Braga Summer School on Generative and Transformational
Techniques in Software Engineering
July 2005

The Transformational Approach to Database Engineering

Jean-Luc Hainaut

University of Namur
Institut d'informatique
LIBD - Laboratory of Database Application Engineering
www.info.fundp.ac.be/libd

Introduction

This tutorial is not a mistake!

This school is about *Software engineering*

Every large business-oriented software includes a *database*

So, *Database engineering* is a part of *Software engineering*

Contents

Transformational database engineering in a nutshell

(*introductory demonstration*)

Contents

- **Introduction**
- **Modelling data structures**
- **Schema transformations**
- **Semantics preservation properties of transformations**
- **Typology of practical transformations**
- **Transformational modelling of database engineering processes**
- **Schema transformations in CASE tools**
- **Conclusions and perspectives**
- **Appendices**
  - **Semantics of the GER**
  - **Proving the reversiblity of GER transformations**
  - **IDMS migration: a case study**

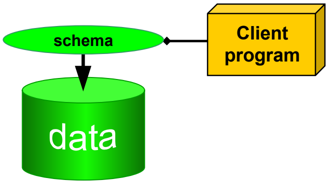
Introduction

Introduction

What is a *Database?*

The structured collection of the data necessary to

- **keep the memory of an organization (structures, rules and facts)**
- **to act as a reliable and efficient data server for an application system**



LIBD

Introduction

Programs and databases

- 1 database for N programs
- database is independent of the application programs
- the database is built *before* developing the programs, and generally survives them,
- very long life span (20 to 30 years is not uncommon)
- disputable flexibility
- intrinsically *not Object-oriented* (despite some pathetic attempts in SQL3);

LIBD

Introduction

First remark: ***a model is not (always) a model***

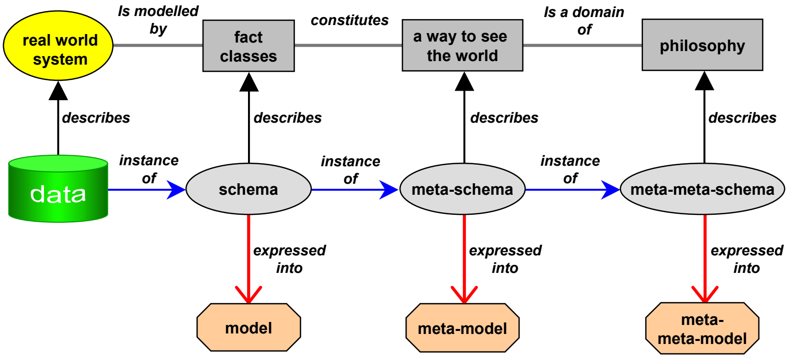
Introduction

- *in UML:*
  - a **meta-model** is a structured system of abstract constructs that can be used to describe any situation of an application domain
  - a **model** is an artefact using the constructs of a meta-model, and that specifies the structures of a definite situation of an application domain
- *in the Database realm:*
  - a **model** is a structured system of abstract constructs that can be used to describe any situation of an application domain;

    can be given N notations/languages;
  - a **schema** is an artefact using the constructs of a model, and that specifies the structures of one definite situation of an application domain
- *Examples*: the relational model
  the Entity-relationship model
  the schema of the GTTSE'05 database.

Introduction

Specialization of Jean Bézivin's framework



Introduction

Is the database domain so complex anyway?

Introduction

Some facts about databases (1)

- a company may use more than 10 DMSs (Data Management Systems) to implement its information system;
- a new version of a DMS appears every 4 years, most often involving changes in the data and in the programs;
- a database may be used by several thousands programs;
- the schema of a database may include more than 1,000 entity types and 30,000 attributes (technically, 1,250 files/tables and 40,000 fields/columns); SAP = 15,000-30,000 tables and 200,000 columns;
- some database schemas have become so large and complex that no single data administrator can master them any longer;

Introduction

Some facts about databases (2)

- the precise description of one entity type and its attributes may span from 1 to 100 pages (what does a banking company mean by *product*?)
- the functional documentation of a large database may (*should*) comprise more than 5,000 pages;
- the SQL-DDL code of a database (tables, constraints, indexes, triggers, checks, etc.) may comprise 200,000 LOC (5,000 pages);
- however, many databases have no documentation.

Introduction

Some facts about databases (3)

- database schemas share some interesting properties with programs:
  - bugs
  - awkward design
  - dead parts (never used but the system crashes without them)
  - obscure sections (*terra incognita*)
  - (*nearly*) duplicated sections
  - developed on obsolete platforms
  - poorly documented (if ever)
- corrective, preventive and adaptive maintenance (*no added value*) of an program/database system may require more than 50% of development effort;

Introduction

What is *Database Engineering?*

Technologies, **theories**, **models**, **techniques**, **methods** and **tools** dedicated to

- specifying, modeling, designing, implementing, optimizing **databases**
- extracting, migrating, web-publishing data **from a database**
- reverse engineering **legacy databases**
- maintaining, reengineering, evolving, migrating **existing databases**
- integrating, federating, wrapping, mediating a **set of independent databases**

Introduction

Is the database domain plagued by the MDA/MDE hype?

In some way:

BE has been intrinsically MDE-compliant and transformational, for more than 30 years

Introduction

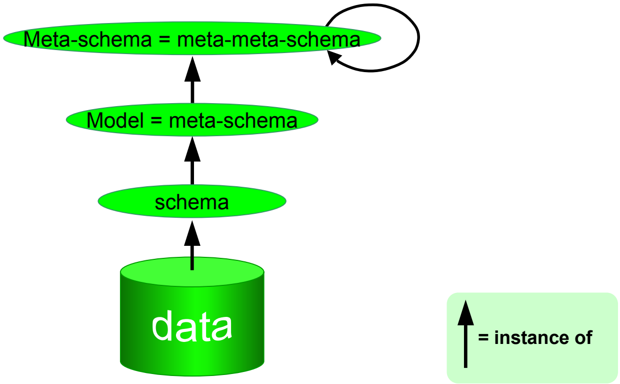
Database Engineering and MDA/MDE

- *Information Algebra*, CODASYL, Comm. ACM , **1962**
- *A relational model of data for large shared data banks*, Codd, Comm. ACM, **1970**
- *The Individual Model*, first version of the Merise methodology,**1974**, first proposal of a 3-level methodology;
- *DIAM II: multilevel description of database structures* (M. Senko), **1974**
- *The Entity-relationship model* (P. Chen), **1976**
- *The ANSI/X3/SPARC DBMS framework* (conceptual, logical, physical, external), Information System, **1978**
- ISO/TC97/SC5 proposals (identification of a hierarchy of modeling abstractions), **1982**

Introduction

The schemas and models of a database

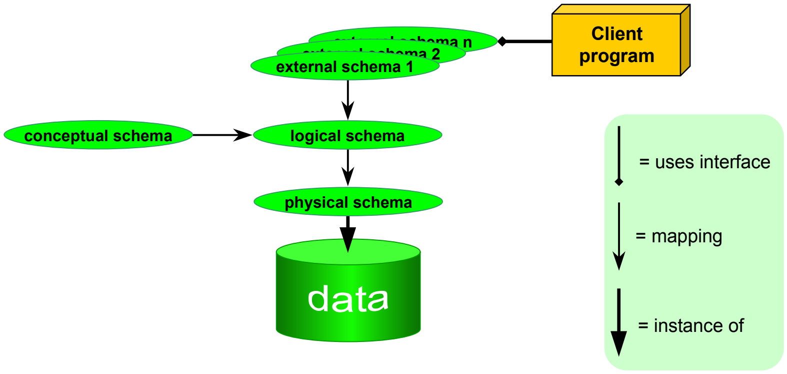
Models and meta-models



Introduction

The schemas and models of a database

Models in the ANSI-Sparc architecture


external schema n
external schema 2
external schema 1
Client program
conceptual schema
logical schema
physical schema
data

= uses interface

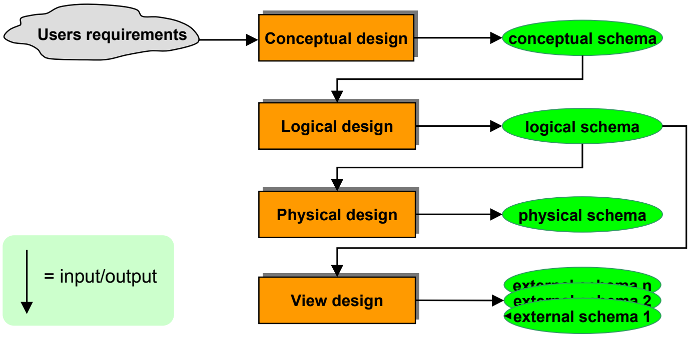
= mapping

= instance of

Introduction

The schemas and models of a database

Models in methodologies


Users requirements
Conceptual design
conceptual schema
Logical design
logical schema
Physical design
physical schema
View design
external schema n
external schema 2
external schema 1

= input/output

Introduction

Rule-based *vs* Transformation-based engineering

Introduction

- **Rule-based engineering**

  the target specification is produced following a set of translation rules.

- **Transformation-based engineering**

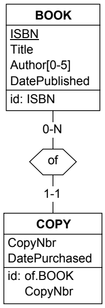
  the target specification is produced by application of a chain of substitution operators to the source specifications.

LIBD

Introduction

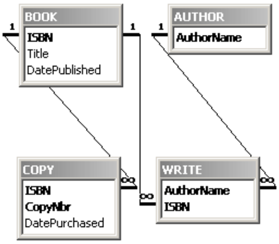
Rule-based view of *Database Engineering*

Example*: producing a relational schema from a conceptual schema*



Conceptual schema (ER)

⇒



Physical schema (MS Access)

LIBD

Introduction

Rule-based view of *Database Engineering*

Natural procedure*: through translation rules*

Conceptual schema	Physical constructs
Entity type **E**	table **E**
Level-1 multivalued atomic attribute **A** of entity type **E**	table **A**, comprising: column **A**; primary key made up of **A**. table **EA**, comprising column(s) copied from the primary key of table **E**; column copied from the primary key of table **A**; primary key comprising all these columns.
relationship-type **R** from **B** (with card. 1-1) to **A** (with card. 0-N)	in table **B**, column(s) copied from the primary key of table **A**; foreign key to **A** comprising these columns; if R was part of a candidate (primary) key of B, then add these attributes to the key.

