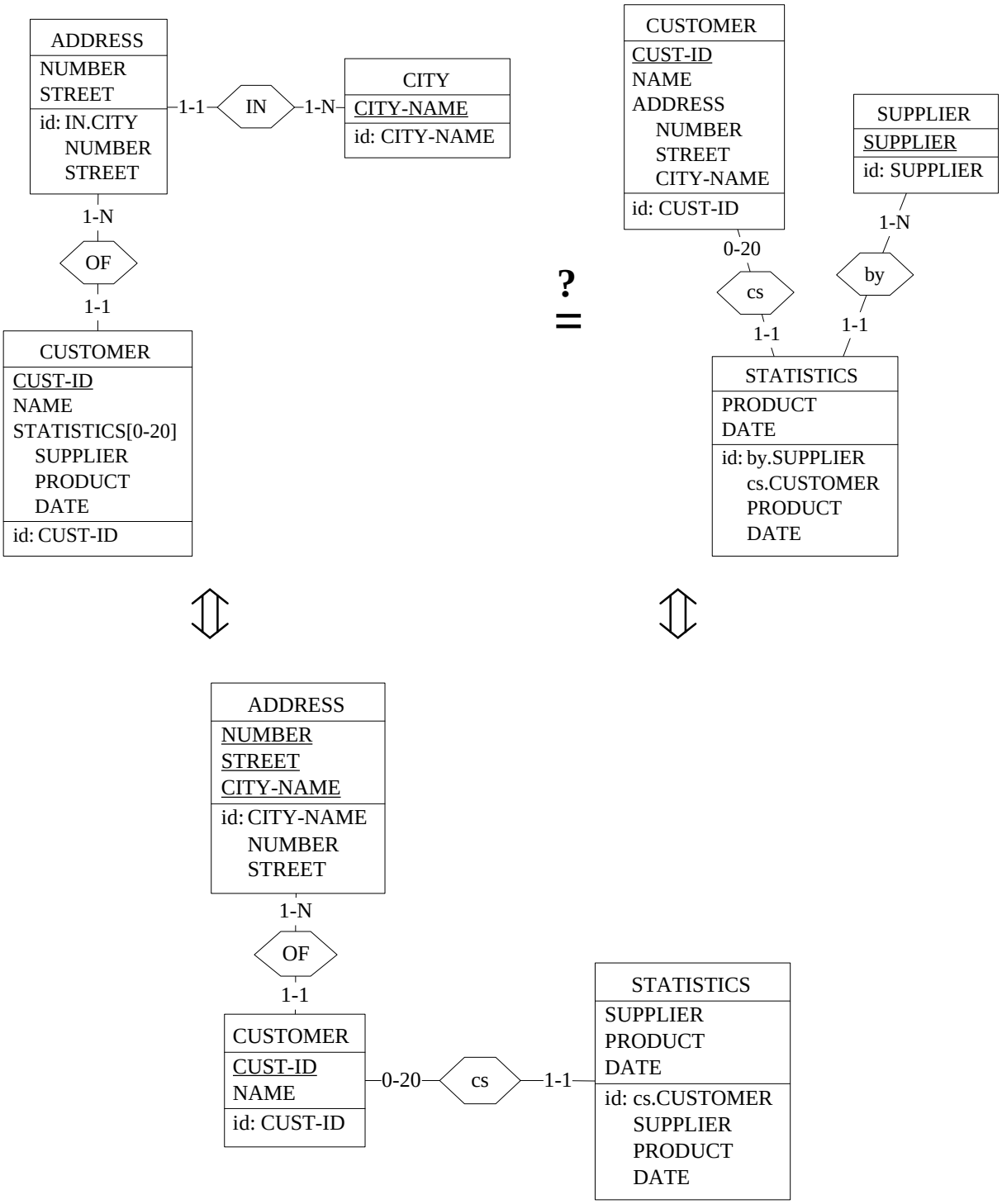
References : [LIEN,82], [JAJODIA,83], [DATRI,84], [KOBAYASHI,86], [HAINAUT,91a], [BATINI,93], [VIDAL,95]

**Script**

?

Some proposals exist for comparing relational structures, but much remains to be done for higher-order formalisms such as ER and OO models. For instance, wouldn't it be better to transform both S1 and S2 into some sort of canonical form (a primitive binary model for instance), then to compare their expressions, as suggested in the second definition ?

## Example


ADDRESS
NUMBER
STREET
id: IN.CITY
NUMBER
STREET
1-1
IN
1-N
CITY
CITY-NAME
id: CITY-NAME
1-N
OF
1-1
CUSTOMER
CUST-ID
NAME
STATISTICS[0-20]
SUPPLIER
PRODUCT
DATE
id: CUST-ID
?
=
CUSTOMER
CUST-ID
NAME
ADDRESS
NUMBER
STREET
CITY-NAME
id: CUST-ID
SUPPLIER
SUPPLIER
id: SUPPLIER
1-N
0-20
cs
by
1-1
1-1
STATISTICS
PRODUCT
DATE
id: by.SUPPLIER
cs.CUSTOMER
PRODUCT
DATE
ADDRESS
NUMBER
STREET
CITY-NAME
id: CITY-NAME
NUMBER
STREET
1-N
OF
1-1
CUSTOMER
CUST-ID
NAME
id: CUST-ID
0-20
cs
1-1
STATISTICS
SUPPLIER
PRODUCT
DATE
id: cs.CUSTOMER
SUPPLIER
PRODUCT
DATE

**Script history of the example (from left to right)**

CITY-NAME ← ET-to-Att(CITY)

(STATISTICS,cs) ← Att-to-ET/Instance(CUSTOMER,{STATISTICS})

*then*

ADDRESS ← ET-to-Att(ADDRESS)

(SUPPLIER,by) ← Att-to-ET/Value(STATISTICS,{SUPPLIER})

**Objective**

Integrating schemas, or views, consists in building a minimal schema whose semantics encompasses that of these origin schemas.

Most studies have coped with integrating conceptual views. However, a strong need also exists in reverse engineering activities. For instance, analysing COBOL programs yields one logical/physical view of the files per program. Recovering the complete description of these files requires merging these views.

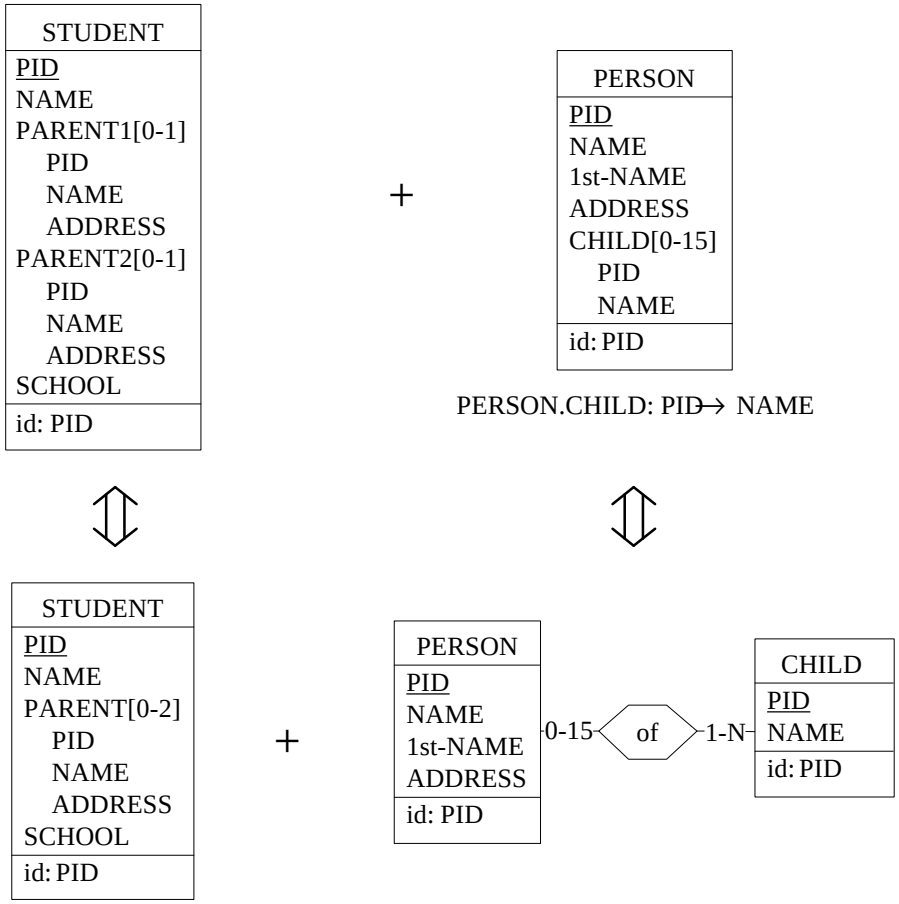
References : [BATINI,83], [NAVATHE,84], [MOTRO,87], [BATINI,92], [JORIS,92], [JOHENNESSON,93], [SPACCAPIETRA,92]

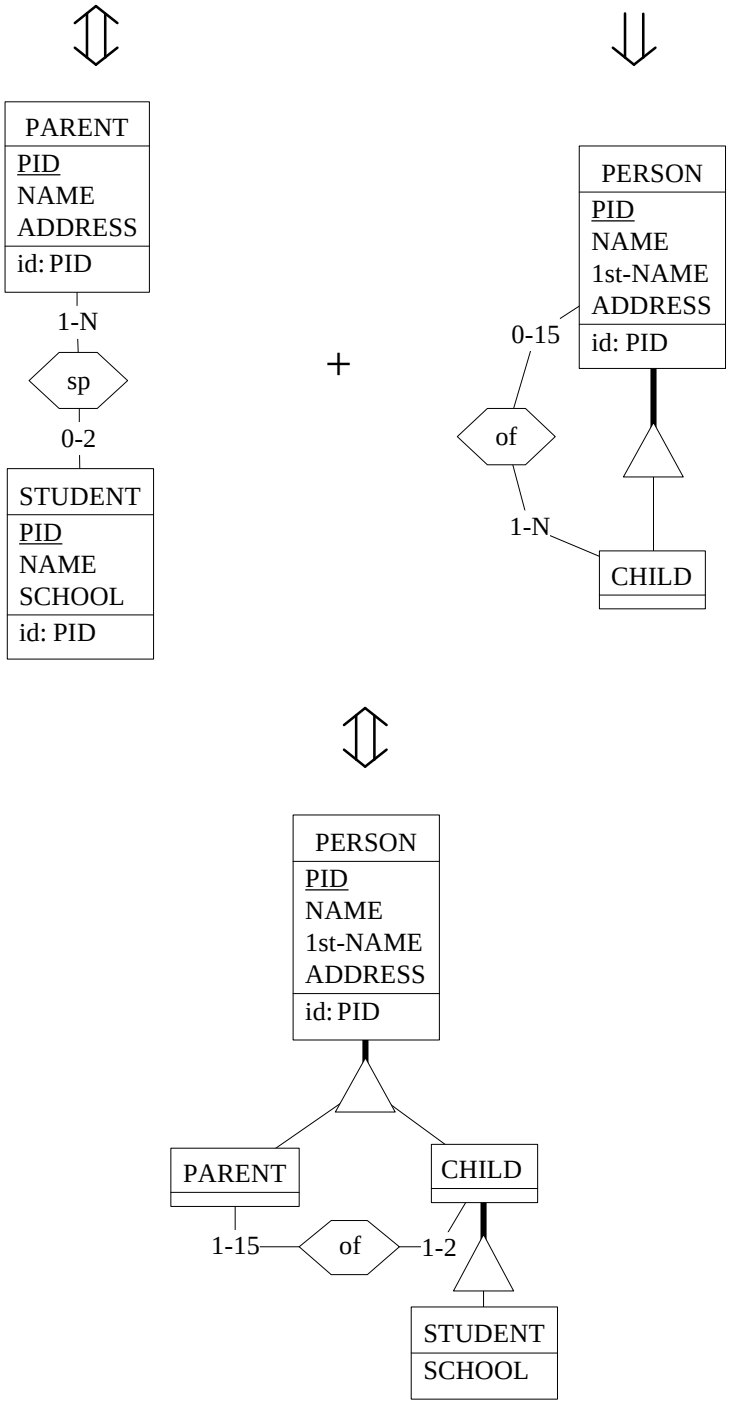
**Script**

Same uncertainty as for proving schema equivalence (7.6).

Question : does a *canonical* form exist (e.g. some binary ER model), which could make merging easier ?

**Example**


STUDENT
PID
NAME
PARENT1[0-1]
PID
NAME
ADDRESS
PARENT2[0-1]
PID
NAME
ADDRESS
SCHOOL
id: PID
+
PERSON
PID
NAME
1st-NAME
ADDRESS
CHILD[0-15]
PID
NAME
id: PID
PERSON.CHILD: PID→ NAME
STUDENT
PID
NAME
PARENT[0-2]
PID
NAME
ADDRESS
SCHOOL
id: PID
+
PERSON
PID
NAME
1st-NAME
ADDRESS
id: PID
0-15
of
1-N
CHILD
PID
NAME
id: PID


PARENT
PID
NAME
ADDRESS
id: PID
1-N
sp
0-2
STUDENT
PID
NAME
SCHOOL
id: PID
+
PERSON
PID
NAME
1st-NAME
ADDRESS
id: PID
0-15
of
1-N
CHILD
PERSON
PID
NAME
1st-NAME
ADDRESS
id: PID
PARENT
CHILD
1-15
of
1-2
STUDENT
SCHOOL

**Script history of the example (Left)**

PARENT ← SerialAtt-to-MultiAtt(STUDENT,{PARENT1,PARENT2})
(PARENT,sp) ← Att-to-ET/Value(STUDENT,{PARENT})

**Script history of the example (Right)**

(CHILD,of) ← Att-to-ET/Value(PERSON,{CHILD})
(PERSON,{}) ← Make-Subtype((PARENT,{}),(CHILD,{}))

then **merge** (*which is another story*)

### Objective

A view is a virtual data structure whose instances are derived from the database instances through a (more or less complex) query.

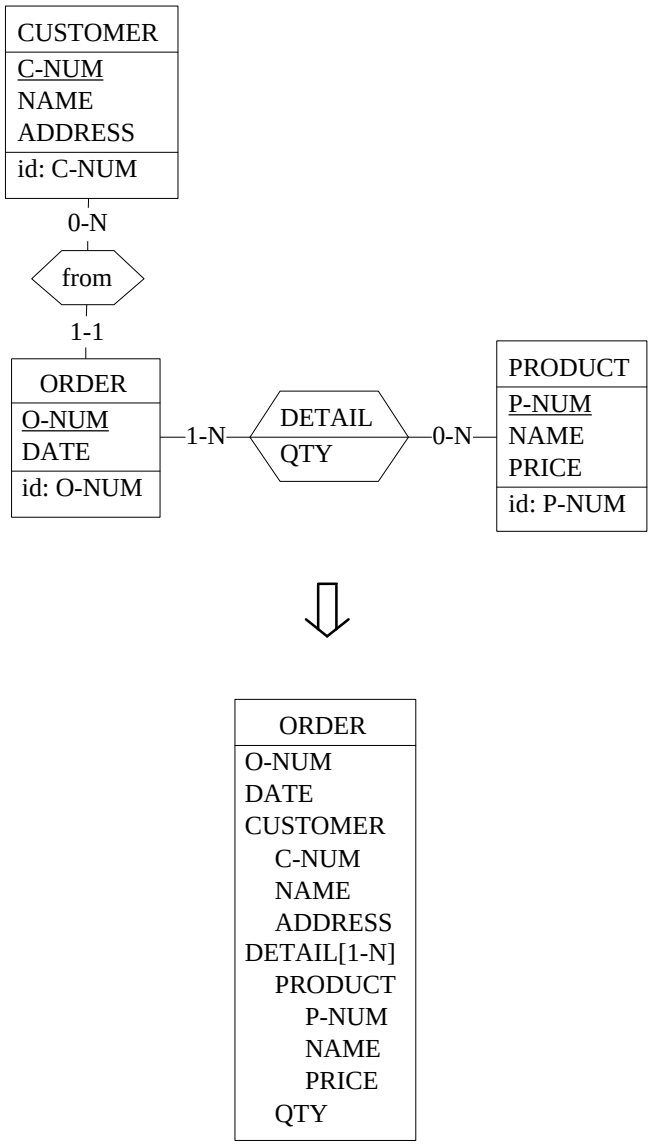
References : [MOTRO,87], [BATINI,93], [LING,89]

### Script

Since each view is defined by a query, it can also be defined by a chain of instance transformations, and therefore, more generally, by a chain of transformations.