Introduction

Rule-based view of *Database Engineering*

OK, but what if:

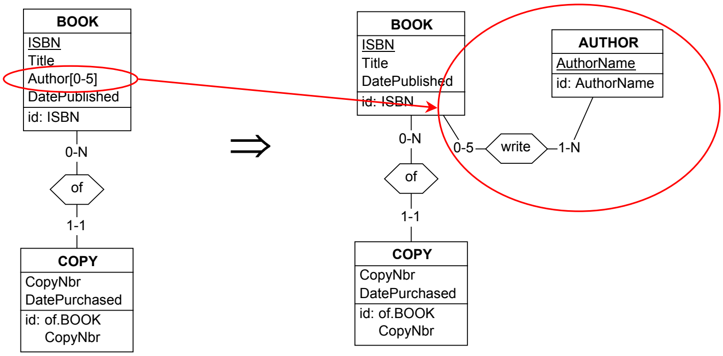
- attribute A is at level 2, 3, …?
- attribute A is not atomic?
- relationship type R is many-to-many, or one-to-one, or N-ary?
- the primary key of E has not been translated yet (e.g., it comprises a FK still untranslated)?

⇒ **Combinatorial explosion and complexity of the set of rules.**

Introduction

Transformation-based view of *Database Engineering*

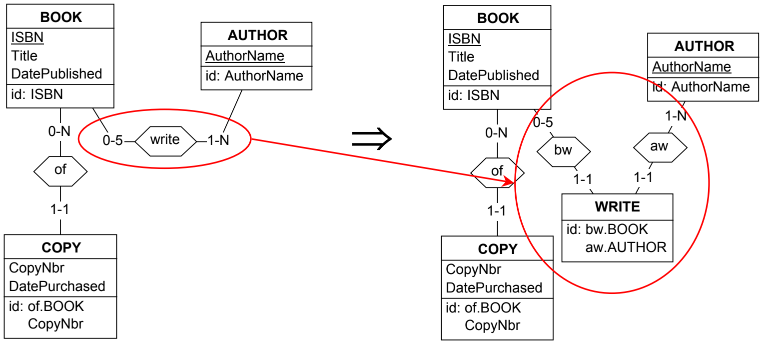
Transforming the multivalued attribute *Author*



Introduction

Transformation-based view of *Database Engineering*

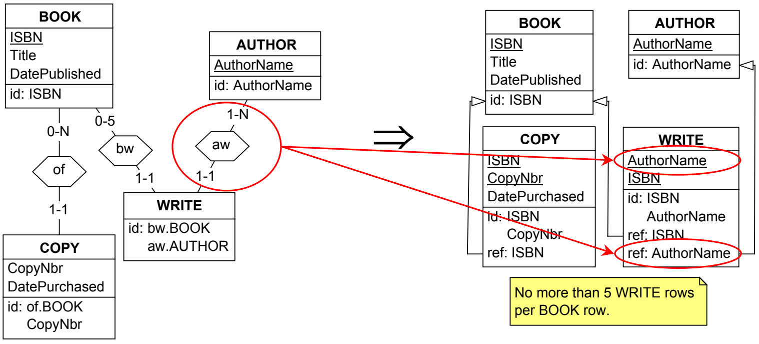
<u>Transforming</u> the *many-to-many* relationship type *write*



Introduction

Transformation-based view of *Database Engineering*

<u>Transforming</u> the *one-to-many* relationship type *aw* (and the others)

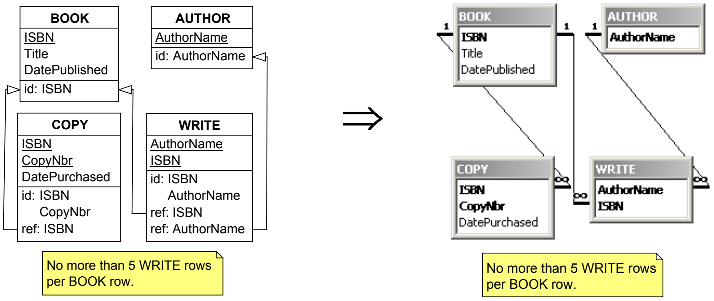


No more than 5 WRITE rows per BOOK row.

Introduction

Transformation-based view of *Database Engineering*

Coding (generally simple; rule-based or transformational)



BOOK
ISBN
Title
DatePublished
id: ISBN

AUTHOR
AuthorName
id: AuthorName

COPY
ISBN
CopyNbr
DatePurchased
id: ISBN
CopyNbr
ref: ISBN

WRITE
AuthorName
ISBN
id: ISBN
AuthorName
ref: ISBN
ref: AuthorName

No more than 5 WRITE rows per BOOK row.

⇒

BOOK
ISBN
Title
DatePublished

AUTHOR
AuthorName

COPY
ISBN
CopyNbr
DatePurchased

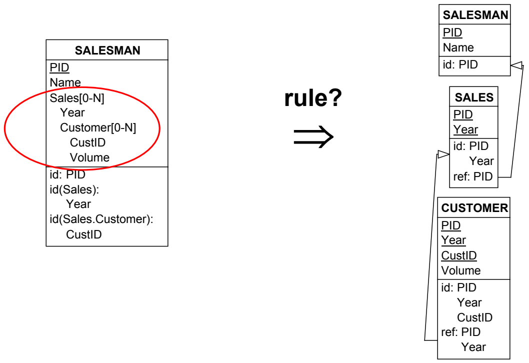
WRITE
AuthorName
ISBN

No more than 5 WRITE rows per BOOK row.

Introduction

Transformation-based view of *Database Engineering*

*What if the attribute is **multivalued**, **compound** and comprises **other multivalued components** ?*



SALESMAN
PID
Name
Sales[0-N]
Year
Customer[0-N]
CustID
Volume
id: PID
id(Sales):
Year
id(Sales.Customer):
CustID

rule?
⇒

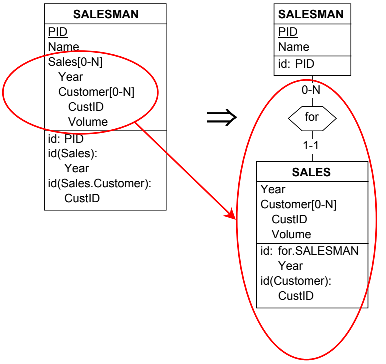
SALESMAN
PID
Name
id: PID

SALES
PID
Year
id: PID
Year
ref: PID

CUSTOMER
PID
Year
CustID
Volume
id: PID
Year
CustID
ref: PID
Year

Introduction

Transformation-based view of *Database Engineering*



SALESMAN
PID
Name
Sales[0-N]
Year
Customer[0-N]
CustID
Volume
id: PID
id(Sales):
Year
id(Sales.Customer):
CustID

⇒

SALESMAN
PID
Name
id: PID

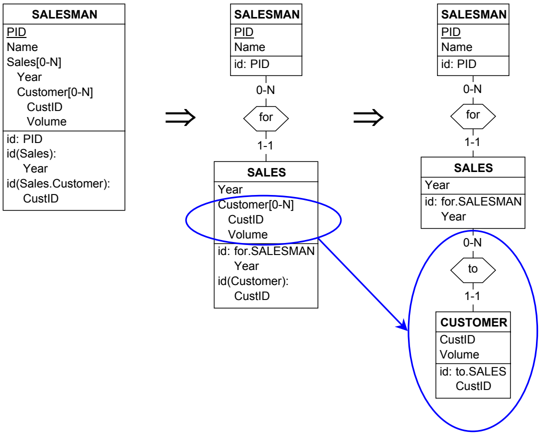
0-N
for
1-1

SALES
Year
Customer[0-N]
CustID
Volume
id: for.SALESMAN
Year
id(Customer):
CustID

***Note**: slightly different variant of the transformation of an attribute into an entity type*

Introduction

Transformation-based view of *Database Engineering*



SALESMAN
PID
Name
Sales[0-N]
Year
Customer[0-N]
CustID
Volume
id: PID
id(Sales):
Year
id(Sales.Customer):
CustID

⇒

SALESMAN
PID
Name
id: PID

0-N
for
1-1

SALES
Year
Customer[0-N]
CustID
Volume
id: for.SALESMAN
Year
id(Customer):
CustID

⇒

SALESMAN
PID
Name
id: PID

0-N
for
1-1

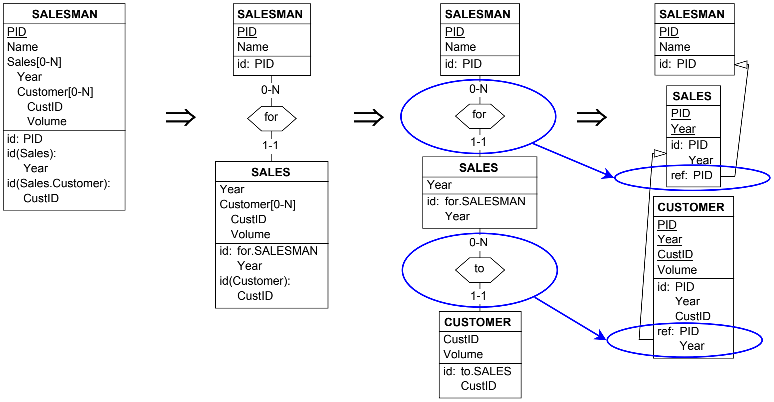
SALES
Year
id: for.SALESMAN
Year

0-N
to
1-1

CUSTOMER
CustID
Volume
id: to.SALES
CustID

Introduction

Transformation-based view of *Database Engineering*



SALESMAN
PID
Name
Sales[0-N]
Year
Customer[0-N]
CustID
Volume
id: PID
id(Sales):
Year
id(Sales.Customer):
CustID

⇒

SALESMAN
PID
Name
id: PID
0-N
for
1-1
SALES
Year
Customer[0-N]
CustID
Volume
id: for.SALESMAN
Year
id(Customer):
CustID

⇒

SALESMAN
PID
Name
id: PID
0-N
for
1-1
SALES
Year
id: for.SALESMAN
Year
0-N
to
1-1
CUSTOMER
CustID
Volume
id: to.SALES
CustID

⇒

SALESMAN
PID
Name
id: PID
SALES
PID
Year
id: PID
Year
ref: PID
CUSTOMER
PID
Year
CustID
Volume
id: PID
Year
CustID
ref: PID
Year

Introduction

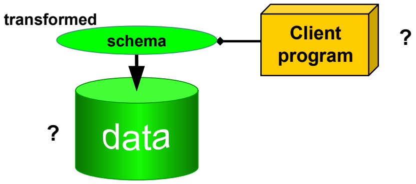
Transformation-based view of *Database Engineering*

Observations

- no new operators
- iterative application of known operators
- compositional property of transformations (the composition of two transformations still is a transformation)
- no combinatorial explosion, just the right (small) set of operators
- need for meta-rules for applying the operators (a *transformation plan*)

Introduction

Beyond data structure transformation



If the schema under transformation is that of an existing, in-use, database, then we also have to convert:

- **the data**
- **the client programs**

accordingly.

Introduction

What now?

LIBD

Introduction

Questions

- We need to represent schemas in a great variety of models (GER: a generic ER model)
- What is a transformation and how to specify it?
- Does a transformation preserve the information contents of a schema?
- Let us be more concrete: what about PRACTICAL transformations?
- How do transformations help in REAL database engineering processes?
  - what about Database design?
  - and Database reverse engineering?
- Can transformations and CASE tools coexist?

LIBD

Introduction

- and now, for the *DB geeks*:
  - is the GER just another nice way to draw schemas? (Semantics of the GER)?
  - Can one prove that a transformation always preserves the information contents of the source schema?

Modelling data structures

Modelling data structures

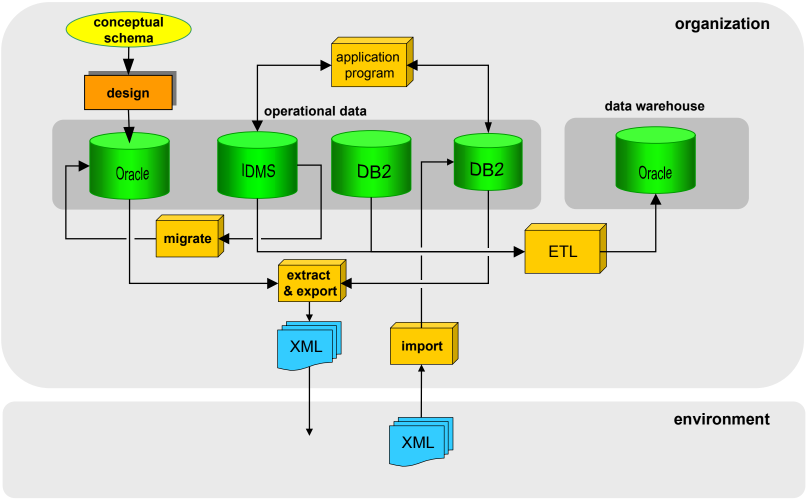
Dealing with multiple models

A typical organization uses N different data models. E.g., it

- **commonly uses DB2 databases,**
- **also uses a legacy IDMS database,**
- **writes its conceptual schemas in the ER model,**
- **quite often transfers data between databases,**
- **exchanges data with its environment,**
- **standardizes on XML format,**
- **plans to migrate some databases to other platforms,**
- **prepares the development of a datawarehouse,**
- **study the feasibility to merge several departments (and their information systems),**
- **etc.**

Modelling data structures

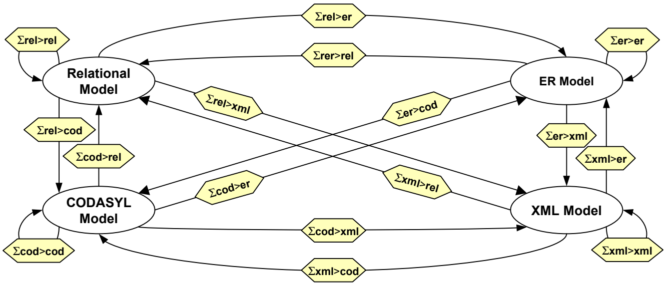
Dealing with multiple models



Modelling data structures

Dealing with multiple models

Considering all the inter-model and intra-model conversions, the organization requires *N x N different mappings (= 16)*.



LIBD

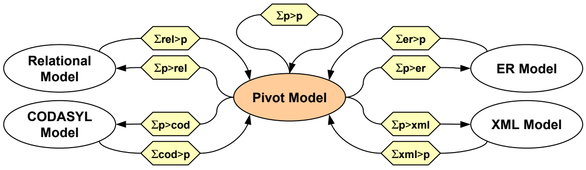
Modelling data structures

Dealing with multiple models

The usual answer: *introducing a pivot model*.

Considering all the inter-model and intra-model conversions,

the organization requires *2 x N + 1 different mappings (= 9)*.


Σp>p

Σrel>p

Σp>rel

Relational Model

Σer>p

Σp>er

ER Model

Pivot Model

CODASYL Model

Σp>cod

Σcod>p

Σp>xml

XML Model

Σxml>p

LIBD

Modelling data structures

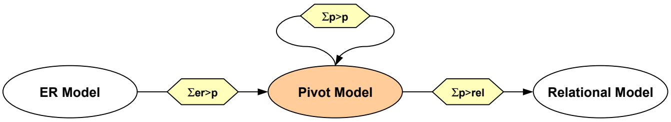
Dealing with multiple models

Example: *relational logical design*.


conceptual schema

Logical design

logical schema


Σp>p

ER Model

Σer>p

Pivot Model

Σp>rel

Relational Model

Modelling data structures

GER: the Generic Entity-Relationship model

A large spectrum data structure model

- **Encompasses several paradigms: ER, UML, SQL, CODASYL, IMS, file structures, XML, etc.**
- **Encompasses several levels of abstraction: conceptual, logical, physical, external**

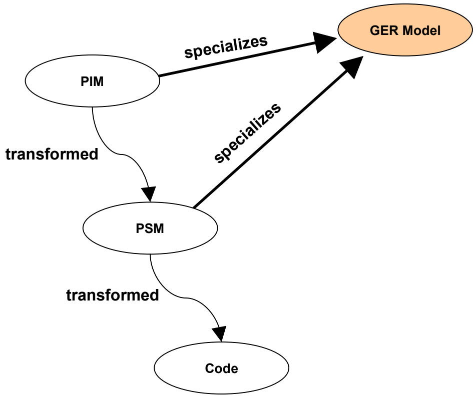
Chosen as the pivot model in this tutorial

Pivot Model := **GER Model**

Modelling data structures

GER: the Generic Entity-Relationship model

position in the MDA

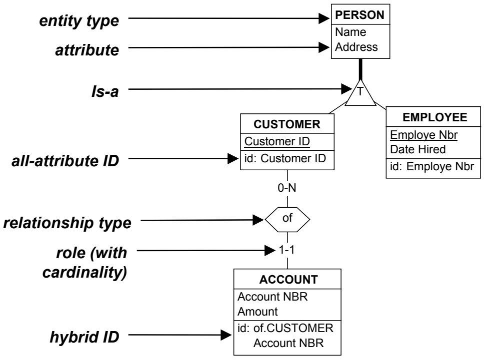


LIBD

Modelling data structures

GER: the Generic Entity-Relationship model

Conceptual schema fragment (1)



entity type

attribute

Is-a

all-attribute ID

relationship type

role (with cardinality)

hybrid ID

PERSON
Name
Address

T

CUSTOMER
Customer ID
id: Customer ID

EMPLOYEE
Employe Nbr
Date Hired
id: Employe Nbr

0-N

of

1-1

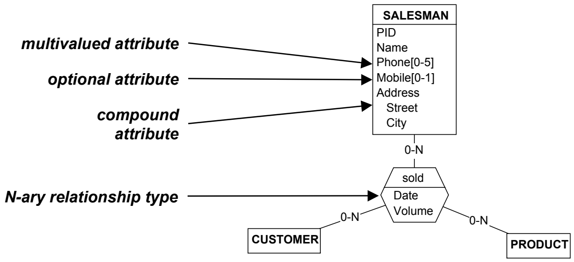
ACCOUNT
Account NBR
Amount
id: of.CUSTOMER
Account NBR

LIBD

Modelling data structures

GER: the Generic Entity-Relationship model

Conceptual schema fragment (2)



multivalued attribute

optional attribute

compound attribute

N-ary relationship type

SALESMAN
PID
Name
Phone[0-5]
Mobile[0-1]
Address
Street
City

0-N

sold
Date
Volume

0-N

0-N

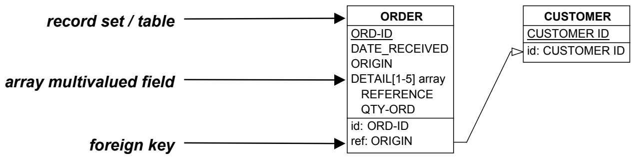
CUSTOMER

PRODUCT

Modelling data structures

GER: the Generic Entity-Relationship model

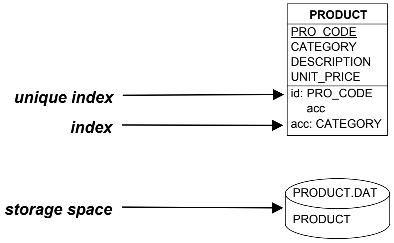
Logical schema fragment



Modelling data structures

GER: the Generic Entity-Relationship model

Physical schema fragment: RDB



Modelling data structures

Specifying operational models in the GER

Operational = in practical use in the organization

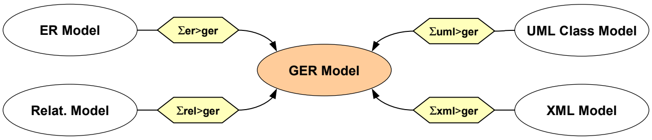
- **DB2, ER, UML diagrams, IDMS, IMS, standard file structures, XML Schema are usual operational models.**
- **The GER *is not an operational model* (yet)**

Modelling data structures

Specifying operational model M in the GER

Procedure

- **identifying the concepts of the GER that are pertinent in M**
- **specifying the structural constraints that hold in valid M schemas**
- **renaming the selected constructs according to the taxonomy of M.**



LIBD

Modelling data structures

Specifying operational model M in the GER

Application to the relational model (SQL2)

relational constructs	GER constructs	assembly rules
database schema	schema	
table	entity type	an entity type includes at least one attribute
domain	simple domain	
nullable column	single-valued and atomic attribute with cardinality [0-1]	
not null column	single-valued and atomic attribute with cardinality [1-1]	
primary key	primary identifier	a primary identifier comprises attributes with cardinality [1-1]
unique constraint	secondary identifier	
foreign key	reference group	the composition of the reference group must be the same as that of the target identifier
SQL names	GER names	the GER names must follow the SQL syntax

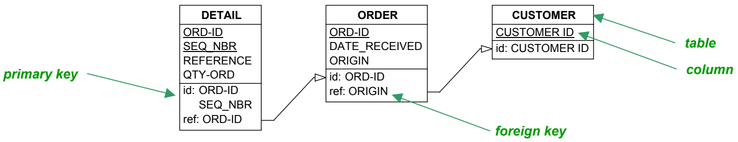
LIBD

Modelling data structures

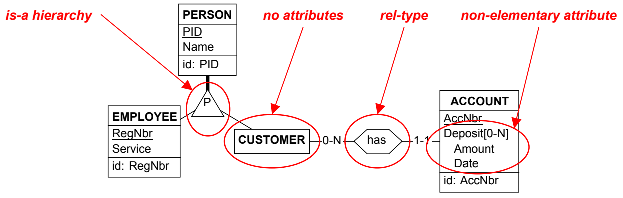
Specifying operational model M in the GER

Notion of M-compliant schema

This schema is SQL2-compliant:



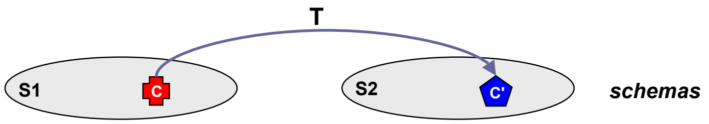
This schema is not SQL2-compliant:



Schema transformations

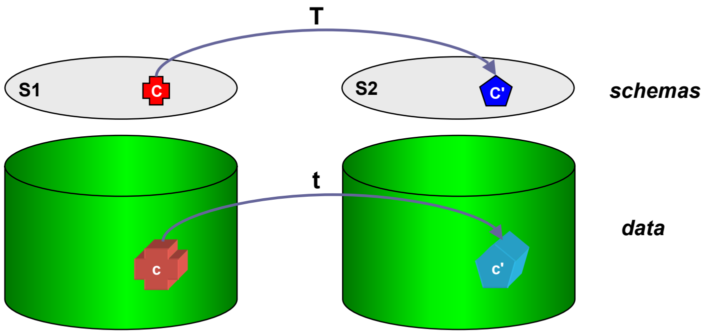
Schema transformations

A transformation T replaces a construct C in a schema S1 with another construct C', leading to schema S2



Schema transformations

If the schema describes actual data, the transformation should also tell how to convert the data (t) ...