Note that in most case, the global transformation is not semantics-preserving. Indeed, some origin constructs are discarded, and some transformations are used in an incomplete way (e.g. some IC, such as identifiers or FD, are dropped).

**Example**


CUSTOMER
C-NUM
NAME
ADDRESS
id: C-NUM
0-N
from
1-1
ORDER
O-NUM
DATE
id: O-NUM
1-N
DETAIL
QTY
0-N
PRODUCT
P-NUM
NAME
PRICE
id: P-NUM
ORDER
O-NUM
DATE
CUSTOMER
C-NUM
NAME
ADDRESS
DETAIL[1-N]
PRODUCT
P-NUM
NAME
PRICE
QTY

**Script history of the example**

Σ1: CUSTOMER ← ET-to-Att(CUSTOMER)

Σ2: (DETAIL,{(ORDER,r1),(PRODUCT,r2)}) ← RT-to-ET(DETAIL)

Σ3: PRODUCT ← ET-to-Att(PRODUCT) *(lossly, due to cardinality 0-N)*

Σ4: DETAIL ← ET-to-Att(DETAIL)

**Instance derivation rule**

$I_V = t_{\Sigma 4}(t_{\Sigma 3}(t_{\Sigma 2}(t_{\Sigma 1}(I_d))))$

$I_V$ : instance of the view

$I_d$ : database instance

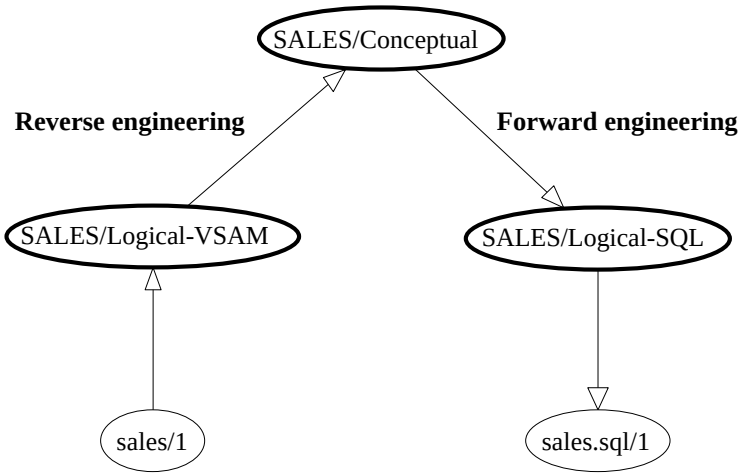
## Objective

This is a traditional process which can be supported by a transformational approach. Starting from database D1, implemented with technology T1, it consists in building a new database D2, equivalent to D1, but implemented in technology T2.

The problem is three-fold :

- converting the database schema (tractable),
- converting the data (fairly complex),
- converting the programs (very complex).

The cleanest, and most general, way to convert the schema consists in reverse engineering D1, then in implementing this schema in technology T2 :



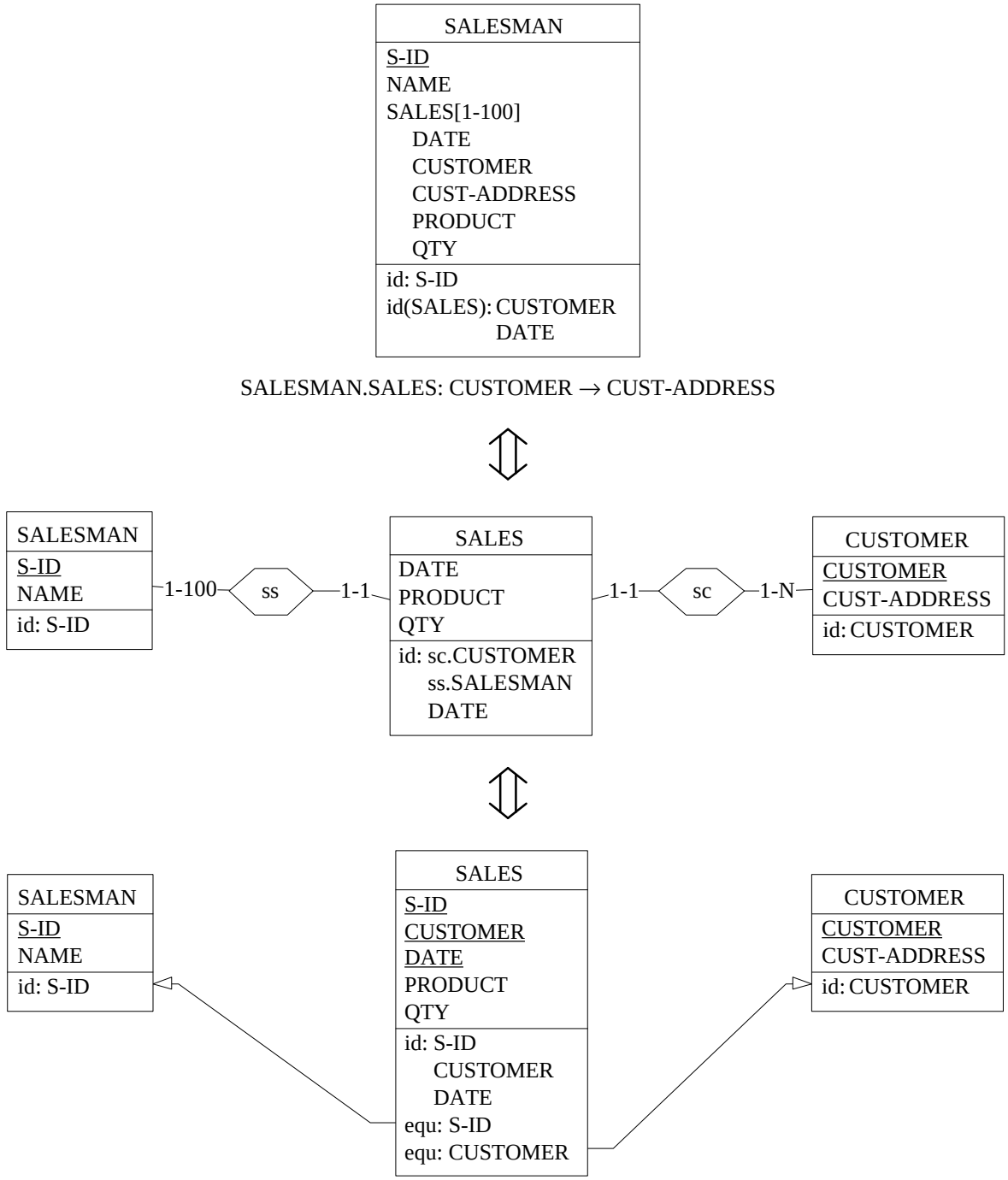

References : [NAVATHE,80], [HAINAUT,94b], [HAINAUT,95a]

## Script

= Script for Reverse Engineering

+ Script for Forward Engineering (Translation [7.3] + Optimization [7.4])

## Example (COBOL to SQL conversion)



**Script history of the example**

*COBOL file Reverse engineering*

Σ1: (SALES,ss) ← Att-to-ET/Instance(SALESMAN,{SALES})

Σ2: (CUSTOMER,sc) ← Att-to-ET/Value(SALES,{CUSTOMER,CUST-ADDRESS})

*SQL Database Forward engineering*

Σ3: {S-ID} ← RT-to-FK(ss,SALES)

Σ4: {CUSTOMER} ← RT-to-FK(sc,CUSTOMER)

**Instance derivation rule**

$I_{sql} = t_{\Sigma 4}(t_{\Sigma 3}(t_{\Sigma 2}(t_{\Sigma 1}(I_{cob}))))$

$I_{cob}$ : instance of the COBOL file

$I_{sql}$ : instance of the SQL database

**Program derivation rule ?**

Excellent question ! No clear answer so far.

Most probably, the instance mappings should play a central role. However, since the procedural code must be reverse engineered first, and since this problem still is unsolved, automatically converting any program by modifying the source code should be considered intractable at the present time.

### Objective

A collection of existing databases, developed independently, are to be considered from now on as a single, homogeneous and consistent database against which new applications can be developed.

Unsurprisingly, this problem is quite close to Database conversion, Schema integration and View derivation.

References : [BATINI,92], [SHETH,90], [RUSINKEIWICS,92]; see also [CoopIS,93], [CoopIS,94], [CoopIS,95], [DS-5,92], [RIDE-IMS,91], [RIDE-IMS,93], [FGCS,94], [WHDS,89] and [WIDS,93] (as a starter!).

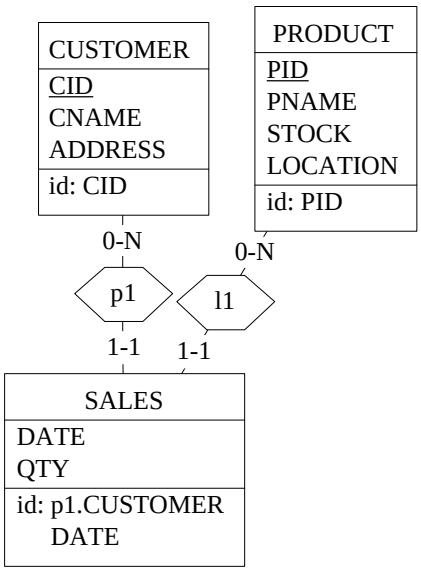
### Script

?

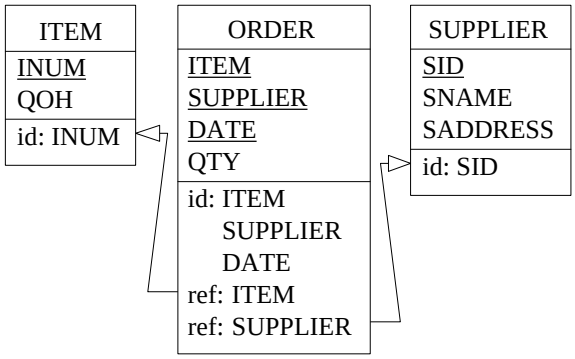
Depends on the chosen *canonical model* (the common model in which the schemas of the participating databases are expressed).

## Example

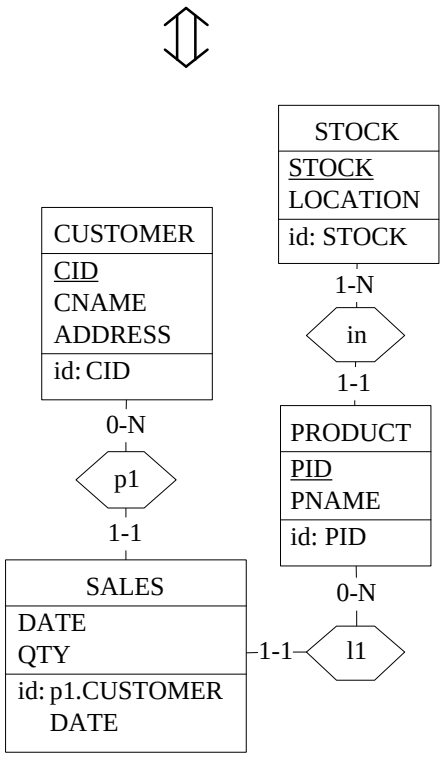
### IMS database

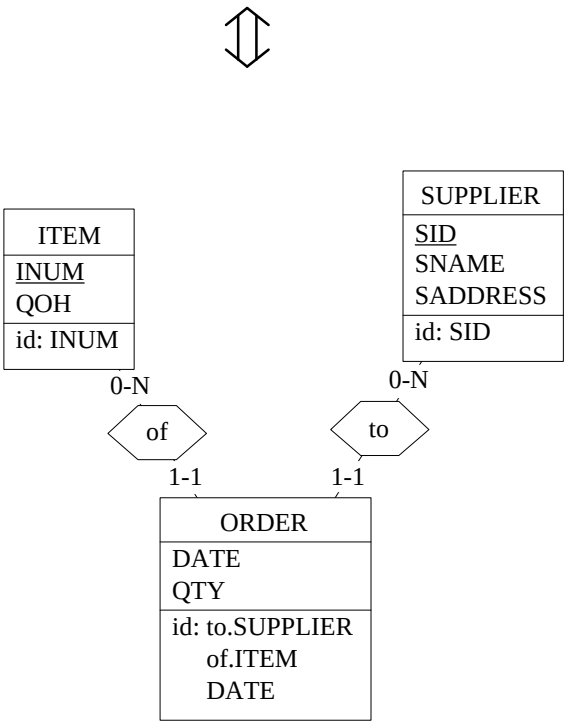


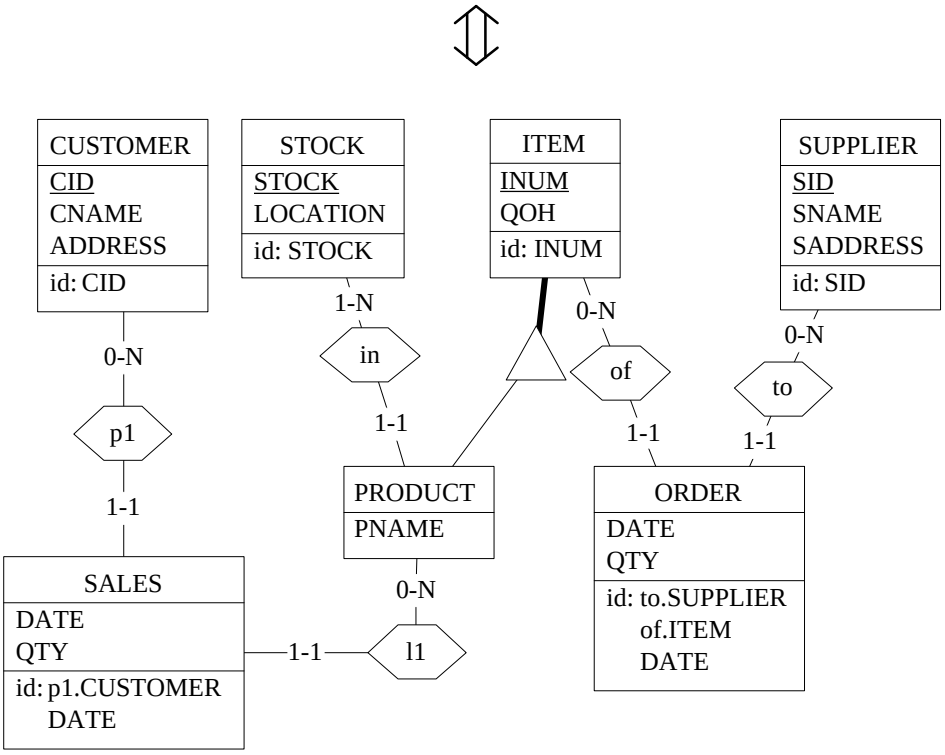

### SQL database




PRODUCT: STOCK → LOCATION






**Script history of the example**

*IMS to canonical schema mapping*

(STOCK,in) ← Att-to-ET/Value(PRODUCT,{STOCK,LOCATION})

*SQL to canonical schema mapping*

of ← FK-to-RT(ORDER,{ITEM},ITEM)

to ← FK-to-RT(ORDER,{SUPPLIER},SUPPLIER)

*then* ***merge***

**Objective**

*Reverse engineering* database D1 produces the conceptual schema C1. In addition, it can provide the analyst with an invaluable by-product : a possible design of D1, i.e. a chain of operations which could have been carried out when developing D1. Carrying out this design on C1 yields the schema of D1.