Schema transformations

A transformation Σ is defined by two mappings T and t

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

$$\begin{tikzcd}
C \arrow[r, "T"] & C' = T(C) \\
c \arrow[u, "inst\_of"'] \arrow[r, "t"] & c' = t(c) \arrow[u, "inst\_of"']
\end{tikzcd}$$

T: *structural mapping = syntax of Σ*

t: *instance mapping = semantics of Σ*

Schema transformations

Mapping T can be specified with two predicates:

P: minimal pre-condition

Q: maximal post-condition

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

Schema transformations

Expressing structural predicates

through any logic-based language

relational (more concise, a name denotes an object)

entity-type(E)	*there exists an entity type with name E*

object-based (more general, a name is a property of an object)

entity-type(e)	*e is an entity type*
name(e,E)	*the name of e is E*

must allow <u>specification</u> AND reasoning (e.g., DL)

Schema transformations

Expressing structural predicates

intuitive example

entity-type(E)	*there exists an entity type with name E*
attribute(O,A,m,M,T)	*object (with name) O has an attribute with name A, cardinality m-M and type T*
id(O,Cp)	*object (with name) O has an identifier comprising components Cp*
rel-type(R)	*there exists a rel-type with name R*
role(R,r,E,m,M)	*rel-type R has a role with name r, played by E, with cardinality m-M*

Schema transformations

Specifying an entity type:

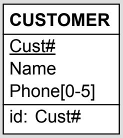
entity-type(CUSTOMER)
∧ attribute(CUSTOMER,Cust#,1,1,integer)
∧ attribute(CUSTOMER,Name,1,1,string)
∧ attribute(CUSTOMER,Phone,0,5,string)
∧ id(CUSTOMER,{Cust#})

Schema transformations

Practically, a structural predicate can be defined graphically:

entity-type(CUSTOMER)
∧ attribute(CUSTOMER,Cust#,1,1,integer)
∧ attribute(CUSTOMER,Name,1,1,string)
∧ attribute(CUSTOMER,Phone,0,5,string)
∧ id(CUSTOMER,{Cust#})

=



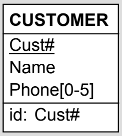
Schema transformations

The structural mapping of a transformation can be defined graphically:

P

P = entity-type(CUSTOMER)
∧ attribute(CUSTOMER,Cust#,1,1,integer)
∧ attribute(CUSTOMER,Name,1,1,string)
∧ attribute(CUSTOMER,Phone,0,5,string)
∧ id(CUSTOMER,{Cust#})

||

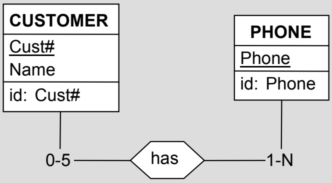


⇒

Q

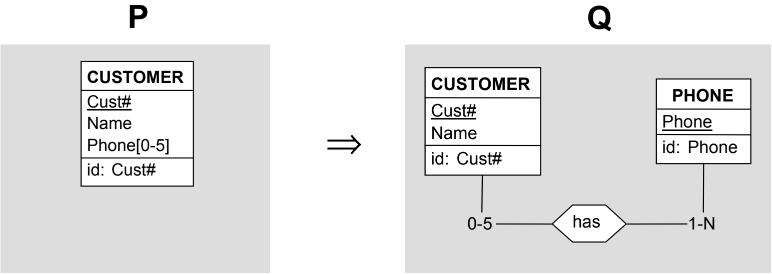
Q = entity-type(CUSTOMER)
∧ attribute(CUSTOMER,Name,1,1,string)
∧ attribute(CUSTOMER,Phone,0,5,string)
∧ id(CUSTOMER,{Cust#})
∧ entity-type(PHONE)
∧ attribute(PHONE,Phone,1,1,string)
∧ id(PHONE,{Phone})
∧ rel-type(has)
∧ role(has,,CUSTOMER,0,5)
∧ role(has,,PHONE,1,N)

||



Schema transformations

From now on:

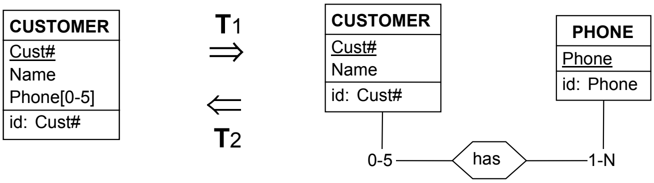


Schema transformations

Inverse transformation

$$\Sigma 2 = \Sigma 1^{-1} \quad \textit{iff}$$

$$\forall C: P1(C) \Rightarrow C = T2(T1(C))$$



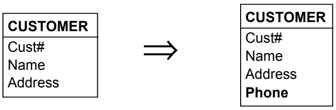
- Intuitively, $\Sigma 2$ *undoes* the effect of $\Sigma 1$ at the structural level
- $\Sigma 1$ not necessarily the inverse of $\Sigma 2$

Semantics preservation properties of transformations

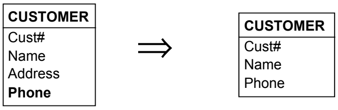
Semantics preservation properties of transformations

A transformation can ...

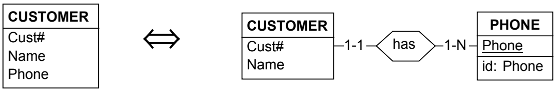
- **augment** the information contents of the schema



- **decrease** the information contents of the schema



- **preserve** the information contents of the schema



- *more complex patterns exist*

Semantics preservation properties of transformations

A transformation can be ...

- **not reversible**: not semantics-preserving
- **reversible**: "half" semantics-preserving
- **symmetrically reversible**: fully semantics-preserving

Semantics preservation properties of transformations

Examples

P: R(A,B,C); Q: R1(A,B); R2(A,C);	***not reversible***
P: R(A,B,C); **A →→ B|C** Q: R1(A,B); R2(A,C);	***reversible (Fagin's theorem)***
P: R(A,B,C); **A →→ B|C** Q: R1(A,B); R2(A,C); **R1[A] = R2[C];**	***symmetrically reversible***

Semantics preservation properties of transformations

Reversible transformation

A transformation is reversible if
there is **an inverse mapping for instances** as well

$\Sigma 1$ **is reversible** ***iff*** $\exists\ \Sigma 2 = \Sigma 1^{-1}$:

$$\forall\ C: P(C) \Rightarrow C = T2(T1(C))$$

$$\wedge$$

$$\forall\ c \in inst(C): c = t2(t1(c))$$

Semantics preservation properties of transformations

Symmetrically reversible transformation

Σ is symmetrically reversible *iff* **both** Σ **and** Σ^{-1} **are reversible**

$$\Sigma = \langle P,Q,t \rangle \Rightarrow \Sigma^{-1} = \langle Q,P,t' \rangle$$

- **SR-transformations are first class operators**
- **They preserve the information contents of the source schema**
- **SR-transformation are *semantics-preserving***

Semantics preservation properties of transformations

Big question

How can we prove that a transformation

is semantics-preserving, i.e., that it is SR

Answer in the proceedings

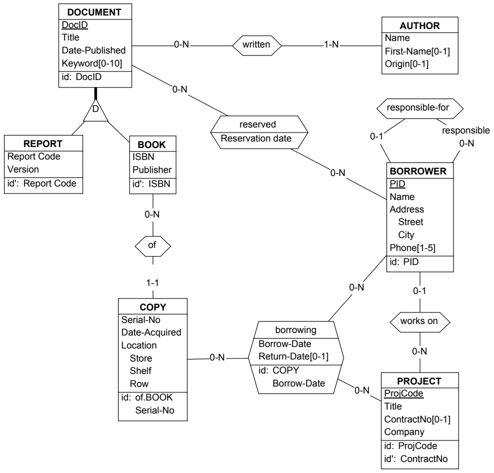
Typology of practical transformations

Elementary transformations

LIBD

Typology of practical transformations

The working example



LIBD

Typology of practical transformations

The main classes of elementary SR-transformations

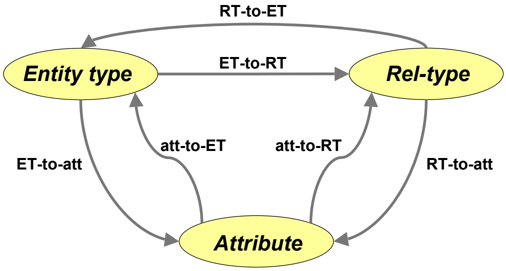
- **mutation transformations**
- **ISA transformations**
- **other elementary transformations**

Typology of practical transformations

Mutation transformations

A mutation changes the *gender* of an object while preserving its information contents

3 *genders* ⇒ 6 mutations



RT-to-ET

Entity type

ET-to-RT

Rel-type

att-to-ET

att-to-RT

ET-to-att

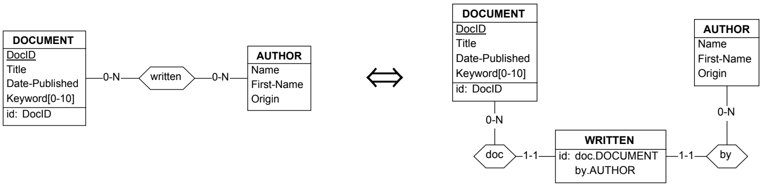
RT-to-att

Attribute

Typology of practical transformations

Mutation transformations (SR)

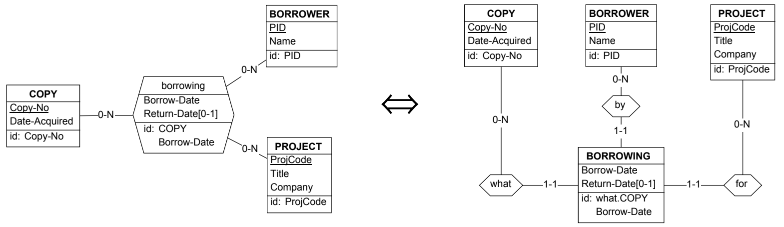
Entity types and Rel-types (1)



Typology of practical transformations

Mutation transformations (SR)

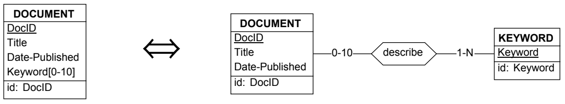
Entity types and Rel-types (2)

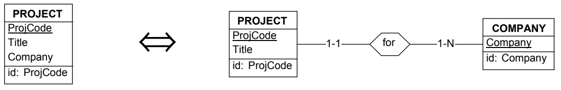


Typology of practical transformations

Mutation transformations (SR)

Entity types and Attributes

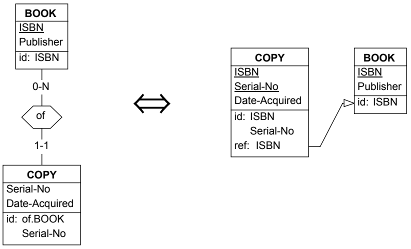




Typology of practical transformations

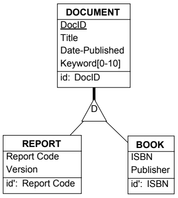
Mutation transformations (SR)

Rel-types and Attributes

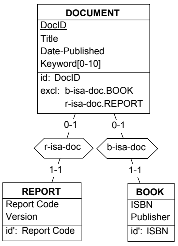


Typology of practical transformations

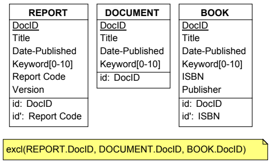
ISA transformations (SR)



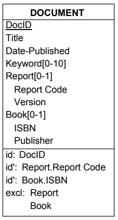
Materialization



Downward inheritance



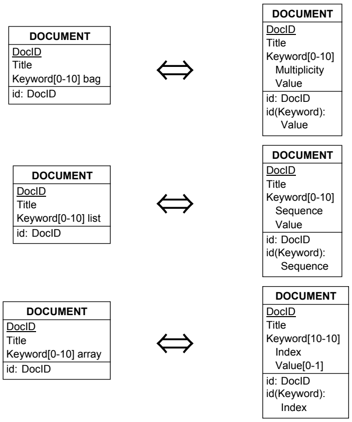
Upward inheritance



Typology of practical transformations

Other elementary transformations

Non-set attributes (SR)



DOCUMENT
DocID
Title
Keyword[0-10] bag
id: DocID

⇔

DOCUMENT
DocID
Title
Keyword[0-10]
  Multiplicity
  Value
id: DocID
id(Keyword):
  Value

DOCUMENT
DocID
Title
Keyword[0-10] list
id: DocID

⇔

DOCUMENT
DocID
Title
Keyword[0-10]
  Sequence
  Value
id: DocID
id(Keyword):
  Sequence

DOCUMENT
DocID
Title
Keyword[0-10] array
id: DocID

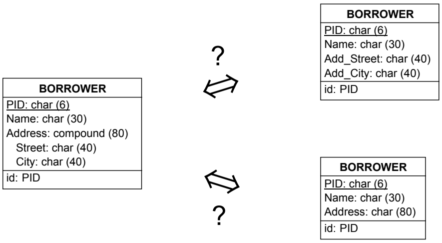
⇔

DOCUMENT
DocID
Title
Keyword[10-10]
  Index
  Value[0-1]
id: DocID
id(Keyword):
  Index

Typology of practical transformations

Other elementary transformations

Compound attribute: disagregation and concatenation



BORROWER
PID: char (6)
Name: char (30)
Address: compound (80)
  Street: char (40)
  City: char (40)
id: PID

?
⇔

BORROWER
PID: char (6)
Name: char (30)
Add_Street: char (40)
Add_City: char (40)
id: PID

⇔
?

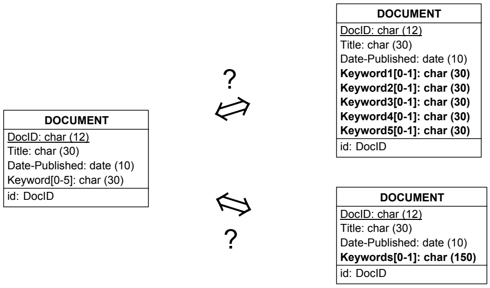
BORROWER
PID: char (6)
Name: char (30)
Address: char (80)
id: PID

Very common but not SR

Typology of practical transformations

Other elementary transformations

Multivalued attribute: instanciation and concatenation



DOCUMENT
DocID: char (12)
Title: char (30)
Date-Published: date (10)
Keyword[0-5]: char (30)
id: DocID

? ⇔

DOCUMENT
DocID: char (12)
Title: char (30)
Date-Published: date (10)
Keyword1[0-1]: char (30)
Keyword2[0-1]: char (30)
Keyword3[0-1]: char (30)
Keyword4[0-1]: char (30)
Keyword5[0-1]: char (30)
id: DocID

⇔ ?

DOCUMENT
DocID: char (12)
Title: char (30)
Date-Published: date (10)
Keywords[0-1]: char (150)
id: DocID

Very common but not SR

Typology of practical transformations

Complex transformations

Typology of practical transformations

Elementary transformations are just
building blocks for more complex operators

Challenge

Developing higher-level transformations
with elementary SR-transformations
in such a way that the SR property is preserved

Typology of practical transformations

The main classes of complex SR-transformations

- **compound transformations**
- **predicate-driven transformations**
- **model-driven transformations**

Typology of practical transformations

Compound transformations

The composition of two transformations is a transformation

The composition of two SR-transformations is an SR-transformation

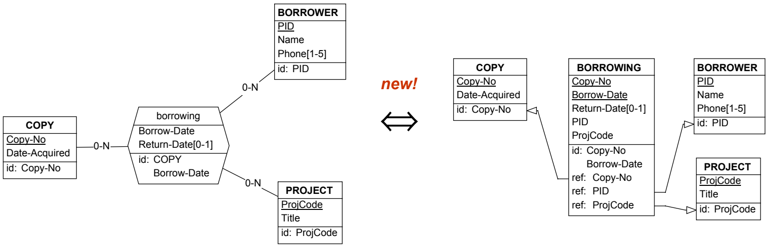
$$\Sigma 1 = \langle T1, t1 \rangle$$

$$\Sigma 2 = \langle T2, t2 \rangle$$

$$\Sigma 12 = \Sigma 2 \circ \Sigma 1 = \langle T2 \circ T1, t2 \circ t1 \rangle$$

Typology of practical transformations

Compound transformations



BORROWER
PID
Name
Phone[1-5]
id: PID

COPY
Copy-No
Date-Acquired
id: Copy-No

borrowing
Borrow-Date
Return-Date[0-1]
id: COPY
Borrow-Date

PROJECT
ProjCode
Title
id: ProjCode

0-N
0-N
0-N

new!

⇔

COPY
Copy-No
Date-Acquired
id: Copy-No

BORROWING
Copy-No
Borrow-Date
Return-Date[0-1]
PID
ProjCode
id: Copy-No
Borrow-Date
ref: Copy-No
ref: PID
ref: ProjCode

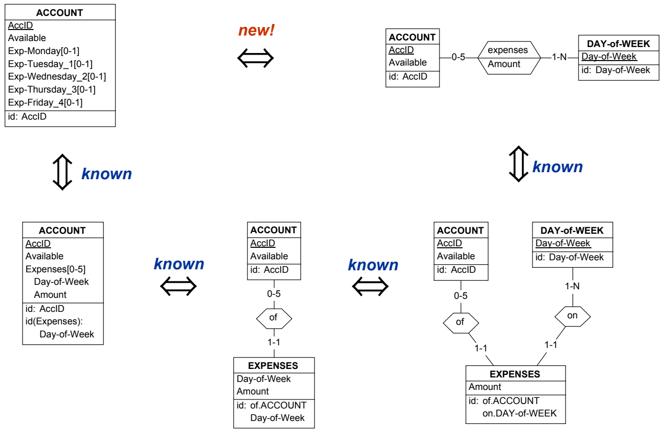
BORROWER
PID
Name
Phone[1-5]
id: PID

PROJECT
ProjCode
Title
id: ProjCode

intuitively: Σ = RT-to-ET ∘ RT-to-att

Typology of practical transformations

Compound transformations



ACCOUNT
AccID
Available
Exp-Monday[0-1]
Exp-Tuesday_1[0-1]
Exp-Wednesday_2[0-1]
Exp-Thursday_3[0-1]
Exp-Friday_4[0-1]
id: AccID

new!
⇔

ACCOUNT
AccID
Available
id: AccID
0-5
expenses
Amount
1-N
DAY-of-WEEK
Day-of-Week
id: Day-of-Week

⇕ known

⇕ known

ACCOUNT
AccID
Available
Expenses[0-5]
Day-of-Week
Amount
id: AccID
id(Expenses):
Day-of-Week

known
⇔

ACCOUNT
AccID
Available
id: AccID
0-5
of
1-1
EXPENSES
Day-of-Week
Amount
id: of.ACCOUNT
Day-of-Week

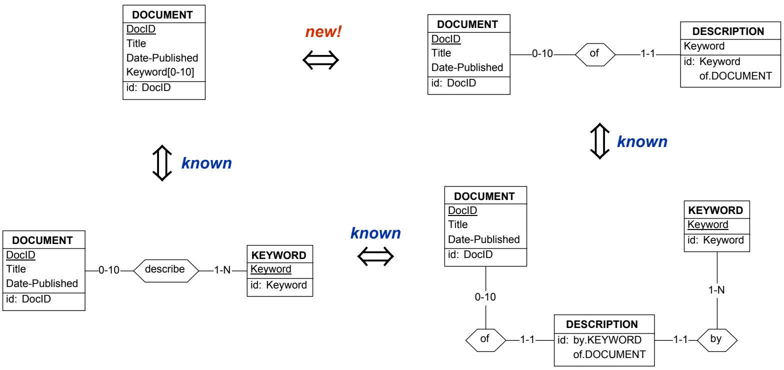
known
⇔

ACCOUNT
AccID
Available
id: AccID
0-5
of
1-1
DAY-of-WEEK
Day-of-Week
id: Day-of-Week
1-N
on
1-1
EXPENSES
Amount
id: of.ACCOUNT
on.DAY-of-WEEK

dom(Day-of-Week) = {'Monday','Tuesday', .. ,'Friday'}

Typology of practical transformations

Compound transformations



DOCUMENT
DocID
Title
Date-Published
Keyword[0-10]
id: DocID

new!
⇔

DOCUMENT
DocID
Title
Date-Published
id: DocID
0-10
of
1-1
DESCRIPTION
Keyword
id: Keyword
of.DOCUMENT

⇕ known

⇕ known

DOCUMENT
DocID
Title
Date-Published
id: DocID
0-10
describe
1-N
KEYWORD
Keyword
id: Keyword

known
⇔

DOCUMENT
DocID
Title
Date-Published
id: DocID
0-10
of
1-1
DESCRIPTION
id: by.KEYWORD
of.DOCUMENT
1-1
by
1-N
KEYWORD
Keyword
id: Keyword

a very popular mutation transformation

LIBD

Typology of practical transformations

Predicate-driven (conditional) transformations

Transformations that apply on a <u>set of qualified objects</u> in the current schema

Σ (p)

where Σ is a transformation

p is a structural predicate

interpretation: apply Σ to all the objects that satisfy **p**

ambiguous: if $p(o1) \wedge p(o2) \wedge$ **(**apply $\Sigma(o1) \Rightarrow \neg p(o2)$ **)**
then should o2 be processed anyway?

usual strategy (*snapshot*): first compute the set of objects that satisfy p,
then apply Σ to each of its elements that *survive*.