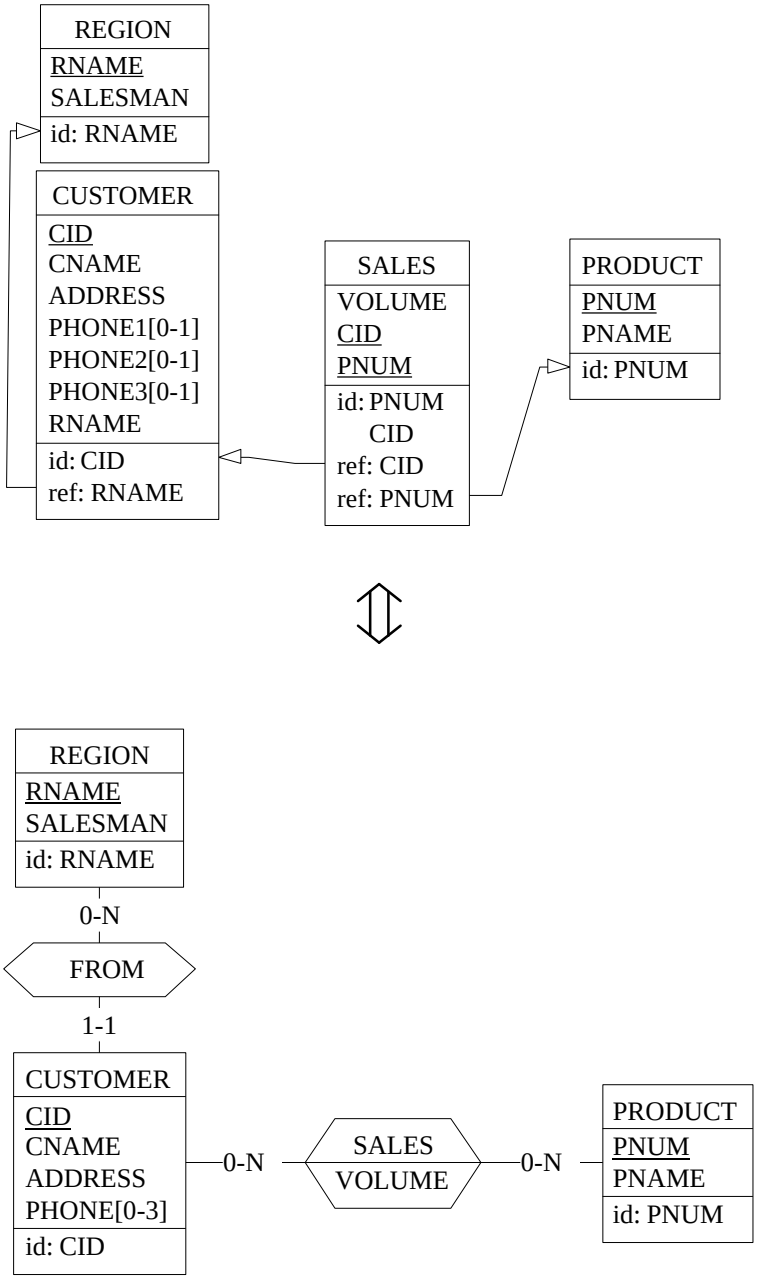
This is a nice application of the traceability property induced by the transformational approach.

References : [HAINAUT,94b]

**Procedure (simplified)**

1. Record the history **Hr** of reverse engineering
2. Replace each action of **Hr** by its inverse
3. Reverse the order of the actions of **Hr**
4. Group these actions into higher-level processes (Translation, Optimization, Coding, etc)

**Example**


REGION
RNAME
SALESMAN
id: RNAME
CUSTOMER
CID
CNAME
ADDRESS
PHONE1[0-1]
PHONE2[0-1]
PHONE3[0-1]
RNAME
id: CID
ref: RNAME
SALES
VOLUME
CID
PNUM
id: PNUM
CID
ref: CID
ref: PNUM
PRODUCT
PNUM
PNAME
id: PNUM
REGION
RNAME
SALESMAN
id: RNAME
0-N
FROM
1-1
CUSTOMER
CID
CNAME
ADDRESS
PHONE[0-3]
id: CID
0-N
SALES
VOLUME
0-N
PRODUCT
PNUM
PNAME
id: PNUM

**Script history of reverse engineering**

from ← FK-to-RT(CUSTOMER,{RNAME},REGION)

r1 ← FK-to-RT(SALES,{CID},CUSTOMER)

r2 ← FK-to-RT(SALES,{PNUM},PRODUCT)

PHONE
← SerialAtt-to-MultiAtt(CUSTOMER,{PHONE1,PHONE2,PHONE3})

SALES ← ET-to-RT(SALES)

**Building the design history**

*Reversing the transformations*

{RNAME} ← RT-to-FK(from,CUSTOMER)

{CID} ← RT-to-FK(r1,SALES)

{PNUM} ← RT-to-FK(r2,SALES)

{PHONE1,PHONE2,PHONE3}
← MultiAtt-to-SerialAtt(CUSTOMER,PHONE)

(SALES,{(CUSTOMER,r1),(PRODUCT,r2)}) ← RT-to-ET(SALES)

*Reversing the order of the transformations*

(SALES,{(CUSTOMER,r1),(PRODUCT,r2)}) ← RT-to-ET(SALES)

{PHONE1,PHONE2,PHONE3}
← MultiAtt-to-SerialAtt(CUSTOMER,PHONE)

{PNUM} ← RT-to-FK(r2,SALES)

{CID} ← RT-to-FK(r1,SALES)

{RNAME} ← RT-to-FK(from,CUSTOMER)

*Grouping the actions : the script of the final design*

| **LOGICAL DATABASE DESIGN** |
|---|
| ***Schema simplification*** |
| (SALES,{(CUSTOMER,r1),(PRODUCT,r2)}) ← RT-to-ET(SALES) |
| ***Schema optimization*** |
| {PHONE1,PHONE2,PHONE3}<br>← MultiAtt-to-SerialAtt(CUSTOMER,PHONE) |
| ***Schema translation*** |
| {PNUM} ← RT-to-FK(r2,SALES)<br>{CID} ← RT-to-FK(r1,SALES)<br>{RNAME} ← RT-to-FK(from,CUSTOMER) |

## Database Evolution

How to propagate a conceptual change down to the physical schema, the data and the application programs ? How to propagate a physical change (e.g. adding a column to a table) up to the conceptual schema ?

Reference : [RODICK,92], [RODICK,93], [HAINAUT,94b]

## Model equivalence

Is NIAM as powerful as ER ? Can OO models express more semantics than ER models ?

## Model translation

How to express an ER schema in NIAM ? How to translate an OO schema into an ER schema ?

## Engineering traceability

A script history is a completely formalized trace of transformational activities which can be processed both way. In particular, this trace allows

- replaying design processes
- undoing former processes
- analyzing analysts behaviour

Reference : [HAINAUT,94b]

# Part 8

## TRANSFORMATIONS and CASE tools

## State of the art

Potentially, many DB design CASE tools support schema transformations, but most often implicitly, and without user control.

Some exceptions : **DDEW** [ROSENTHAL,94], **TRAMIS** [HAINAUT,92a] and **DB-MAIN** [HAINAUT,95b].

## The DB-MAIN CASE tool

DB-MAIN is a graphical, transformation-based, programmable, CASE tool dedicated to Database Applications Engineering.

## Objectives

To support the major Database Engineering activities
- database design
- database reverse engineering
- database re-engineering
- database maintenance and evolution

To support flexible and non-standard strategies

To allow functional extensibility

To allow methodological customization

Low cost and performance

## Origin

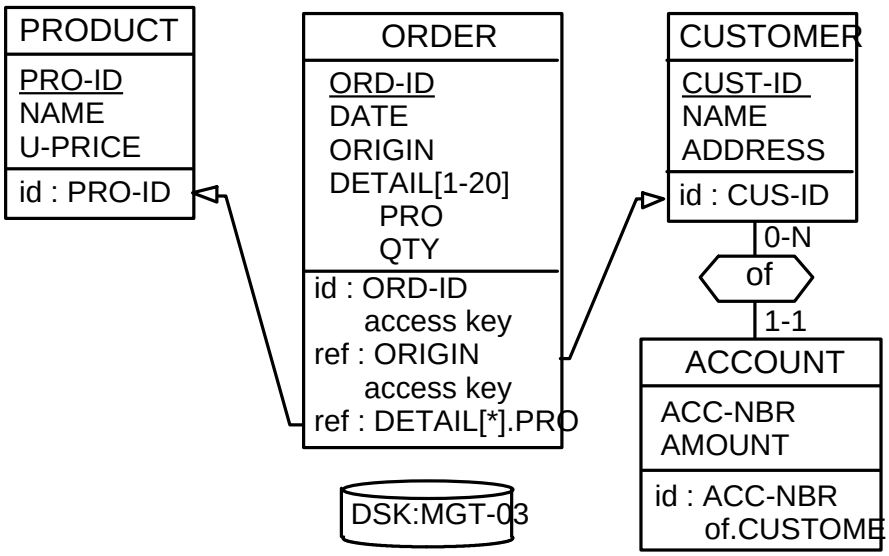
- One of the main results of the DB-MAIN R&D programme dedicated to *Database Applications Maintenance et Evolution*
- 1993-2001+
- Version 4.0 : ± 40 man/year,
  released in October 1995, 1996, 1997, 1998
  addresses database engineering processes

## Implementation

- PC/Windows workstation
- developed in C++
- **an Education/Demo version is available**

# The DB-MAIN specification model

DB-MAIN includes a wide-spectrum specification model that supports the representation of schemas at different abstraction levels and according to various ER/OR paradigms.




***Conceptual objects***

**entity types** PRODUCT**,** CUSTOMER**,** ACCOUNT**;**
**rel-type** of

***Logical objects***

**record type** ORDER**, with single-valued and multivalued foreign keys**

***Physical objects***

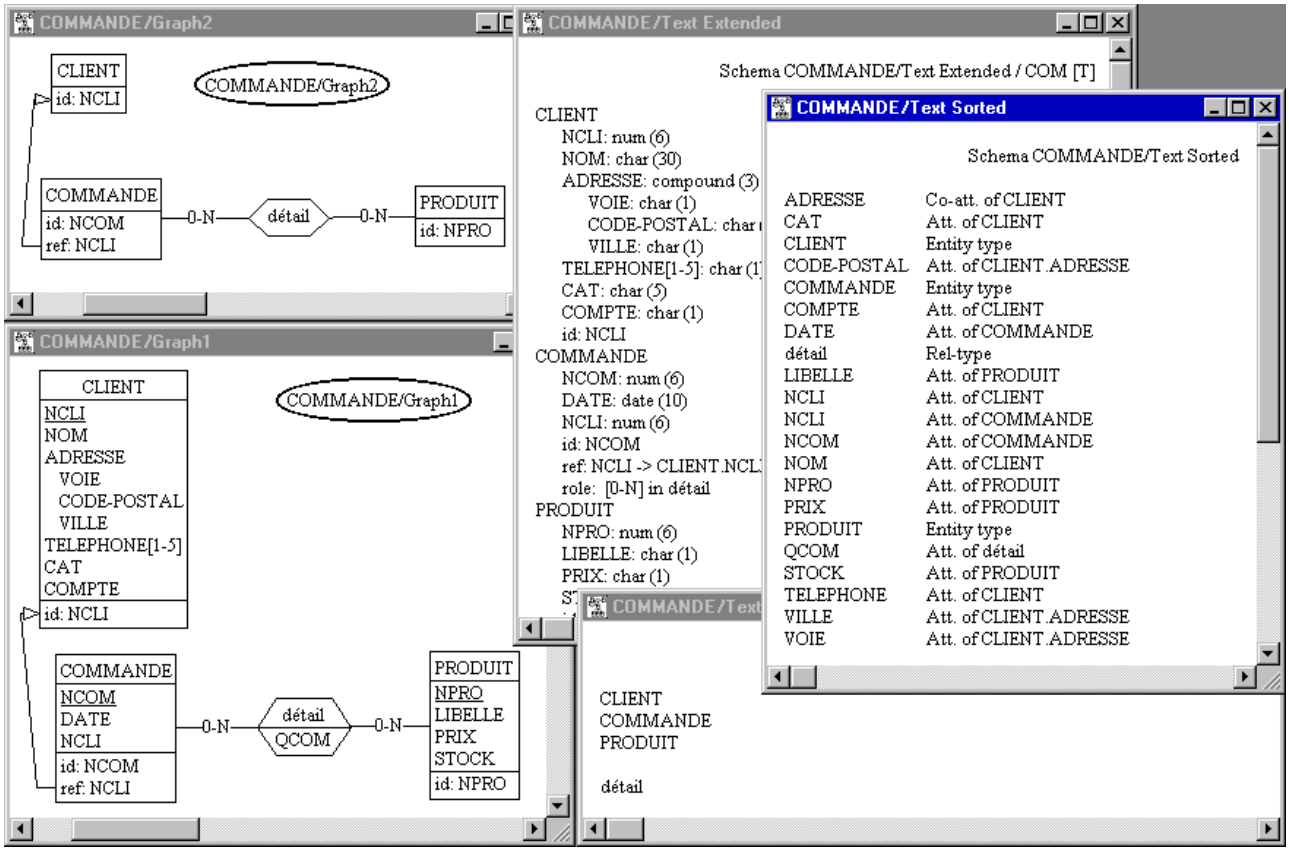
**access keys** ORDER.ORD-ID **and** ORDER.ORIGIN**;**
**file** DSK:MGT-03

# The DB-MAIN multiple view interface

Due to the large functional scope of the tool, several views of the specifications are provided : 4 hypertext views and two graphical views.

All the operators, including the transformations, can be activated whatever the view in which the source object appears.

The result of a transformation is immediately propagated to the views.



# The DB-MAIN transformation toolkits

The tool offers three levels of transformation operators :

• elementary transformations (T)
*apply transformation T to current object O*
(see section 8.4)

• global transformations (P,T) - through a specific ASSISTANT
*apply transformation T to all the objects that satisfy condition P*
(see section 8.5)

• model-driven transformations (M)
*apply the transformations needed to make the current schema satisfy model M*
(see section 8.6)

# The DB-MAIN text analysers (1)

## Motivation

One of the objectives of DB-MAIN is to support Reverse Engineering activities. It therefore includes sophisticated text analysis processors. These tools can be used in other design processes as well, such as in conceptual analysis, where there is a need to search large documents for specific information.

## Parsers

analyse data structures declarations, and store their abstract representation in the repository of the tool, as a first cut-logical schema;

- **COBOL**
- **SQL**
- **CODASYL**
- **IMS**

## Pattern-matching engine

searches external texts, such as source programs, or the repository content, for instances of specific patterns;

- **interactive or procedurally controlled;**
- **generic + specific user's pattern libraries**
- **BNF/grep flavour + pattern variables (@)**
- **coupled with *Voyager-2*.**

# The DB-MAIN text analysers (2)

## Exemple

```
select ...
from  T1,T2            ⇒   C2 may be a foreign key to T1
where T1.C1 = T2.C2
```

***The SQL generic patterns***

```
T1 ::= table-name
T2 ::= table-name
C1 ::= column-name
C2 ::= column-name
join-qualif ::= begin-SQL
                  select select-list
                  from  ! {@T1 ! @T2 | @T2 ! @T1}
                  where ! @T1"."@C1 _ "=" _ @T2"."@C2 !
                end-SQL
```

***The COBOL/DB2 specific patterns***

```
_ ::= ({"/n"|"/t"|" "})+
- ::= ({"/n"|"/t"|" "})*
begin-SQL ::= {"exec"|"EXEC"}
_{"sql"|"SQL"}_
end-SQL ::= _{"end"|"END"}
{"-exec"|"-EXEC"}-"."
select ::= {"select"|"SELECT"}
from ::= {"from"|"FROM"}
where ::= {"where"|"WHERE"}
select-list ::= any-but(from)
! ::= any-but({where|end-SQL}) {","|"/n"|"/t"|" "}
AN-name ::= [a-zA-Z][-a-zA-Z0-9]
table-name ::= AN-name
column-name ::= AN-name
```

# The DB-MAIN functional extensibility

## Motivation

Any CASE tool should allow adding specific, user-defined functions. The *Voyager-2* language (V2) allows the development of generators, extractors and loaders, evaluators, **complex transformations**, etc.

For example, complex transformation plans can be expressed into Voyager-2 procedures.

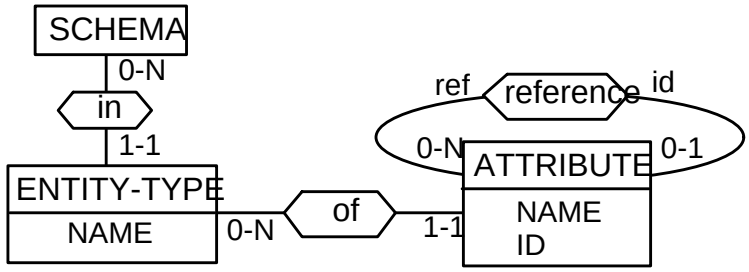
These functions enrich the basic toolset.

## Main features of *Voyager-2*

- communicating with the repository through either predicative or navigational queries;
- functions and procedures can be recursive;
- generic, shared, list structures are provided, with powerful list operators; in particular, lists of repository objects can be built and processed; automatic garbage collection is provided;
- powerful input/output text functions allow easy development of parsing and generating functions;
- all the DB-MAIN basic tools are available from V2;
- a V2 procedure can be attached to DB-MAIN objects (dialog boxes, patterns, buttons, etc)
- a V2 procedure can appear in a DB-MAIN menu in the same way as basic tools do (seamless functional extension);
- a V2 procedure is precompiled into an internal binary code. This code is interpreted by a virtual V2 machine.

## *Voyager-2* : an example

This procedure is attached to the pattern `join-qualif`, and is executed for each of its instances. It checks the conditions for C2 being a foreign key of T2 to T1, then it creates the foreign key in the repository (strongly simplified here).




```
function integer MakeForeignKey (string:T1,T2,C1,C2)
/* if C1 is an identifying attribute of entity type T1 and if C2 is an attribute of T2, and if C1 and C2
are compatible, then define C2 a foreign key to T1 */
   schema : S;
   entity_type : E;
   attribute : A,ID,FK;
   list : ALI,ALF;
{
   S := GetCurrentSchema();   /* S is the current schema */
/* ALI = list of the attributes (with name C1 and which are identifier) of the entity types in S with
name T1 */
   ALI := attribute[A]{of:entity_type[E]{in:[S] and E.NAME = T1}
                  and A.NAME = C1
                  and A.ID = true};
/* ALF = list of the attributes (with name C2) of the entity types in S with name T2 */
   ALF := attribute[A]{of:entity_type[E]{in:[S] and E.NAME = T2}
                  and A.NAME = C2};
/* if both list are not-empty, then
   if the attributes are compatible then define the attribute in ALF as a foreign key to the
   attribute in ALI */
   if not(empty(ALI) or empty(ALF)) then
      {ID := GetFirst(ALI); FK := GetFirst(ALF);
       if ID.TYPE = FK.TYPE and ID.LENGTH = FK.LENGTH then
           {connect(reference,ID,FK); return true;}
       else {return false;};}
   else  {return false;};
}
```

# The DB-MAIN assistants

DB-MAIN includes a series of Assistants, each dedicated to a specific class of problems or of manipulations. Two examples :

## The transformation assistant

It allows applying one or several transformations to selected objects (described in section 8.5).

## The analysis assistant

This tool is dedicated to the analysis of schemas. It provides two processing modes.

### *Validation mode*

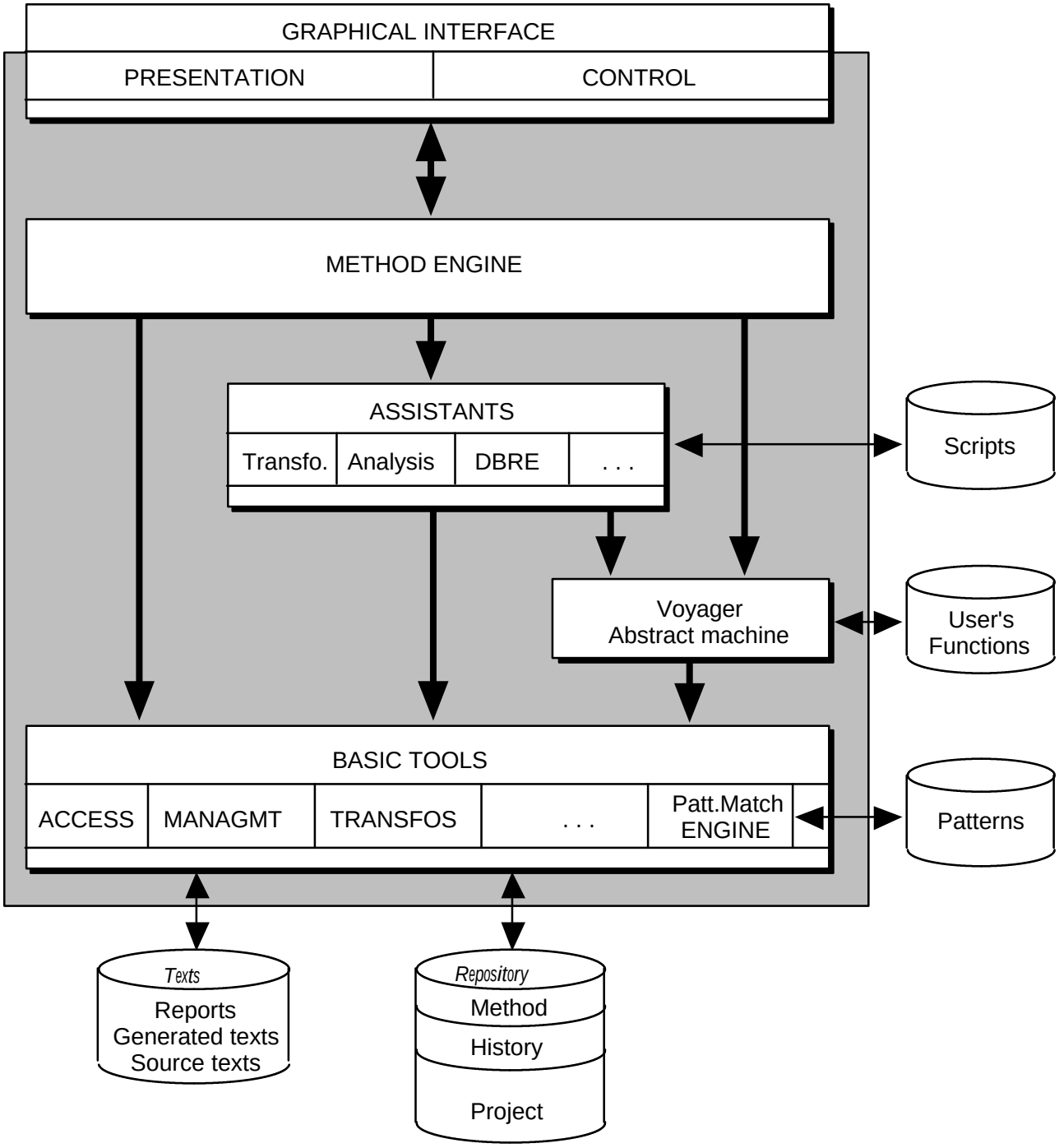
The first step consists in defining a **submodel** as a restriction of the generic specification model. This restriction appears as a boolean expression of elementary predicates stating which specification patterns are valid. Some examples : "*an entity type must have from 1 to 100 attributes*", "*a relationship type has from 2 to 2 roles*", "*the entity type names are less than 18-character long*", "*a name does not include spaces*", "*no names belong to a given list*", "*an entity type has from 0 to 1 supertype*", "*the schema is hierarchical*", "*there is no access keys*". A submodel appears as a script which can be saved and loaded. Predefined submodels are available : Normalized ER, Binary ER, NIAM, Functional ER, Bachman, Relational, CODASYL, etc. Customized predicates can be added via V2 functions.

The second step consists in evaluating the current schema against a specific submodel. This provides a list describing the violations detected.

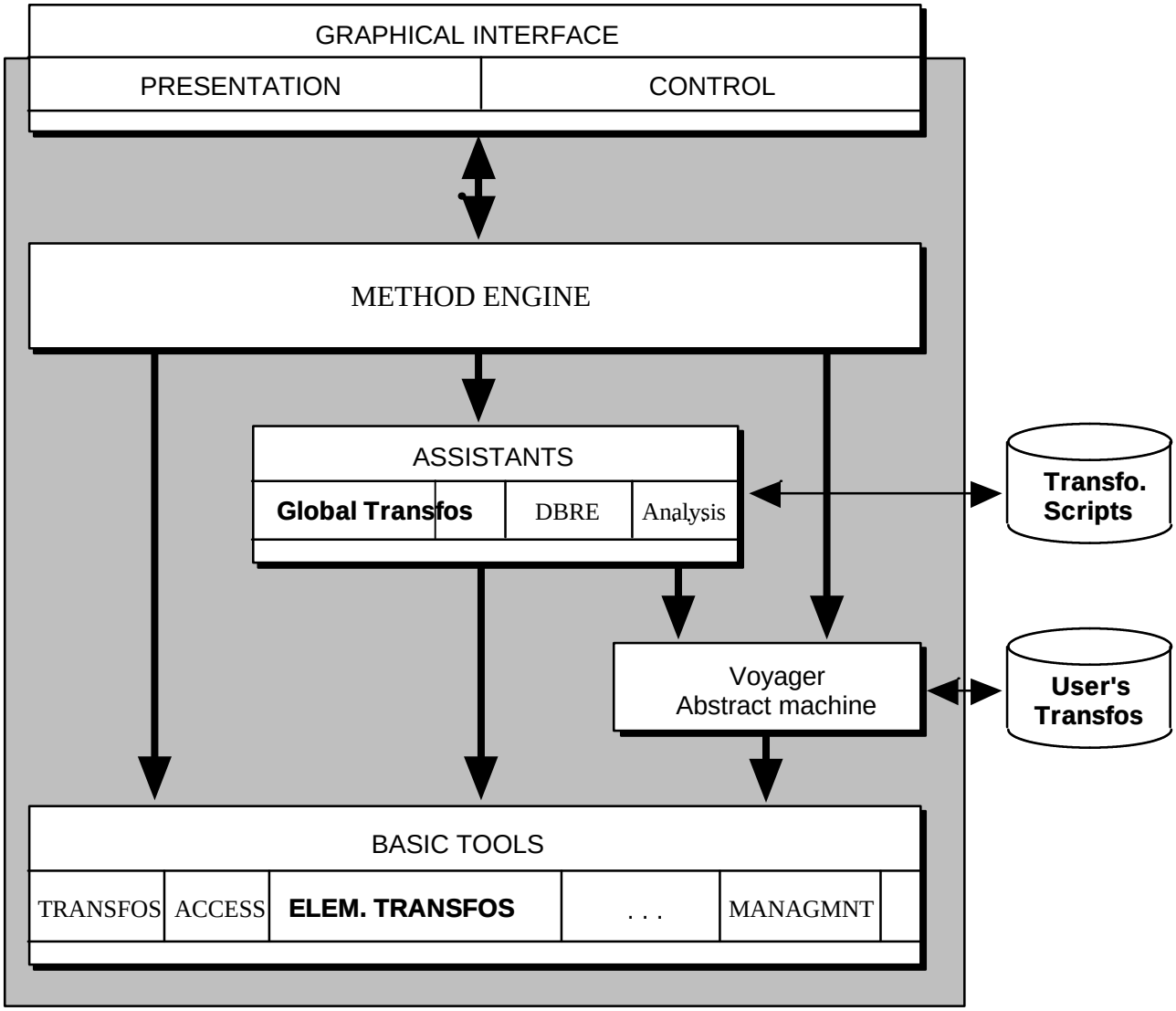
### *Search mode*

The Search mode of the Analysis assistant allows to search a schema for a complex pattern which can be described by a submodel. For instance : "*retrieve all N-ary rel-types with at least 1 attribute*", "*retrieve all the attributes the name of which begins with string 'CUST' and which do not include string 'DATE'*".

# General architecture



# Transformation-oriented architecture



# The elementary transformations (sample)

• ***Entity type***
- transform the current entity type into a rel-type
- transform the current entity type into an attribute
- transform the IS-A relations into 1-1 rel-types
- transform the 1-1 rel-types into IS-A relations
- split / merge the entity type
- add a technical primary identifier
- integrate two entity types

• ***Relationship type***
- transform the current rel-type into an entity type
- transform the current rel-type into a foreign key

• ***Attribute***
- transform the current attribute into an entity type (2 techn.)
- disaggregate the current compound attribute
- concatenate the current compound attribute
- transform the current multivalued attribute by serial attributes
- concatenate the current multivalued attribute
- make the current single-valued attribute multivalued

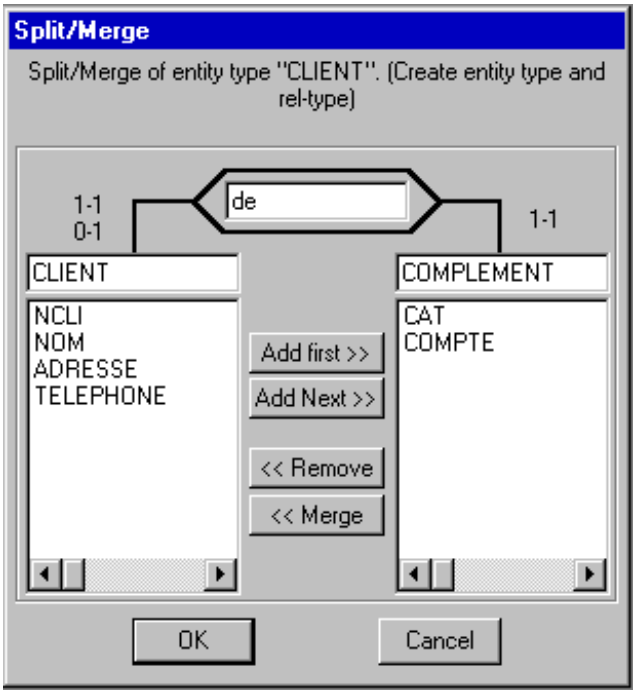
• ***Goup of attributes/roles***
- transform the current foreign key into a rel-type
- make a compound attribute from the current group

• ***Names***
- change/add/remove prefix of names
- replace the names, or parts thereof, of selected objects

# Example : the Split/Merge transformation

This practical transformation proposes three ER/OR transformations in one intuitive panel :

1. splitting an entity type into two entity types
2. merging two entity types
3. migrating attributes/roles from one entity type to another one.