LIBD

Typology of practical transformations

Predicate-driven transformations

We need a **language for p**

- structural (e.g., DL): complex and leading to huge expressions
- ad hoc for the GER: expressive, concise, parametric,
  but not generic, not closed

ROLE_per_RT(I J): *the number of roles of the current rel-type is between I and J*

ONE_ROLE_per_RT(1 2): *the number of "one" roles (with cardinality ?-1)is between I and J*

MAX_CARD_of_ATT(I J): *the maximum cardinality of the current attribute is between I and J*

DEPTH_of_ATT(I J): *the level of the current attribute is between I and J*

Typology of practical transformations

Predicate-driven transformations

Σ (p)

RT_into_ET(ROLE_per_RT(3 N)):

transform each rel-type into an entity type
*(**if** tey are at least 3 roles)*

RT_into_REF(ROLE_per_RT(2 2) and ONE_ROLE_per_RT(1 2)):

transform each rel-type into referential attributes
*(**if** they are binary and one-to-many or one-to-one)*

INSTANTIATE(MAX_CARD_of_ATT(2 4)):

instanciate each attribute
*(**if** they are "slightly" multivalued: from 2 to 4values)*

ATT_into_ET_VAL(DEPTH_of_ATT(1 1) and MAX_CARD_of_ATT(5 N)):

transform each attribute into an entity type
*(**if** they are at the top level and they are "strongly" multivalued: at least 5 values)*

Typology of practical transformations

Model-driven transformation

Goal: considering schema S1 in model M1, transform S1 into S2 that complies with model M2. Of course, as far as possible through SR-transformations!

Example: *considering the Entity-relationship schema S1, transform S1 into S2 that complies with the relational model. Of course, as far as possible without information loss!*

Structure: a compound transformation comprising predicate-driven transformations.

Practical form: a transformation plan.

Typology of practical transformations

Model-driven transformation

Principle:

Identify the constructs of M1 that violate M2

For each such construct C, choose a transformation <T,t> = <P,Q,t> such that

P(C)

T(C) satisfies M2

Things may be a bit more complex, requiring a compound transformation.

Example, processing N-ary rel-types for relational compliance requires two successive transformations

---

Typology of practical transformations

Model-driven transformation

Example: ER to Binary (*flat Bachman*) conversion

The binary model is a variant of the ER model in which:

- there is no ISA relations
- the rel-types are functional (binary, one-to-many or one-to-one)
- the rel-types have no attributes
- each rel-type is defined on two distinct entity types (no cyclic rel-types)
- the attributes are single-valued and atomic.

Typology of practical transformations

Model-driven transformation

Flat Bachman schemas - invalid constructs:

- ISA relations
- cyclic rel-types
- complex rel-types (with attributes, N-ary)
- many-to-many binary rel-types
- multivalued attributes
- compound attributes.

Typology of practical transformations

Model-driven transformation

Flat Bachman schemas - processing invalid constructs:

- ISA relations: *materialization*
- cyclic rel-types: *transform into entity types*
- complex rel-types (with attributes, N-ary): *transform into entity types*
- many-to-many binary rel-types: *transform into entity types*
- multivalued attributes: *transform into entity types*
- compound attributes: *disagregate*.

Typology of practical transformations

Model-driven transformation

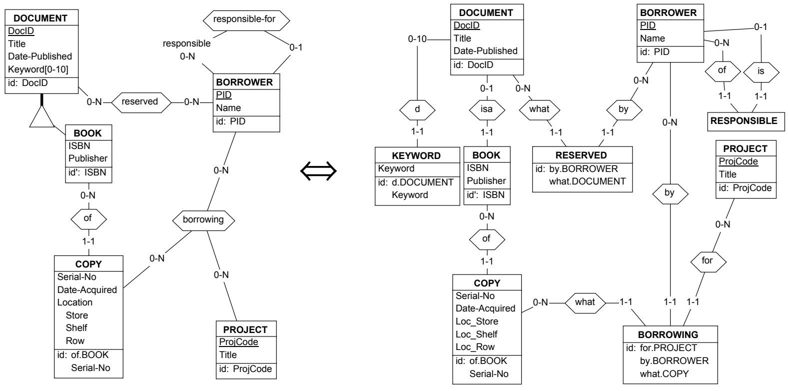
Transformation plan for ER to *Flat Bachman* conversion

```
ISA_into_RT;                                          transform ISA relations by materialization;
RT_into_ET(RECURSIVITY_in_RT(2 N));                   transform rel-types in which the same entity type
                                                      appears more than once;
RT_into_ET(ATT_per_RT(1 N) or ROLE_per_RT(3 N));      transform complex rel-types;
RT_into_ET(ONE_ROLE_per_RT(0 0));                     transform rel-types in which there is no "one" role;
LOOP;                                                 iteratively flatten the attribute structure
    ATT_into_ET_INST(MAX_CARD_of_ATT(2 N))
    DISAGGREGATE
ENDLOOP
```

Typology of practical transformations

Model-driven transformation

Example of ER to *Flat Bachman* conversion



Typology of practical transformations

Model-driven transformation

Other popular examples

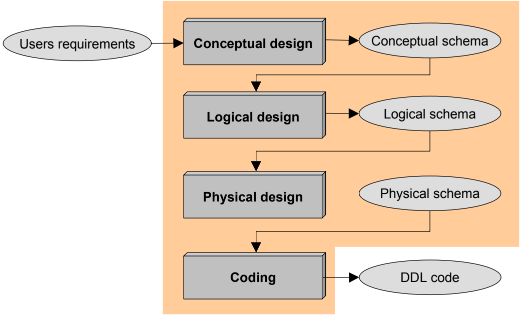
- **ER to UML**
- **UML to ER**
- **ER to relational**
- **relational to ER**
- **COBOL files to ER**
- **ER to XML**
- **relational to XML**

Transformational modelling of database engineering processes

Transformational modeling of database engineering processes

Most database engineering processes are high-level transformations

Example 1: database design



DDL code = DB-design(Users Requirements)

DB-design = Coding ∘ PhysD ∘ LogD ∘ ConcD

Transformational modeling of database engineering processes

Logical design

Logical_schema = Logical_design(Conceptual_schema)

Logical_design clearly is a **model-driven** transformation

Let us develop its **transformation plan**

Transformational modeling of database engineering processes

What are the invalid GER constructs in the relational model?

- ISA relations
- relationship types
    - complex
    - functional
- multivalued attributes
- compound attributes
- names not compliant with the SQL syntax

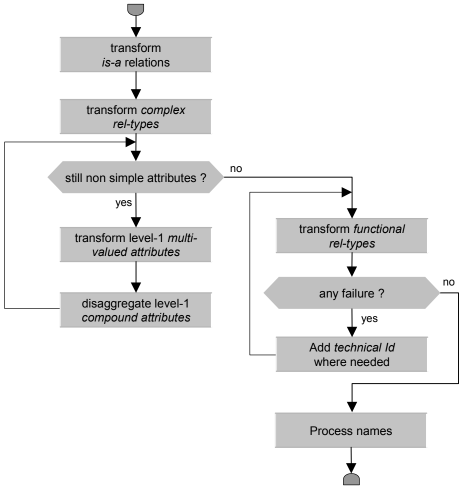
Transformational modeling of database engineering processes

How to get rid of them?

- ISA relations: transform by materialization; then *go to* functional rel-type
- relationship types
    - complex: transform into entity types, then *go to* functional rel-type
    - functional: transform into foreign keys
- multivalued attributes: transform into entity types, *goto* functional rel-type
- compound attributes: disagregate
- names not compliant with the SQL syntax: change

Transformational modeling of database engineering processes

A transformation plan



transform *is-a* relations

transform *complex rel-types*

still non simple attributes ?

yes

transform level-1 *multi-valued attributes*

disaggregate level-1 *compound attributes*

no

transform *functional rel-types*

any failure ?

yes

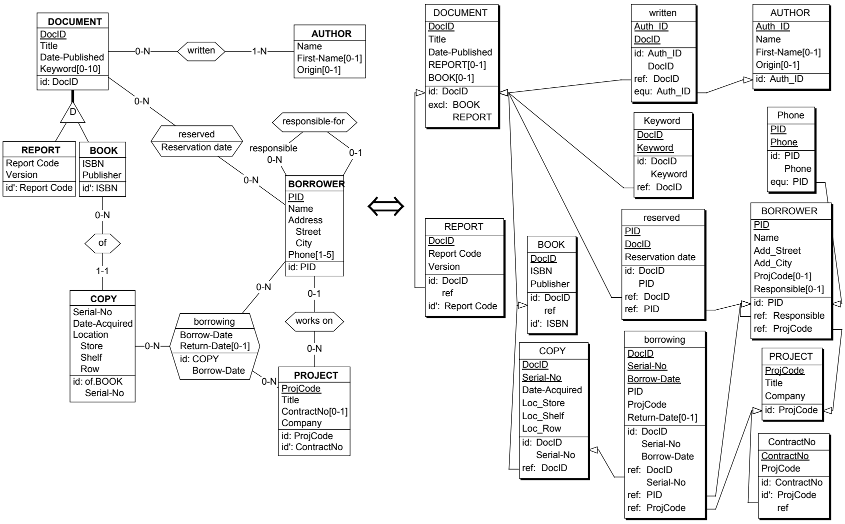
Add *technical Id* where needed

no

Process names

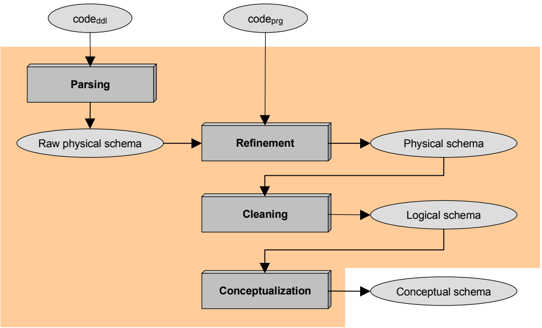
Transformational modeling of database engineering processes

Application



Transformational modeling of database engineering processes

Example 2: database reverse engineering



Conceptual schema = DB-REng(code$_{ddl}$, code$_{prg}$)