## The transformation assistant

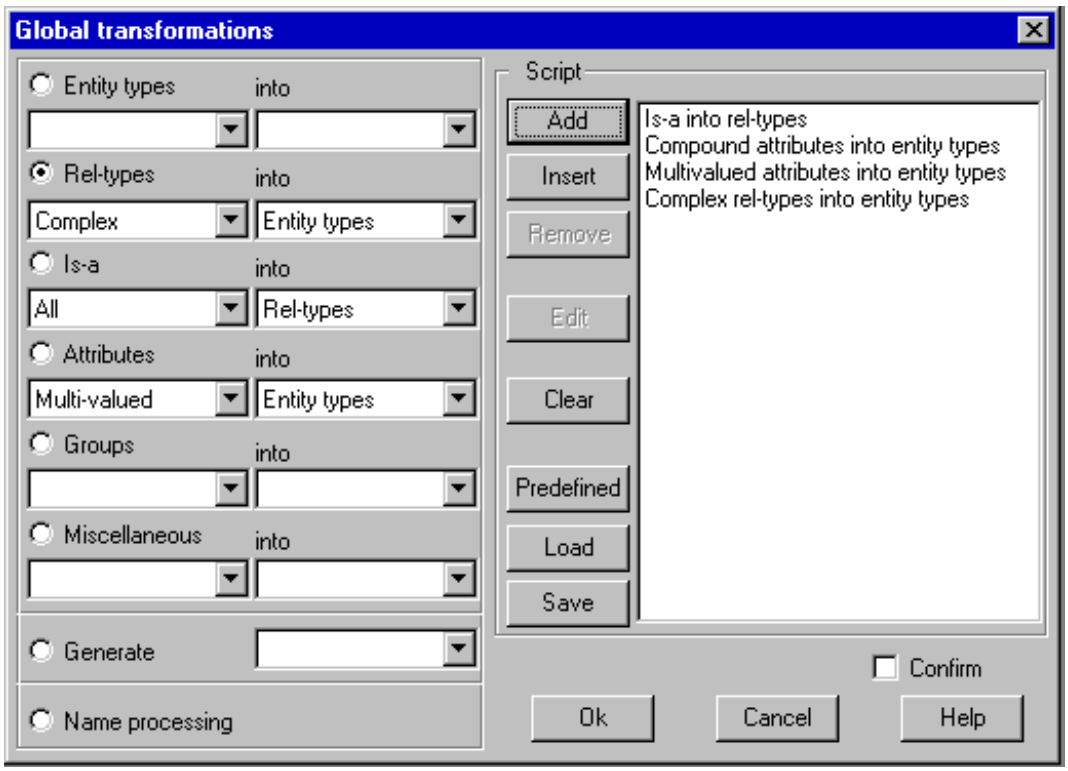
It allows applying one or several transformations to selected objects.

Each operation appears as a *problem/solution* couple, in which the problem is defined by a pre-condition (e.g. the object is a *many-to-many relationship type*), and the solution is an action resulting in eliminating the problem (e.g. *transform it into an entity type*).

Several dozens of problem/solution items are proposed. The user can select one of them, and execute it automatically or in a controlled way.

Alternatively, he can build a script comprising a list of operations, execute it, save and load it. Predefined scripts are available to transform any schema according to popular models : Bachman model, binary model, relational, CODASYL, standard files, conceptualization of relational schemas (DBRE).

Customized *problems* and *solutions* can be developed in Voyager-2, and included in the assistant



A model-based transformation is an operator which applies all the necessary transformations in such a way that the source schema is translated into an equivalent schema satisfying a specific submodel. For instance, there exists a transformation which derives a relational schema from any ER conceptual schema. A model-based transformation is defined by a transformation plan.

DB-MAIN proposes built-in, hard-coded, transformations for some popular DBMS. In addition, it offers two ways to build one's own model-based transformations :

- for simple, sequential, transformation plans, through the scripting facility of the Global Transformation Assistant; predefined scripts for some popular models are provided;
- for transformation plans of arbitrary complexity, through the development of *Voyager-2* functions.

DB-MAIN can maintain a **log** of all the design activities that have been carried out by the developer, or by the tool itself (during script or Voyager-2 function execution).

This log records the **history** of the activities.

This history is expressed into a **formal language** which is both readable, and machine-processable.

It can be examined, processed (through Voyager-2 functions) and replayed automatically, or under user control.

For further detail, see [HAINAUT,94b].

# Part 9

## CASE STUDY

The transformational paradigm will be illustrated by a more in-depth application consisting in translating a collection of COBOL files into an optimized relational database, an activity often called *Database Reengineering*, *Conversion* or *Migration* (see 7.9).

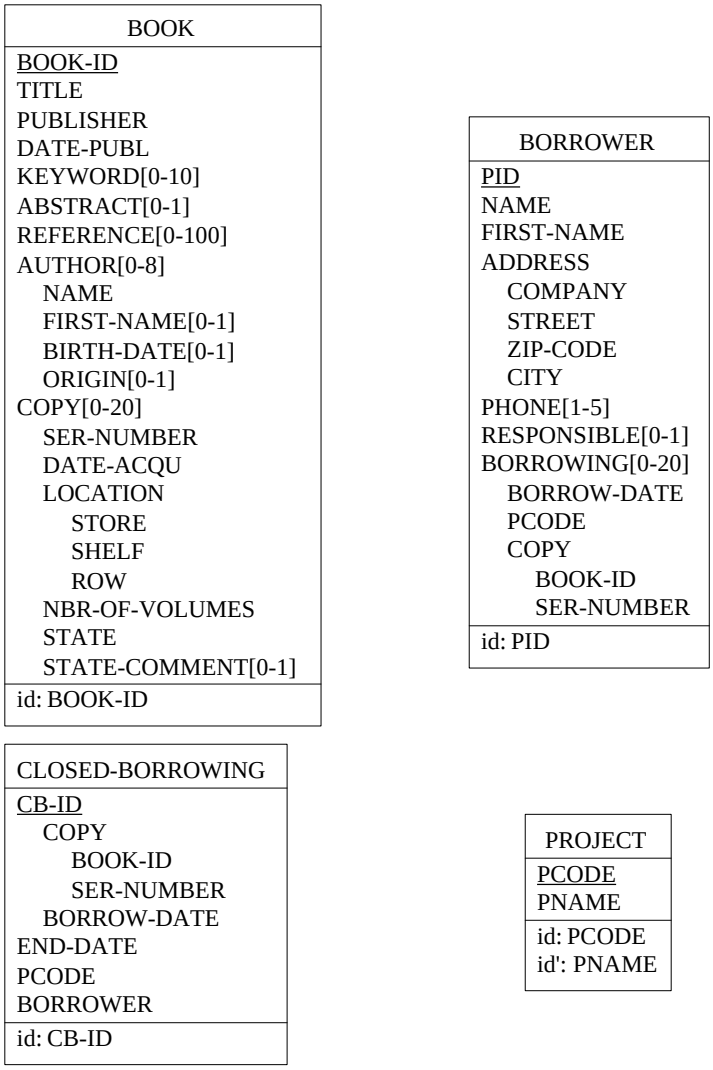
As developed in [HAINAUT,95c], this process comprises two main phases, namely Database Reverse Engineering and Database Design. Simpler procedures do exist, and are proposed, generally by service and software companies. They consist in translating each COBOL construct into a relational expression. This *physical-to-physical* translation, that bypasses the semantic interpretation, is straighforward, quick and simple, but yields, except in very particular situations, poor results in terms of performance, maintainability and documentation for example. Indeed, the result is a set of COBOL files, with all their idiosynchrasies, disguised into relational clothes.

In **Database Reverse Engineering** (see 7.5), transformations will be most useful in the Data Structure Conceptualization phase, which consists in interpreting the technical structures dependent on the COBOL model (Untranslation), and in eliminating the optimization-related constructs (De-optimization).

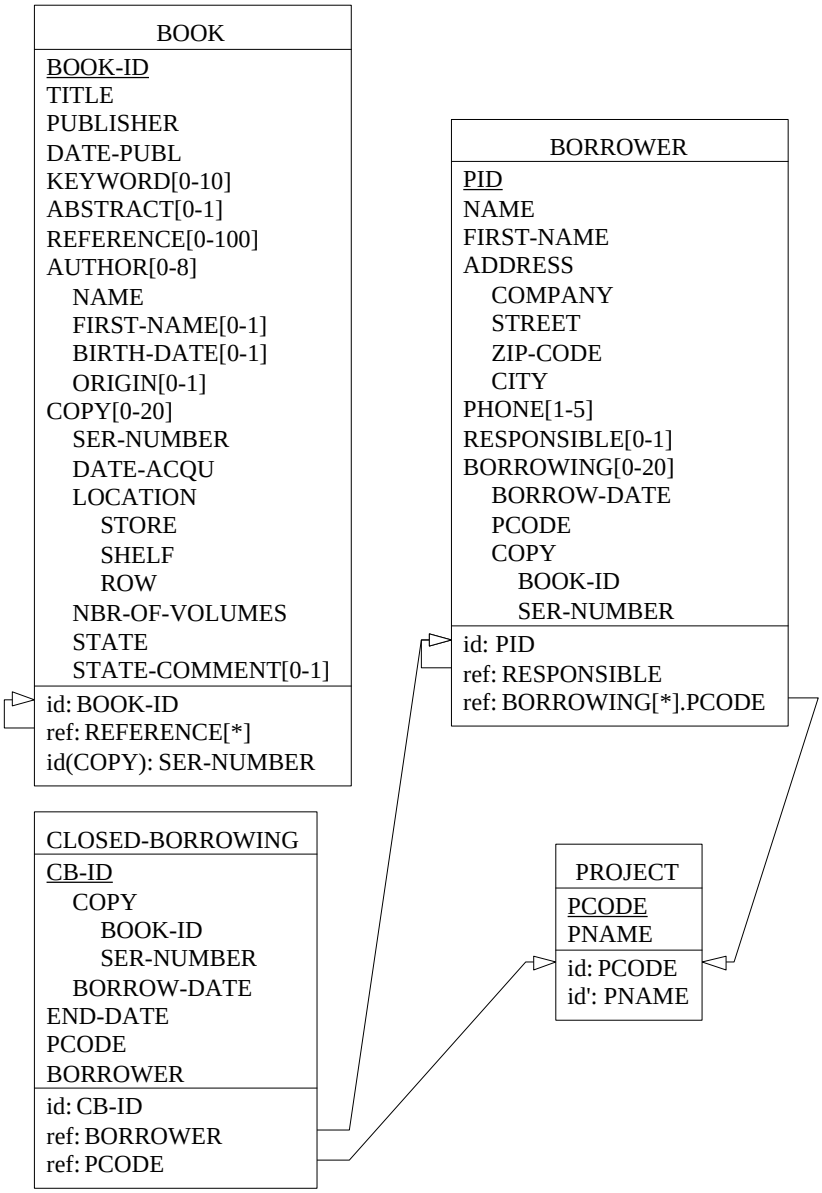
In **Database Design**, the transformations will be the basic tools in DBMS translation (see 7.3) and in Optimization (see 7.4).

Writing the history of this conversion is left as an exercise.

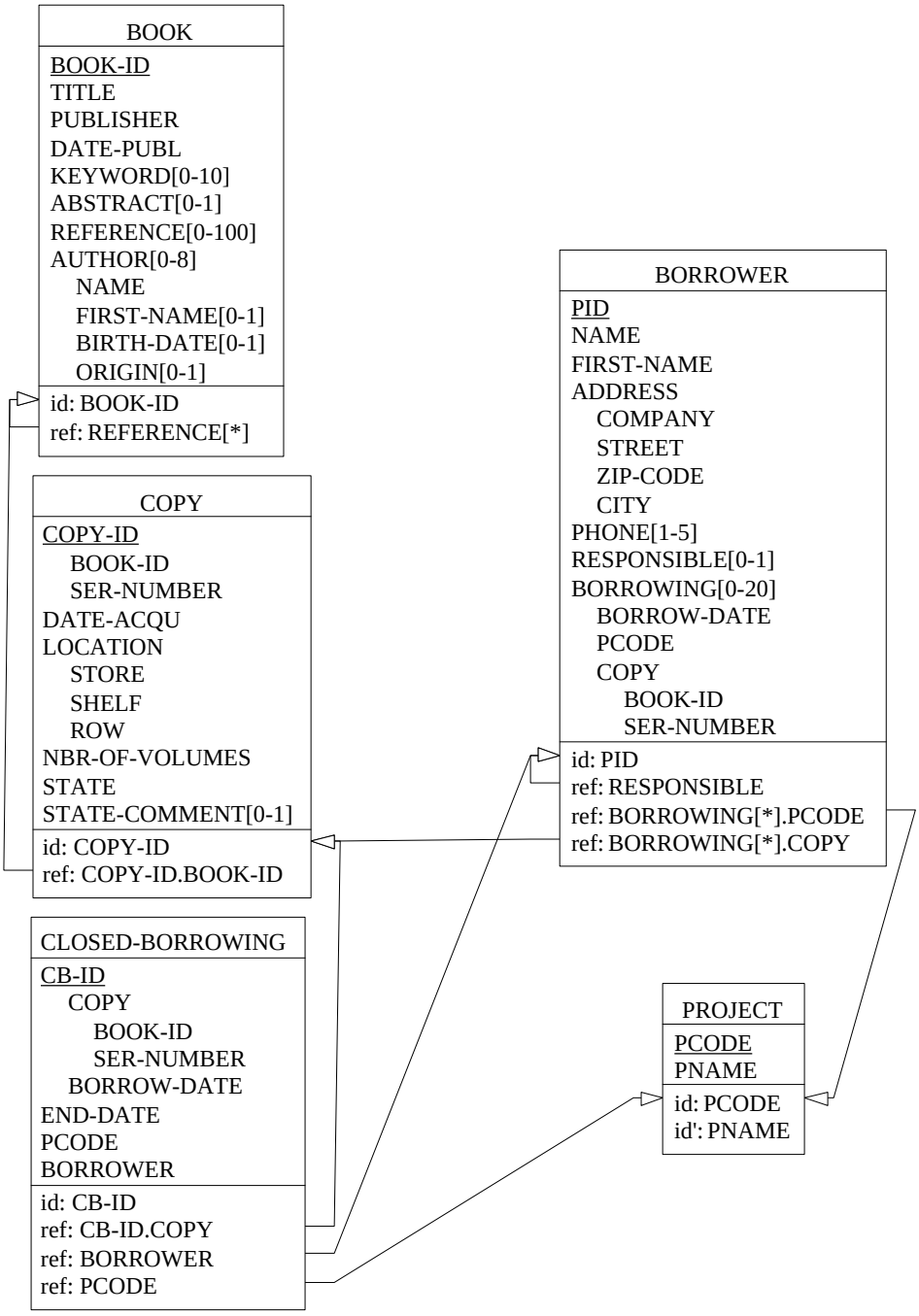
# 1.1 The abstract COBOL physical schema (first-cut)



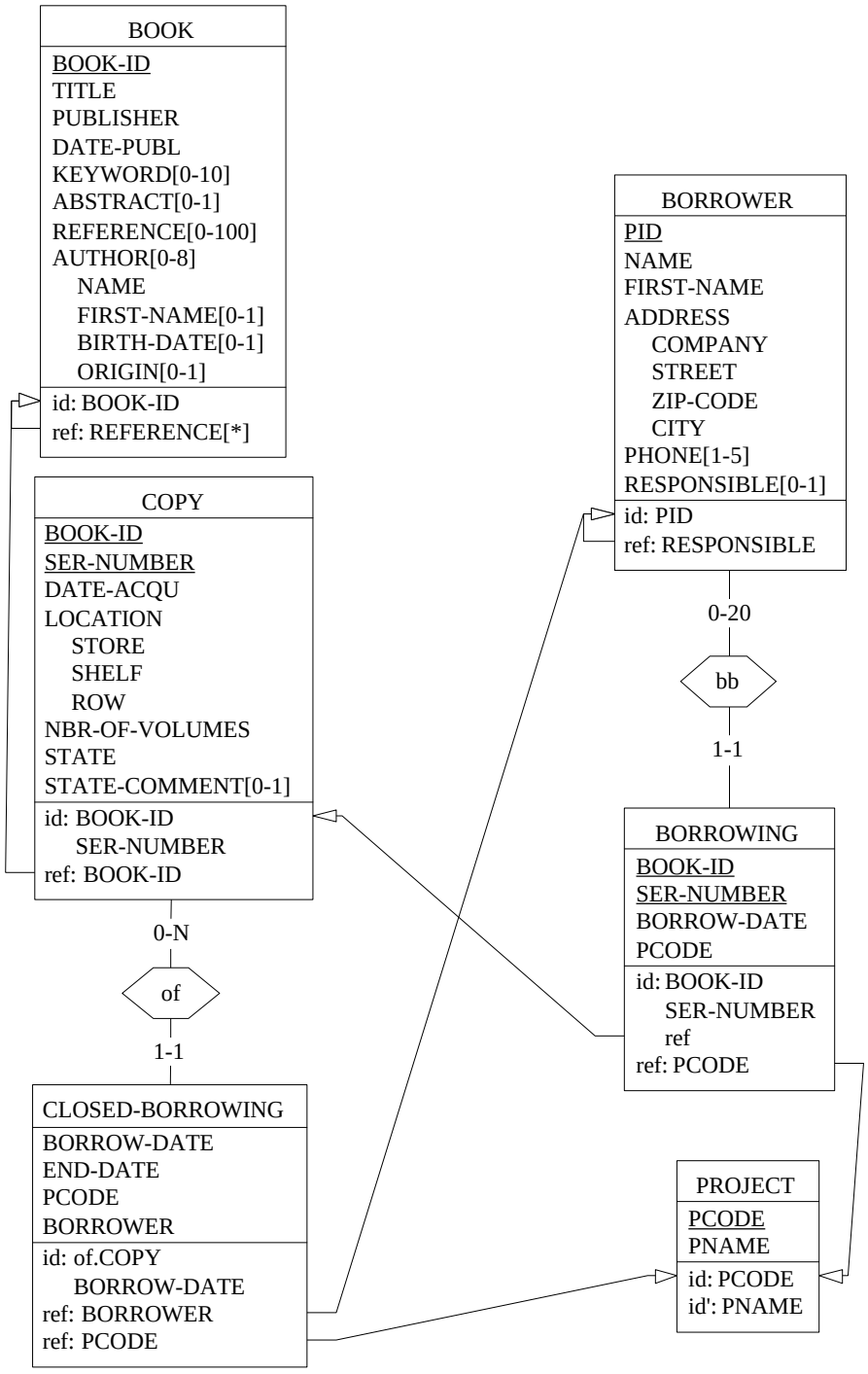
# 1.2 The abstract COBOL physical schema (refined 1)



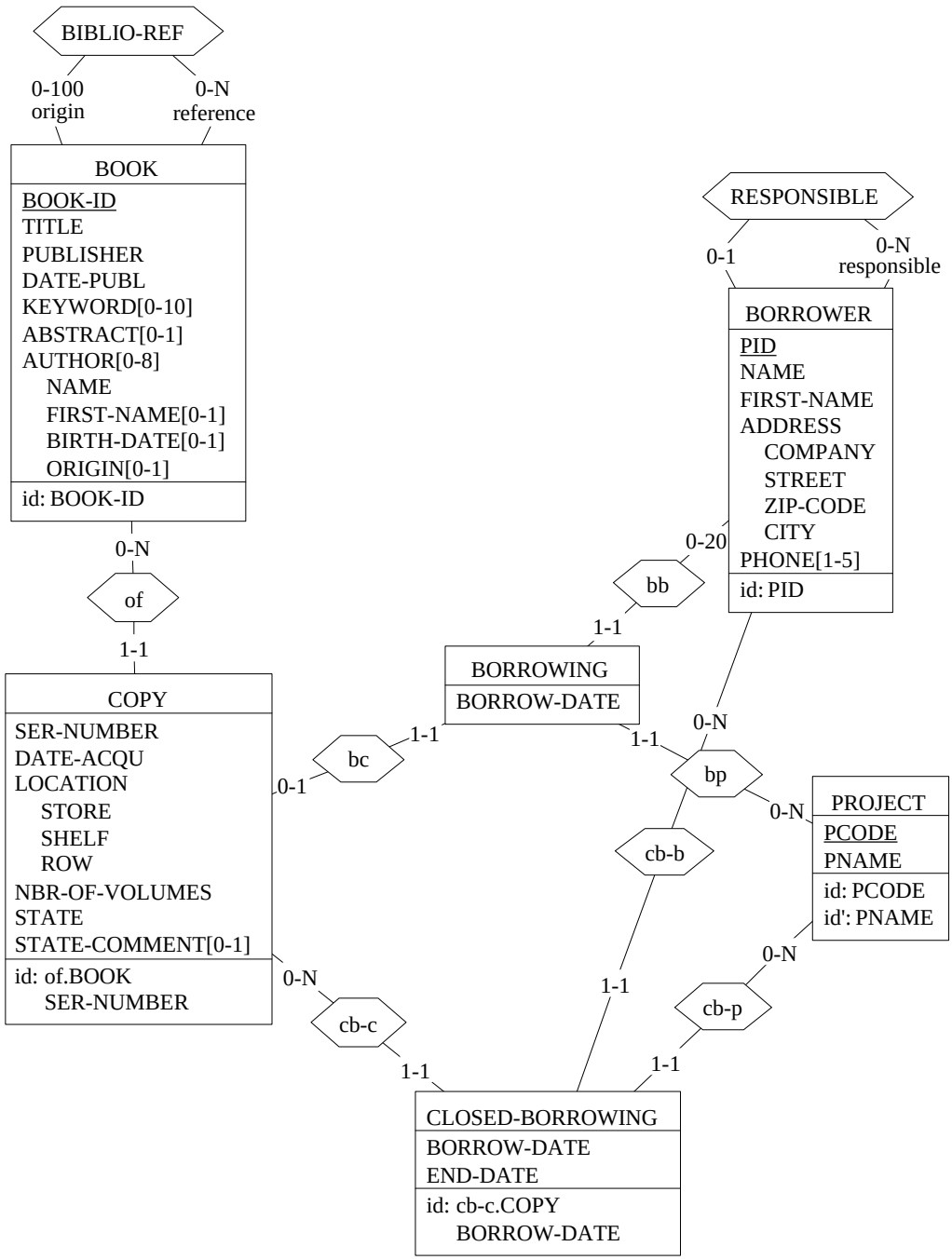
# 1.3 The COBOL Logical Schema



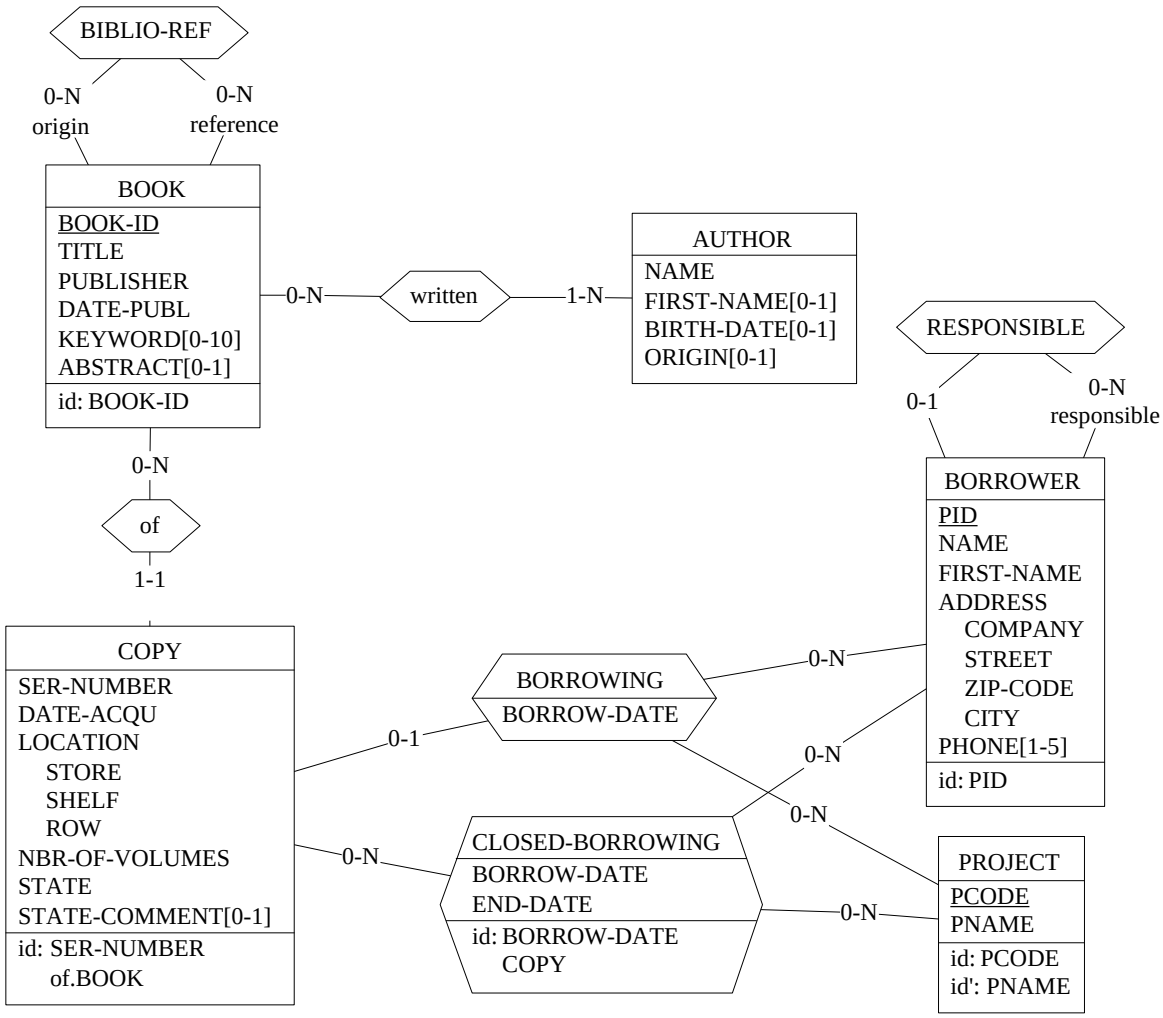
# 1.4 The Conceptual schema (step 1)



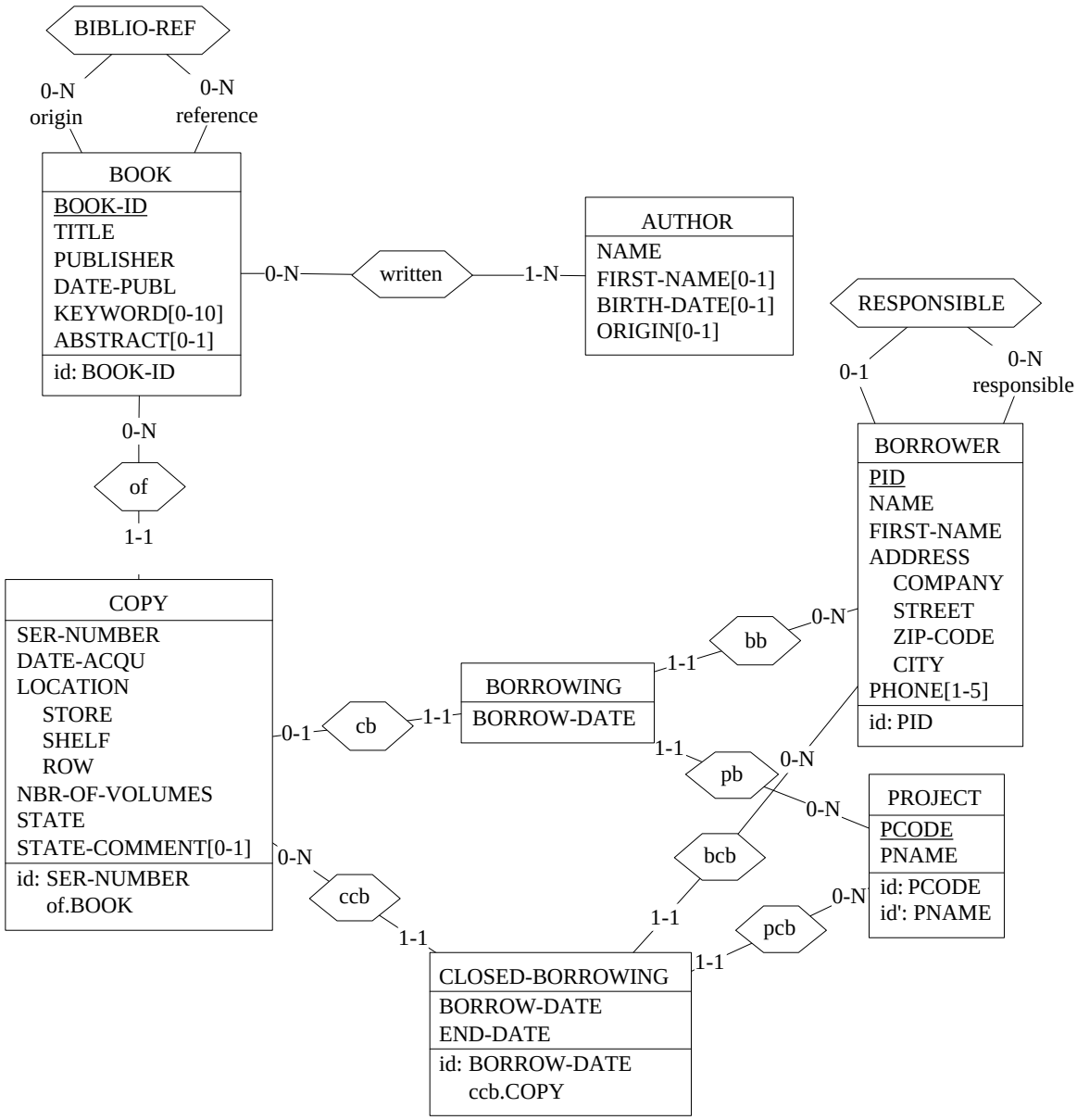
# 1.5 The Conceptual schema (step 2)