DB-design = Concept ∘ Clean ∘ Refine ∘ Parse

Transformational modeling of database engineering processes

Example 2: database reverse engineering

Interesting observations

Refine ∘ Parse = Coding^{-1}

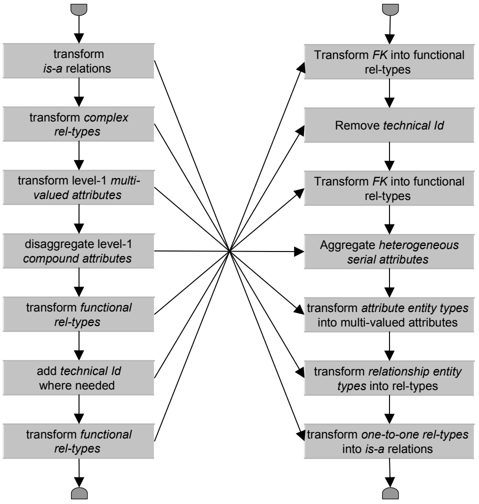
Cleaning = Physical_design^{-1}

Conceptualization = Logical_design^{-1}

Conclusion: **DB reverse engineering** is (grossly speaking) the inverse of **DB engineering**

Transformational modeling of database engineering processes

Hence the transformation plan of Conceptualization:



- transform *is-a* relations
- transform *complex rel-types*
- transform level-1 *multi-valued attributes*
- disaggregate level-1 *compound attributes*
- transform *functional rel-types*
- add *technical Id* where needed
- transform *functional rel-types*

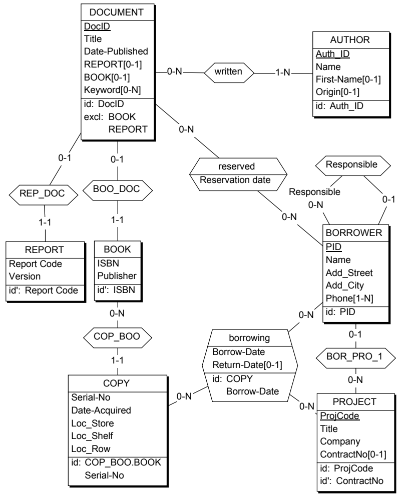
- Transform *FK* into functional rel-types
- Remove technical *Id*
- Transform *FK* into functional rel-types
- Aggregate *heterogeneous serial attributes*
- transform *attribute entity types* into multi-valued attributes
- transform *relationship entity types* into rel-types
- transform *one-to-one rel-types* into *is-a* relations

Transformational modeling of database engineering processes

Experiment:

relational logical schema

⇔



Convincing, but obviously needs some polishing!

Schema transformations in CASE tools

Schema transformations in CASE tools

CASE tools with DB design facilities offer explicit or implicit transformations for the production of the database code and for (limited) reverse engineering.

- **conceptual schema → DDL code**
- **conceptual schema → relational schema → DDL code**
- **DDL code → relational schema → conceptual schema**

Few CASE tools include transformations as a major engineering paradigm.

Example DB-MAIN

Schema transformations in CASE tools

The DB-MAIN CASE environment

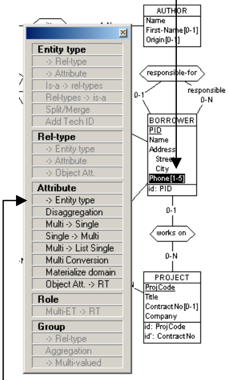
- Toolset of about 30 elementary transformations. Most are SR. The others trigger a warning.
- Predicate-driven transformations (through two transformation assistants)
  - ➢ simplified (for the *dummies*)
  - ➢ advanced (for the *smarties*)
- Model-driven transformations (through two transformation assistants)
  - ➢ scripting facilities for developing transformation plans
  - ➢ a dozen predefined, updatable, transformation plans

Schema transformations in CASE tools

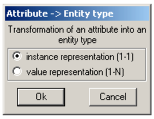
The DB-MAIN CASE environment

Elementary transformations **(Transforming attribute Phone into entity type PHONE)**

1. select an object



2. select a transformation

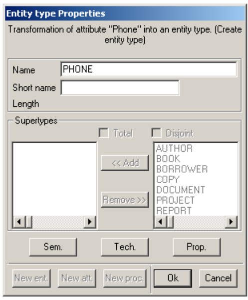


Attribute -> Entity type

Transformation of an attribute into an entity type

● instance representation (1-1)
○ value representation (1-N)

3. if needed, select the variant



Entity type Properties

Transformation of attribute "Phone" into an entity type. (Create entity type)

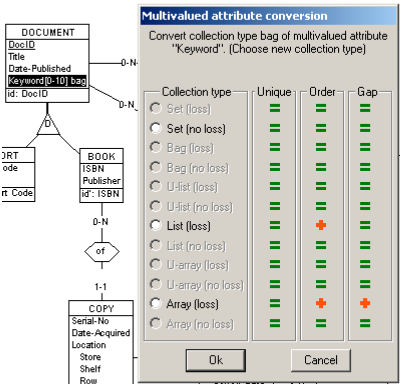
Name: PHONE
Short name:
Length

4. if needed, give target names

Schema transformations in CASE tools

The DB-MAIN CASE environment

Elementary transformations (Transforming bag attribute *Keyword* into a set attribute)



Multivalued attribute conversion

Convert collection type bag of multivalued attribute "Keyword". (Choose new collection type)

Collection type	Unique	Order	Gap
○ Set (loss)	=	=	=
○ Set (no loss)	=	=	=
○ Bag (loss)	=	=	=
○ Bag (no loss)	=	=	=
○ U-list (loss)	=	=	=
○ U-list (no loss)	=	=	=
○ List (loss)	=	+	=
○ List (no loss)	=	=	=
○ U-array (loss)	=	=	=
○ U-array (no loss)	=	=	=
○ Array (loss)	=	+	+
○ Array (no loss)	=	=	=

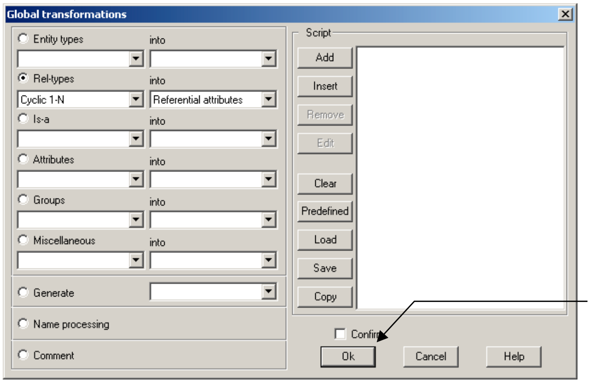
Schema transformations in CASE tools

The DB-MAIN CASE environment

Predicate-driven transformations: simplified assistant

1. choose a pattern

2. choose an action



○ Entity types into
● Rel-types: Cyclic 1-N into Referential attributes
○ Is-a into
○ Attributes into
○ Groups into
○ Miscellaneous into
○ Generate
○ Name processing
○ Comment

3. execute

LIBD

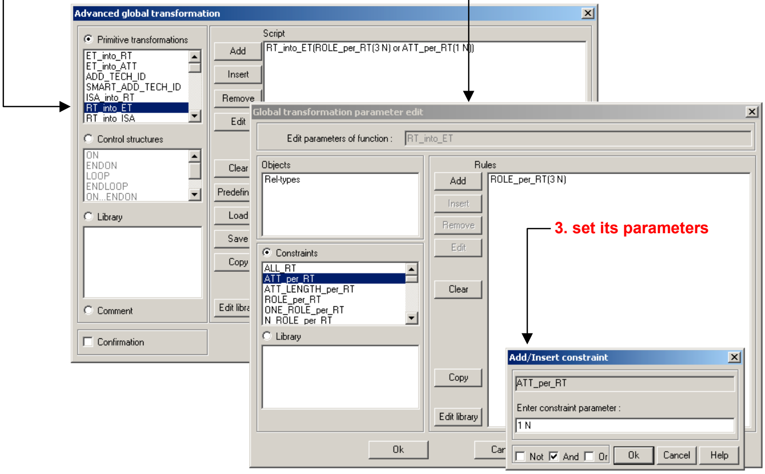
Schema transformations in CASE tools

The DB-MAIN CASE environment

Predicate-driven transformations: advanced assistant

1. select an operation

2. define the predicate



3. set its parameters

Advanced global transformation

● Primitive transformations: ET_into_RT, ET_into_ATT, ADD_TECH_ID, SMART_ADD_TECH_ID, ISA_into_RT, **RT_into_ET** (selected), RT_into_ISA

Script: RT_into_ET(ROLE_per_RT(3 N) or ATT_per_RT(1 N))

Global transformation parameter edit

Edit parameters of function : RT_into_ET

Objects: Rel-types

● Constraints: ALL_RT, **ATT_per_RT** (selected), ATT_LENGTH_per_RT, ROLE_per_RT, ONE_ROLE_per_RT, N_ROLE_per_RT

Rules: ROLE_per_RT(3 N)

Add/Insert constraint

ATT_per_RT

Enter constraint parameter : 1 N

☐ Not ☑ And ☐ Or

LIBD

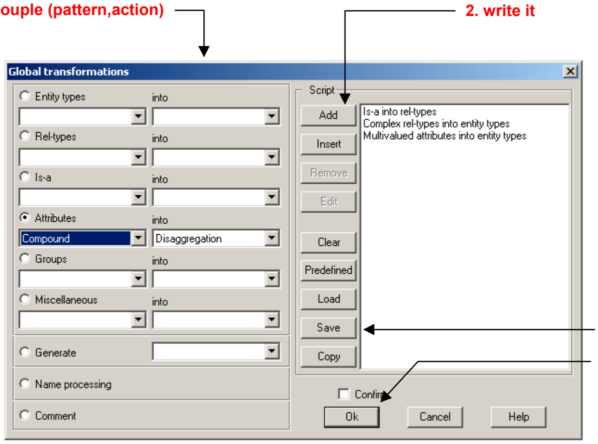
Schema transformations in CASE tools

The DB-MAIN CASE environment

Model-driven transformations: simplified assistant

1. build a couple (pattern,action)

2. write it



3. save script

4. execute script

Global transformations

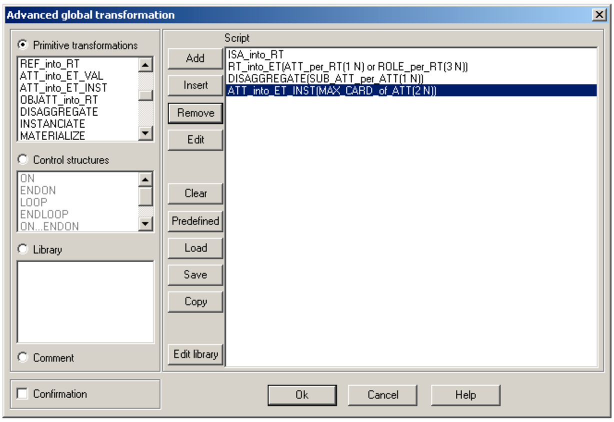
● Attributes: Compound into Disaggregation