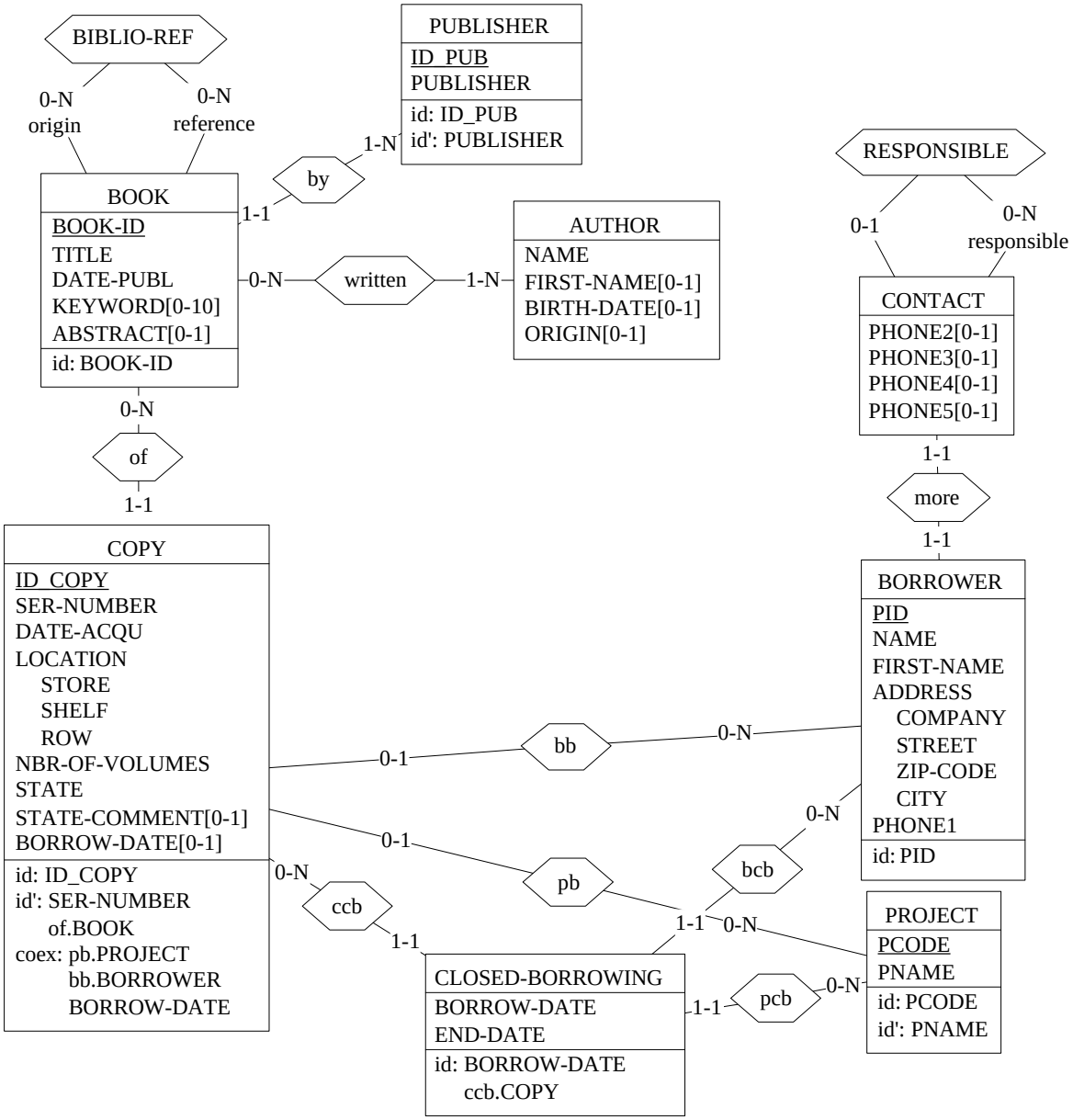
# 1.6 The Conceptual schema (step 3 - final)



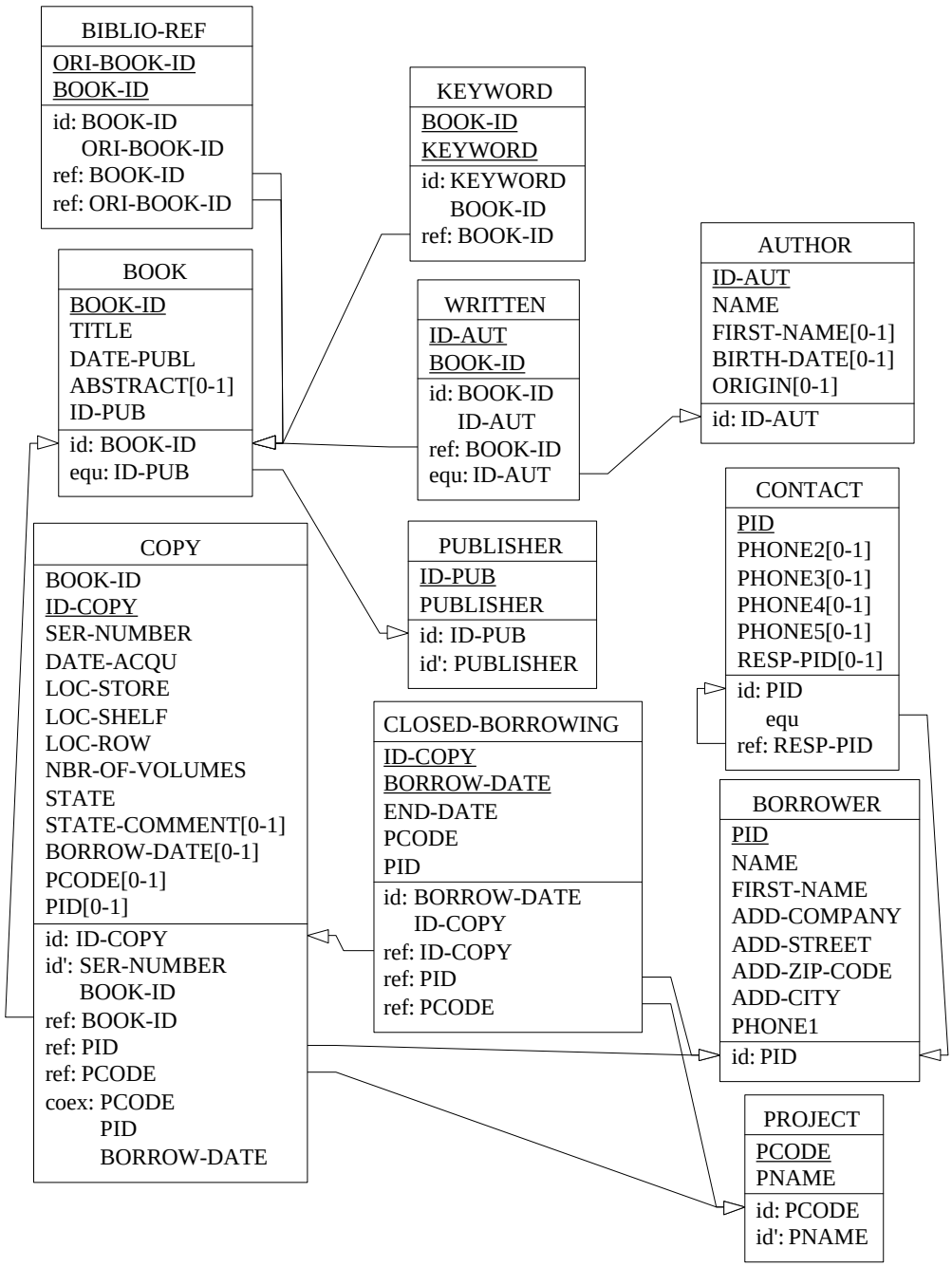
# 2.1 The simplified (binary) Conceptual schema



# 2.2 The Optimized schema



# 2.3 The Optimized SQL Logical schema



# Part 10

## BIBLIOGRAPHY

**ABITEBOUL,87**

Abiteboul, S., Beeri, K., *On the power of languages for the manipulation of complex objects*, INRIA technical report, 1987

**AIKEN,94a**

Aiken, P., Piper, P., *Data Reverse Engineering's Role in Enterprise Integration*, in Proc. of the 4th Reengineering Forum "Reengineering in Practice", Victoria, Canada, 1994

**AIKEN,94b**

Aiken, P., Joseph, M., *Evaluating Data Reverse Engineering Investments*, in Proc. of the 4th Reengineering Forum "Reengineering in Practice", Victoria, Canada, 1994

**ANDANI,91**

Andany, J., Léonard, M., Palissier, C., *Management of Schema Evolution in Databases*, in Proc. of the 17th Int. Conf. on VLDB, Morgan-Kaufmann, pp. 161-170, 1991

**ANDERSSON,94a**

Andersson, M., *Extracting Conceptual Schemas from Legacy Information Systems through Reverse Engineering*, in Proc. of the 4th Reengineering Forum "Reengineering in Practice", Victoria, Canada, 1994

**ANDERSSON,94b**

Andersson, M., *Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering*, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, 1994

**ARNOLD,93**

Arnold, S., A., (Ed.) *Software Reengineering*, IEEE Computer Society Press, 1993

**BALZER,81**

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

**BATINI,83**

Batini, C., Lenzerini, M., Moscarini, M., *View integration, in Methodology and tools for data base design*, Ceri, S., (Ed.)North-Holland, 1983

**BATINI,92**

Batini, C., Ceri, S., Navathe, S., B., *Conceptual Database Design*, Benjamin/ Cummings, 1992

**BATINI,93**

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

**BEERI,86**

Beeri, K., Kiefer, *An integrated approach to logical design of relational database schemes*, ACM TODS, Vol. 11, N° 2, 1986

**BELLAHSENE,93**

Bellahsene, Z., *An Active Meta-Model for Knowledge Evolution in an Object-oriented Database*, in Proc. of CAiSE'93, Springer-Verlag, 1993

**BERT,85**

Bert, M., N., and al., *The logical design in the DATAID Project : the EASYMAP system*, in *Computer-Aided Database Design : the DATAID Project*, Albano and al. (Ed.), North-Holland, 1985

**BLAHA,95**

Blaha, M.R., W.J. Premerlani, W., J., *Observed Idiosyncracies of Relational Database designs*, in Proc. of the 2nd IEEE Working Conf. on Reverse Engineering, Toronto, July 1995, IEEE Computer Society Press, 1995

**BOLOIS,94**

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

**CASANOVA,83**

Casanova, M., Amarel de Sa, J., *Designing Entity Relationship Schemas for Conventional Information Systems*, in Proc. of Entity-Relationship Approach, pp. 265-278, 1983

**CASANOVA,84**

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

**CHEN,76**

Chen, P., *The entity-relationship model - toward a unified view of data*, ACM TODS, Vol. 1, N° 1, 1976

**CHIANG,94a**

Chiang, R., H., Barron, T., M., Storey, V., C., *Performance Evaluation of Reverse Engineering Relational Databases into Extended Entity-Relationship Models*, in Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, Springer-Verlag, 1994

**CHIANG,94b**

Chiang, R., H., Barron, T., M., Storey, V., C., *Reverse Engineering of Relational Databases : Extraction of an EER model from a relational database*, Journ. of Data and Knowledge Engineering, Vol. 12, No. 2 (March 94), pp107-142, 1994

**CHUNG,91**

Chung, L., Katalagarianos, P., Marakakis, M., Mertikos, M., Mylopoulos, J., Vassiliou, Y., *From information system requirements to designs : A mapping framework*, Information Systems, Vol. 16, pp. 429-461, 1991

**CoopIS,93**

First Int. Conference on *Cooperative Information Systems*, 1993

**CoopIS,94**

Second Int. Conference on *Cooperative Information Systems*, 1994

**CoopIS,95**

Third Int. Conference on *Cooperative Information Systems*, 1995

**DARWEN,93**

Darwen, H., Date, C., J., *Relation-valued Attributes*, in Date, C., J., Darwen, H., *Relational Database Writings 1989-1991*, Addison-Wesley, 1993

**D'ATRI,84**

D'Atri, A., Sacca, D., *Equivalence and Mapping of Database Schemes*, in Proc. 10th VLDB conf., Singapore, 1984

**DATE,94**

Date, C., J., *An Introduction to Database Systems*, Volume 1, Addison-Wesley, 1994

**DAVIS,85**

Davis, K., H., Arora, A., K., *A Methodology for Translating a Conventional File System into an Entity-Relationship Model*, in Proc. of Entity-Relationship Approach, October, IEEE/North-Holland, 1985

**DAVIS,88**

Davis, K., H., Arora, A., K., *Converting a Relational Database model to an Entity Relationship Model*, in Proc. of Entity-Relationship Approach : a Bridge to the User, North-Holland, 1988

**DAVIS,94**

Davis, K., H., *August-II: Software Reverse Engineering Tool Produces Flexible Conceptual Data Model*, in Proc. of the 4th Reengineering Forum "Reengineering in Practice", Victoria, Canada, 1994

**DELOBEL,73**

Delobel, C., Casey, R., G., *Decomposition of a data base and the theory of Boolean switching functions*, IBM J. Res. and Develop. 17, 5 (Sept. 1973), pp. 374-386

**DEHENEFFE,75**

Deheneffe, C., Hainaut, J-L, Tardieu, H., *The Individual Model*, in Proc. of the Intern. Workshop on Data Structure Models for Information Systems, Namur, May, 1974, Presses Universitaires de Namur, 1975.