Script:
Is-a into rel-types
Complex rel-types into entity types
Multivalued attributes into entity types

Schema transformations in CASE tools

The DB-MAIN CASE environment

Model-driven transformations: advanced assistant



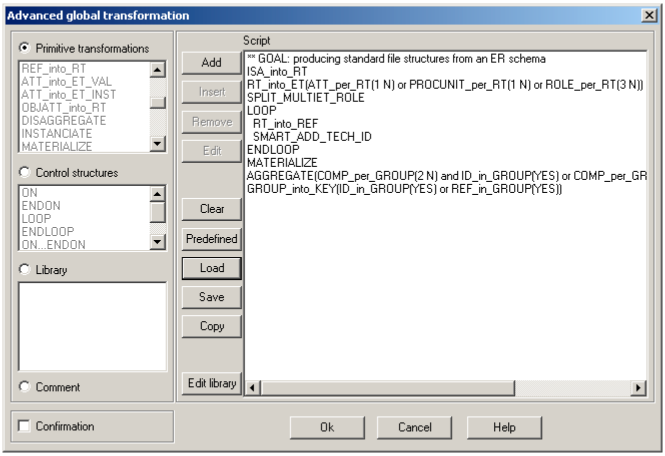
Script

```
ISA_into_RT
RT_into_ET(ATT_per_RT(1 N) or ROLE_per_RT(3 N))
DISAGGREGATE(SUB_ATT_per_ATT(1 N))
ATT_into_ET_INST(MAX_CARD_of_ATT(2 N))
```

Schema transformations in CASE tools

The DB-MAIN CASE environment

Model-driven transformations: advanced assistant



Script

```
** GOAL: producing standard file structures from an ER schema
ISA_into_RT
RT_into_ET(ATT_per_RT(1 N) or PROCUNIT_per_RT(1 N) or ROLE_per_RT(3 N))
SPLIT_MULTIET_ROLE
LOOP
  RT_into_REF
  SMART_ADD_TECH_ID
ENDLOOP
MATERIALIZE
AGGREGATE(COMP_per_GROUP(2 N) and ID_in_GROUP(YES) or COMP_per_GR
GROUP_into_KEY(ID_in_GROUP(YES) or REF_in_GROUP(YES))
```

Conclusions and perspectives

Conclusions and perspectives

- **Intuitively, most database engineering processes are transformational by nature.**
- **By combining elementary transformations, we can give these processes a precise transformational definition.**
- **A transformation can be formalized so that its preservation properties can be proved.**
- **We need a small set of elementary transformations (20 - 40).**
- **Once correctly defined, a transformation is quite reliable, and is guaranteed to preserve information whatever the context in which it is applied.**
- **Transformation are (sort of …) easy to implement in CASE tools.**

Conclusions and perspectives

However, some problems are not (completely) solved:

- **a transformation must address all the aspects of the data structures: documentation, annotations, statistics, operations (methods).**
- **complex problem: propagating the constraints; OK for uniqueness, but others more complex.**
- **how to efficiently transform the data, following schema transformation?**
- **transforming a high-level abstract schema is nice, but how do we propagate them to the lower-level schemas (including the code)?**
- **transforming the data structures is nice, but what about the programs? Notion of *co-transformation*. See Anthony's presentation.**

Conclusions and perspectives

More information at

http://info.fundp.ac.be/libd

***Education Edition* of DB-MAIN at the same address**

LIBD

Conclusions and perspectives

Exercise session

Expected time: **45 minutes**

Requirements: **Windows PC**

- ***Copy the folder « Exercise » from the CD-ROM to your desktop.***
- ***Eject the CD-ROM and give it to your neighbour.***
- ***Open « Exercise », then « Library »***
- ***Run db_main.exe***
- ***Open project « Library-2003 »***

LIBD

Appendix 1
Semantics of the GER

Semantics of the GER

Semantics of the GER

Problems:

- to reason rigorously, we need to associate a formal semantics with the GER,
- the GER is rich and complex
- therefore, its semantics is likely to be complex as well,
- complex formalisms tend to be unreliable.

General solution:

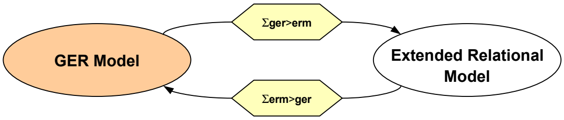
to build a (as far as possible) bijective mapping between the GER and a simpler formalism with a well defined semantics.

Semantics of the GER

Semantics of the GER

Specific solution:

to interpret the GER as a variant of the *non first-normal-form* (N1NF) relational model, the Extended Relational Model (ERM).



LIBD

Semantics of the GER

Semantics of the GER

See proceedings

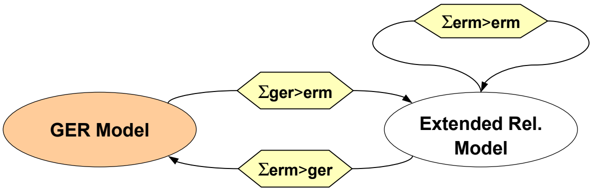
LIBD

Appendix 2

Proving the SR property of the GER transformations

Proving properties of GER transformations

Building a set of SR-transformations for the ERM is (reasonably) easy, thanks to the underlying N1NF relational theory

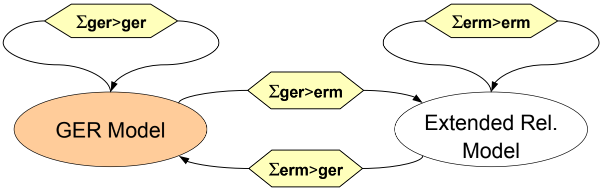


Proving properties of GER transformations

A GER transformation Σ_g is SR if it can be proved that,

$$\Sigma_g = \Sigma_{ger>erm} \circ \Sigma_e \circ \Sigma_{erm>ger}$$

where **Σ_e is a (possibly complex) ERM SR-transformation**



Proving properties of GER transformations

LIBD

The proceedings provides a proof of the SR-property

of the most useful variants of the six mutation transformations.

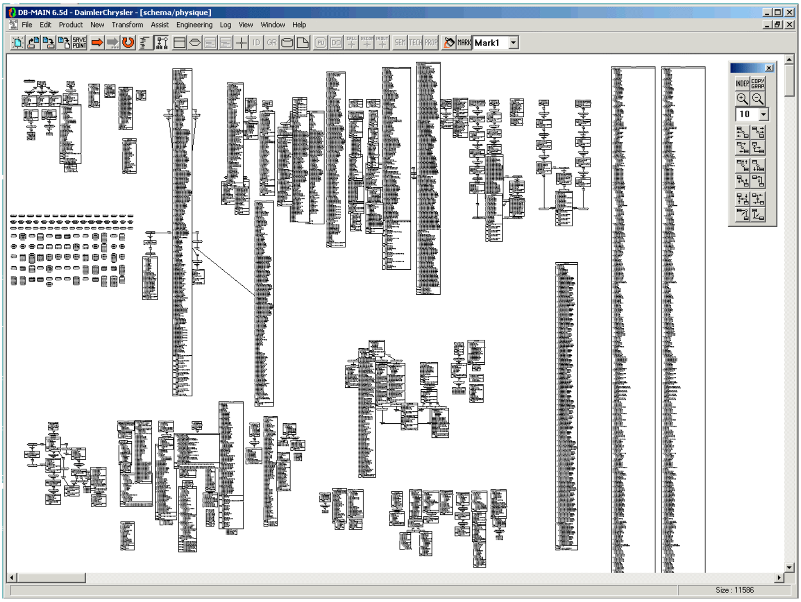
LIBD

Appendix 3

Actual project in IDMS reverse engineering

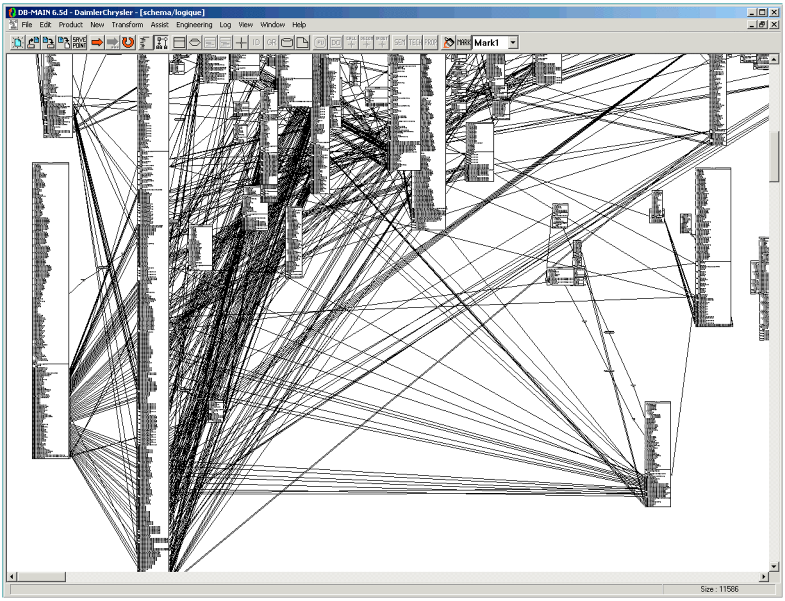
DBRE - Physical schema (IDMS) - *Excerpt*

LIBD



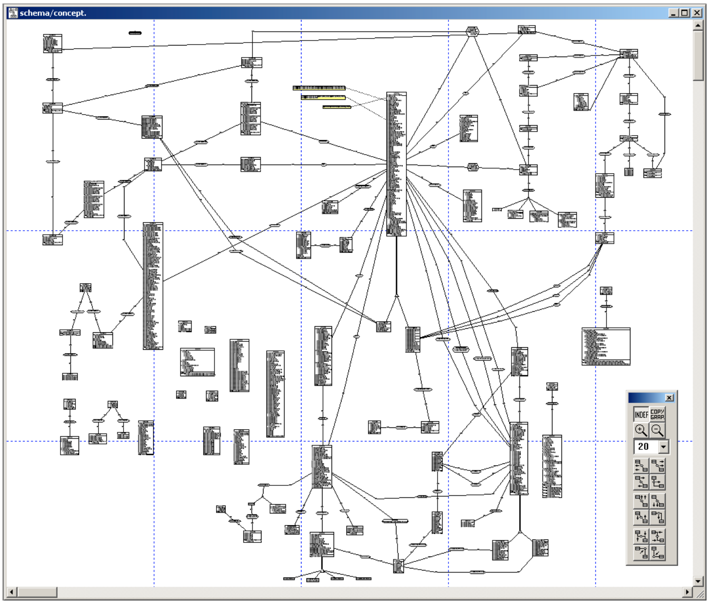
DBRE - Logical schema (IDMS) - *Excerpt*

LIBD



LIBD

DBRE - Conceptual schema - *Excerpt*



LIBD

DBRE - Logical schema (Relational) - *Excerpt*