**DESCLAUX,92**

Desclaux, C., Ribault, M., Cochinal, S., *RE-ORDER : A Reverse engineering Methodology*, in Proc. 5th Int. Conf. on Software Engineering and Applications, Toulouse, 7-11 December, pp. 517-529, EC2 Publish. 1992

**DETROYER,93**

De Troyer, O., *On data schema transformation*, PhD Thesis, University of Tilburg, Tilburg, The Netherlands

**DS-5,92**

IFIP WG 2.6 Conference on *Semantics of Interoperable Database Systems* (Lorne, Victoria, Australia), Nov. 1992

**EDWARDS,95**

Edwards, H., M., Munro, M., *Deriving a Logical Model for a System Using Recast Method*, in Proc. of the 2nd IEEE WC on Reverse Engineering, Toronto, July 1995, IEEE Computer Society Press, 1995

**ELMASRI,94**

Elmasri, R., Navathe, S., *Fundamentals of Database Systems*, Benjamin-Cummings, 1994

**EWALD,93**

Ewald, C., A., Orlowska, M., E., *A Procedural Approach to Schema Evolution*, in Proc. of CAiSE'93, Springer-Verlag, 1993

**FAGIN,77**

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

**FAGIN,81**

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

**FGCS,94**

Workshop on *Heterogeneous Cooperative Knowledge-bases*, Dec. 1994, Tokyo, Japan

**FIKAS,85**

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

**FONG,93**

Fong, J., Ho, M., *Knowledge-based Approach for Abstracting Hierarchical and Network Schema Semantics*, in Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, Springer-Verlag, 1994

**FONKAM,92**

Fonkam, M., M., Gray, W., A., *An approach to Eliciting the Semantics of Relational Databases*, in Proc. of 4th Int. Conf. on Advance Information

Systems Engineering - CAiSE'92, pp. 463-480, May, LNCS, Springer-Verlag, 1992

**GARDARIN,94**

Gardarin, G., *Translating relational to object databases*, Engineering of Information Systems, Vol. 2, No.3, pp317-346, Hermes, 1994

**GIRAUDIN,85**

Giraudin, J-P., Delobel, C., Dardailler, P., *Eléments de construction d'un système expert pour la modélisation progressive d'une base de données*, in Proc. of Journées Bases de Données Avancées, Mars, 1985

**HAINAUT,81**

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

**HAINAUT,89**

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

**HAINAUT,90**

Hainaut, J-L., *Entity-Relationship models : formal specification and comparison - Tutorial*, in Proc. of Entity-Relationship Approach : the Core of Conceptual Modelling, 1990, North-Holland, 1991

**HAINAUT,91a**

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

**HAINAUT,91b**

Hainaut, J-L, *Database Reverse Engineering, Models, Techniques and Strategies*, in Preproc. of the 10th Conf. on Entity-Relationship Approach, San Mateo (CA), 1991

**HAINAUT,92a**

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

**HAINAUT,92b**

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

**HAINAUT,92c**

Hainaut, J-L., Cadelli, M., Decuyper, B., Marchand, O., *TRAMIS : a transformation-base database CASE tool*, in Proc. 5th Int. Conf. on Software Engineering and Applications, Toulouse, 7-11 December 1992, EC2 Publish., 1992

**HAINAUT,93a**

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

**HAINAUT,93b**

Hainaut, J-L, Chandelon M., Tonneau C., Joris M., *Transformational techniques for database reverse engineering*, in Preproc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, ER Institute, 1993

**HAINAUT,94a**

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

**HAINAUT,94b**

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

**HAINAUT,95a**

Hainaut, J-L, Englebert, V., Henrard, J., Hick J-M., Roland, D., *Transformation-based CASE tool for Database Reverse Engineering*, in Proc of the 6th European Workshop on Next Generation CASE tools, CAiSE•95, Jÿvaskÿla (Finland), June 1995

**HAINAUT,95b**

Hainaut, J-L, Englebert, V., Henrard, J., Hick J-M., Roland, D., *Requirements for Information System Reverse Engineering Support*, in Proc. of the 2nd IEEE WC on Reverse Engineering, Toronto, July 1995, IEEE Computer Society Press, 1995

**HAINAUT,95c**

Hainaut, J-L, *Database Reverse Engineering - Problems, Techniques and CASE tools*, Tutorial Notes, CAiSE•95 Conference, Jÿvaskÿla (Finland), June 1995

**HALL,92**

*Software Reuse and Reverse Engineering in Practice*, Hall, P., A., V. (Ed.), Chapman&Hall, 1992

**HALPIN,95**

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

**HULL,87**

Hull, *A survey of theoretical research on typed complex database objects*, in *International Lecture Series in Computer Science*, Academic Press, 1987

**IEEE,90**

*Special issue on Reverse Engineering*, IEEE Software, January, 1990

**JACOBSON,91**

Jacobson, I., Lindström, F., *Re-engineering of old systems to an object-oriented architecture*, in Proc of OOPSLA'91, pp.340-350, 1991

**JAJODIA,83**

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

**JARKE,92**

Jarke, M., et al., DAIDA : *An environment for evolving information systems*, ACM Trans. on Information Systems, Vol. 10, Jan. 1992

**JARKE,93**

Jarke, M., et al., *DAIDA : Requirements Engineering : An Integrated View of Representation, Process and Domain*, NATURE Report Series, No. 93-07, available from <natrep@informatik.rwth-aachen.de>

**JEUSFELD,94**

Jeusfeld, M., A., Johnen, U., A, *An executable Meta-model for Reengineering of Database Schemas*, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, 1994

**JOHANNESSON,89**

Johannesson, P., Kalman, K., *A Method for Translating Relational Schemas into Conceptual Schemas*, in Proc. of the 8th Entity-Relationship Approach, Toronto, North-Holland, 1990

**JOHANNESSON,93**

Johannesson, P., *Schema Integration, Schema translation, and Interoperability in Federated Information Systems*, PhD Thesis, University of Stockholm, 1993

**JONER,95**

Joner, T., Song, I-Y, *Binary representations of Ternary Relationships in ER Conceptual Modelling*, in Proc. of the 14th Int. Conf. on ER/OO Modelling (ERA), Dec. 1995

**JORIS,92**

Joris, M., Van Hoe, R., Hainaut, J-L., Chandelon M., Tonneau C., Bodart F. et al., *PHENIX : methods and tools for database reverse engineering*, in Proc. 5th Int. Conf. on Software Engineering and Applications, Toulouse, 7-11 December 1992, EC2 Publish., 1992

**KOBAYASHI,86**

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

**KOZACZYNSKY,87**

Kozaczynsky, Lilien, *An extended Entity-Relationship (E2R) database specification and its automatic verification and transformation*, in Proc. of Entity-Relationship Approach, 1987

**KRIEG,89**

Krieg-Brückner, B., *Algebraic Specification and Functionals for Transformational Program and Meta Program Development*, in Proc. of the TAPSOFT Conf. LNCS 352, Springer-Verlag, 1989

**LEVENE,92**

Levene, M., *The Nested Universal Relation Database Model*, LNCS 595, Springer-Verlag, 1992

**LIEN,82**

Lien, Y., E., *On the equivalence of database models*, JACM, 29, 2, April 1982

**LING,85**

Ling, T., W., *A Normal Form for Entity-Relationship Diagrams*, in Proc. of the 4th Entity-Relationship Approach, North-Holland, 1985

**LING,89**

Ling, T., W., *External schemas of Entity-Relationship based DBMS*, in Proc. of Entity-Relationship Approach : a Bridge to the User, North-Holland, 1989

**LING,94**

Ling, T., W., Lee, M., L., *Semantic Dependencies in Data Modelling and Database Reverse Engineering*, in Proc. of International Symposium on Advanced Database Technologies and Their Integration (ADTI'94), Nar (Japan), 1994

**MAIER,83**

Maier, *The Theory of Relational Databases*, Computer Science Press, 1983

**MARKOWITZ,90**

Markowitz, K., M., Makowsky, J., A., *Identifying Extended Entity-Relationship Object Structures in Relational Schemas*, IEEE Trans. on Software Engineering, Vol. 16, No. 8, 1990

**MISSAOUI,95**

Missaoui, R., Gagnon, J., *Mapping an Extended Entity-Relationship Schema into a Schema of Complex Objects*, in Proc. of the 14th Int. Conf. on ER/OO Modelling (ERA), Dec. 1995

**MOTRO,87**

Motro, *Superviews: Virtual integration of Multiple Databases*, IEEE Trans. on Soft. Eng. SE-13, 7, July 1987

**MUNTZ,94**

Muntz, A., *A Requirement-Based Approach to Data Modeling and Re-engineering*, in Proc. of the 20th Conf. on VLDB, Santiago, 1994

**MYLOPOULOS,92**

Mylopoulos, J., Chung, L., Nixon, B., *Representing and Using Nonfunctional requirements : A Process-Oriented Approach*, IEEE TSE, Vol. 18, No. 6, June 1992

**NAVATHE,80**

Navathe, S., B., *Schema Analysis for Database Restructuring*, in ACM TODS, Vol.5, No.2, June 1980

**NAVATHE,84**

Navathe, S., B., Sashidhar, T., Elmasri, R., *Relationship Merging in Schema Integration*, in Proc. 10th VLDB conf., 1984

**NAVATHE,88**

Navathe, S., B., Awong, A., *Abstracting Relational and Hierarchical Data with a Semantic Data Model*, in Proc. of Entity-Relationship Approach : a Bridge to the User, North-Holland, 1988

**NGUYEN,89**

Nguyen, G., T., Rieu, D., *Schema evolution in object-oriented database systems, Data & Knowledge Engineering*, 4 (1989) pp. 43-67, North-Holland

**NIJSSEN,89**

Nijssen, G., M., Halpin, T., A., *Conceptual Schema and Relational Database Design*, Prentice-Hall, 1989 (see 2nd Edition too)

**NILSSON,85**

Nilsson,E., G., *The Translation of COBOL Data Structure to an Entity-Rel-type Conceptual Schema*, in Proc. of Entity-Relationship Approach, October, IEEE/North-Holland, 1985

**PARTSCH,83**

Partsch, H., Steinbrüggen, R., *Program Transformation Systems*, Computing Surveys, Vol. 15, No. 3, 1983

**PETIT,94**

Petit, J-M., Kouloumdjian, J., Bouliaut, J-F., Toumani, F., *Using Queries to Improve Database Reverse Engineering*, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, 1994

**POTTS,88**

Potts, C., Bruns, G., *Recording the Reasons for Design Decisions*, in Proc. of ICSE, IEEE, 1988

**PREMERLANI,93**

W.J. Premerlani, W., J., Blaha, M.R., *An Approach for Reverse Engineering of Relational Databases*, in Proc. of the IEEE Working Conf. on Reverse Engineering, Baltimore, May 1993, IEEE Computer Society Press, 1993

**RAUH,95**

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

**REINER,86**

Reiner, D., Brown, G., Friedell, M., Lehman, J., McKee, R., Rheingans, P., Rosenthal, A., *A Database Designer's Worbench*, in Proc. of Entity-Relationship Approach, 1986

**RIDE-IMS,91**

1st Int. Workshop on *Research Issues in Data Engineering : Interoperability in Multidatabase Systems* (Kyoto, Japan), IEEE Comp. Soc. Press, 1991

**RIDE-IMS,93**

3rd Int. Workshop on *Research Issues in Data Engineering : Interoperability in Multidatabase Systems* (Vienna, Austria), IEEE Comp. Soc. Press, 1993

**RISSANEN,77**

Rissanen, *Independent components of relations*, ACM TODS, Vol. 2, N°4, 1977

**ROCK,90**

Rock-Evans, R., *Reverse Engineering : Markets, Methods and Tools*, OVUM report, 1990

**RODDICK,92**

Roddick, J., F., *Schema Evolution in Database Systems - An Annotated Bibliography*, SIGMOD Record, Vol. 21, No. 4, pp. 35-40, Dec. 1992

**RODDICK,93**

Roddick, J., F., Craske, N., G., Richards, T., J., *A taxonomy for Schema Versioning Based on the Relational and Entity-Relationship Models*, in Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, ER Institute, 1993

**ROSENTHAL,88**

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

**ROSENTHAL,94**

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

**RUSINKEIWICZ,92**

Rusinkeiwicz, M., Sheth, A., *Multidatabase Applications : Semantics and System Issues*, Tutorial notes, 18th VLDB Conf., Vancouver (Canada), Aug. 1992

**SABANIS,92**

Sabanis, N., Stevenson, N., *Tools and Techniques for Data Remodelling Cobol Applications*, in Proc. 5th Int. Conf. on Software Engineering and Applications, Toulouse, 7-11 December, pp. 517-529, EC2 Publish. 1992

**SCHEK,86**

Schek, H-J., Scholl, M., H., *The relational model with relation-valued attributes*, Information Systems, 11, pp. 137-147, 1986

**SCHNEIDERMAN,82**

Schneiderman, B., Thomas, G., *An architecture for Automatic Relational Database System Conversion*, ACM TODS, Vol. 7, No. 2, pp. 235-257, 1982

**SELFRIDGE,93**

Selfridge, P., G., Waters, R., C., Chikofsky, E., J., *Challenges to the Field of Reverse Engineering*, in Proc. of the 1st WC on Reverse Engineering, pp.144-150, IEEE Computer Society Press, May, 1993

**SHETH,90**

Sheth, A., Larson, J., *Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases*, ACM Computing Surveys, Vol.22, No.3, Sept. 1990

**SHOVAL,93**

Shoval, P., Shreiber, N., *Database Reverse Engineering : from Relational to the Binary Relationship Model*, Data and Knowledge Engineering, Vol. 10, No. 10, 1993

**SIGNORE,94**

Signore, O, Loffredo, M., Gregori, M., Cima, M., *Reconstruction of ER Schema from Database Applications: a Cognitive Approach*, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, 1994

**SPACCAPIETRA,91**

Spaccapietra, S., Parent, C., *Conflicts and correspondance assertions in interoperable databases*, SIGMOD Records, Dec. 1991

**SPACCAPIETRA,92**

Spaccapietra, S., Parent, C., *View Integration : A Step Forward in Solving Structural Conflicts*, IEEE Trans. on Knowledge and Data Engineering, October, 1992

**SPRING,90**

Springsteel, F., N., Kou, C., *Reverse Data Engineering of E-R designed Relational schemas*, in Proc. of Databases, Parallel Architectures and their Applications, March, 1990

**STEEL,94**

Steel, P., Nolan, M., Ceddia, J., Zaslavsky, A., *Identifying Domains in Relational Databases to Support Reverse Engineering*, in Proc. of Baltic Workshop on National Infrastructure Databases, 1994

**TEOREY,90**

Teorey, T. J., *Database Modeling and Design*, Morgan Kaufmann, 1990

**TEOREY,94**

Teorey, T. J., *Database Modeling and Design : the Fundamental Principles*, Morgan Kaufmann, 1994

**TSENG,88**

Tseng, V., P., Mannino, M., V., *Inferring Database Requirements from Examples in Forms*, in Proc. of the 7th Int. Conf. on the Entity-Relationship Approach, North-Holland, 1989

**TSUDA,91**

Tsuda, K., Yamamoto, K., Hirakawa, M., Tanaka, M., Ichikawa, T., *MORE : An Object-Oriented Data Model with a Facility for Changing Object Structure*, IEEE Trans. on Knowl. and Data Eng., Vol. 3, No. 4, Dec. 1991

**ULLMAN,89**

Ullman, J., D., *Principles of Data- and Knowledge-base Systems* (Vol I & II), Computer Science Press, 1989

**VANBOMMEL,93**

van Bommel, P., *Database Design Modifications based on Conceptual Modelling*, in Proc. of the 3rd European-Japanese Seminar on Information Modelling and Knowledge Bases, May 1993, Budapest, pp. 276-288 (Preprint)

**VANZUYLEN,91**

Van Zuylen, H., *The REDO Handbook - A compendium of reverse engineering for Software Maintenance*, REDO Project report 2487-TN-WL-1027, Nov. 1991

**VIDAL,95**

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

**VERMEER,95**
Vermeer, M., Apers, P., *Reverse Engineering of Relational Databases*, in Proc. of the 14th Int. Conf. on ER/OO Modelling (ERA), Dec. 1995

**WATERS,93**
Waters, R., C., Chikofsky, E., J., (Eds), Proc. of the IEEE Working Conf. on Reverse Engineering, Baltimore, May 1993, IEEE Computer Society Press, May 1993

**WEISER,84**
Weiser, M., *Program Slicing*, IEEE TSE, Vol. 10, 1984, pp 352-357

**WHDS,89**
Workshop on *Heterogeneous Database Systems*, Chicago, Dec. 1989

**WIDS,93**
Workshop on *Interoperability of Database Systems and Database Applications*, Fribourg (CH), October, 1993

**WILLS,95**
Wills, L., Newcomb, P., Chikofsky, E., (Eds), Proc. of the 2nd IEEE Working Conf. on Reverse Engineering, Toronto, July 1995, IEEE Computer Society Press, 1995

**WINANS,90**
Winans, J., Davis, K., H., *Software Reverse Engineering from a Currently Existing IMS Database to an Entity-Relationship Model*, in Proc. of Entity-Relationship Approach : the Core of Conceptual Modelling, pp. 345-360, October, North-Holland, 1990