A Cognitive Approach to the Reconstruction of ER Schema from Database Applications

Oreste Signore
Mario Loffredo - Mauro Gregori - Marco Cima

ISCIS IX
Using Procedural Patterns in Abstracting Relational Schemata

Oreste Signore
Mario Loffredo - Mauro Gregori - Marco Cima

SEAL (Software Engineering and Applications Laboratory)
CNUCE - Institute of CNR - via S. Maria, 36 - 56126 Pisa (Italy)
Phone: +39 (50) 593201 - FAX: +39 (50) 904052
E.mail: oreste@vm.cnuce.cnr.it

WPC’94
3rd Workshop on Program Comprehension
November 14-15, 1994
Washington D.C.
Reconstruction of ER Schema from Database Applications: a Cognitive Approach

Oreste Signore
Mario Loffredo - Mauro Gregori - Marco Cima

SEAL (Software Engineering and Applications Laboratory)
CNUCE - Institute of CNR - via S. Maria, 36 - 56126 Pisa (Italy)
Phone: +39 (50) 593201 - FAX: +39 (50) 904052
E.mail: oreste@vm.cnuce.cnr.it

ER ‘94
The Thirteenth International Conference on
The Entity-Relationship Approach

*Business Modelling and Re-Engineering*

December 13-16 1994

Manchester, UK
Data Base Reverse Engineering
- motivations
- related work

The proposed methodology
- overall architecture
- three phases
- the “clued” approach
- the indicators’ matrix

Conclusions
Maintenance and Re-Engineering

2 Maintenance (corrective, adaptive or perfective):
• up to 95% of EDP department activity
• we must understand the program semantics and the basic design issues

_CHANGEABILITY_

<table>
<thead>
<tr>
<th>Maintain</th>
<th>Enhance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discard</td>
<td>Re-engineer</td>
</tr>
</tbody>
</table>

_Business value_

2 LOGICAL LEVEL

 PHYSICAL LEVEL

Old product  Analyzed product  Structured product

ANALYSIS  RESTRUCTURING

Product abstraction

New product

RDBRE - 2
Database Reverse Engineering: **WHY?**

- More recent DBMS have new features
  - constraints can be defined at schema level

- Reverse Engineering towards Object-Oriented

- Database applications are very often of crucial importance

- Recovering design issues

- The DBRE is faced with the following problem:
  - given the DDL/host language expression of existing data structures (global, schema and/or views)
  - given known operational requirements (e.g. the DMS performance requirements, etc.)
  - find a possible conceptual schema that could lead to these data structures
We must well understand the FE to perform an effective reverse engineering process.

In FE we have several phases:

- mapping conceptual-logical
- optimisation of the logical schema
- mapping logical-physical
- translation of not directly supported specifications

The sequence of transformations induces a progressive degradation of the schema, that becomes:

- less complete
- less simple
- less readable
- less expressive
Restrictive hypotheses:

- requirements completely mapped onto data structures and constraints
- strict application of the mapping rules
- user needs or environment constraints didn’t force any further restructuring of the schema
- existence of a “naming policy”

Batini, Ceri, Navathe


- simple and limited process
- a suitable initial model
- clear and linear description of the steps to follow to analyse relations and identify the concepts
- a good semantic knowledge of the initial relational schema is supposed

Premerlani, Blaha


- “experimental” point of view
- set of methods, techniques and practical examples
- large set of real cases
Chiang, Barron, Storey

- takes information from the catalog and from the data stored in the relations

Hainaut, Chandelon, Tonneau, Joris

- we can split the solving process in two main subsequent phases:
  - *Data Structure Extraction (DSE)*
    (the reverse of the physical phase)
  - *Data Structure Conceptualisation (DSC)*
    (the reverse of the logical phase)
The DBRE process
Some problems in Data Base Reverse Engineering

In the following databases, try to identify:
- domains’ identity
- IS-A hierarchies
- associative relations
- attributive relations

BOOKS (ID, TITLE, MAIN_AUT, PUBLISHER,...)
AUTHORS (ID, NAME,...)
SEC_AUTH (BOOK_ID, AUTH_ID)
STUDENT (ID, FSTNAME, LSTNAME, COURSE,...)
LOAN (ID, STUD_ID, BOOK_ID, DATE,...)

EMPLOYEE (EID, D1,...,Dn)
MANAGERS (EID, M1,...,Mp)
TECHNICIANS (EID, T1,...,Tq)
SECRETARIES (EID, S1,...,Sr)
SKILL (EID, SKILL, LEVEL)
ENGAGED (EID, PROJECT, PERCENTAGE)
PROJECT (P#, TITLE)

We cannot simply rely on column names.

To capture the semantics, we must consider how the applications make use of the data.
Implicit assumptions:

- a first phase of generation of SQL/procedural facts
- a second phase of generation of catalog facts
The RDBRE process

Initial knowledge base

Phase 1: Identification of the primary keys

Phase 2: Detection of the indicators

Phase 3: Conceptualisation

The three RDBRE process phases:
- Phase 1: Identification of the primary keys
- Phase 2: Detection of the indicators
- Phase 3: Conceptualisation
The indicators

**Definition of an indicator**
a set of information detectable from one or more available sources (catalog, SQL code, output of a previous analysis phase), that could characterise, in the conceptual model, one or more relational schema items

**Classes of indicators:**

- **schema indicators**
taken from the catalog and the information deduced in the key identification phase

- **key indicators**
taken from the analysis of the primary keys
  help in defining the properties of the PK of a given relation

- **SQL indicators**
taken from the analysis of the SQL commands
  give information about the kind of usage the DML statements make of the table elements

- **procedural indicators**
taken from the analysis of the host language code
  integrate the information supplied by the SQL indicators: made of some typical (standard) patterns for conditional manipulation of the database data

**Examples:**
- fetch loops
- referential integrity constraints’ checking
- actions on tables implementing class hierarchies
Identification of the primary keys: generalities

A trivial case if explicitly defined

If we have only one index with the **UNIQUE** option

PK can be identified as the attribute (or attribute set) the index is defined upon

If we have more than one index with the **UNIQUE** option

- we consider every set as a candidate key
- we calculate the frequencies of usage
- we ask the user to make a choice

If we do not succeed in identifying a primary or candidate key:
we can identify some *indicators* by analysing procedural patterns:

- at least one **WHERE** clause must mention all the columns composing the potential key (a)
- no DML statement making use of these columns and returning a set of tuples should exist (b-g)
## Identification of the primary keys: the SQL patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>WHERE (a_1 = \text{scalar_exp}_1) AND ...\AND (a_s = \text{scalar_exp}_s)</td>
</tr>
</tbody>
</table>
| b | No declaration of a cursor like:  
DECLARE <cursor_id> FOR  
SELECT <selection>  
FROM T  
WHERE \(a_1 = \text{scalar\_exp}_1\) AND \...\AND \(a_s = \text{scalar\_exp}_s\)  
followed by  
OPEN <cursor_id>  
and a loop containing:  
FETCH <cursor_id> INTO <list_of_host_var>  
or:  
No assignment of the selected tuples to an array. |
| c | No statement contains:  
SELECT ALL|DISTINCT <selection>  
FROM T  
WHERE \(a_1 = \text{scalar\_exp}_1\) AND \...\AND \(a_s = \text{scalar\_exp}_s\) |
| d | No statement contains:  
SELECT <function-ref>  
FROM T  
WHERE \(a_1 = \text{scalar\_exp}_1\) AND \...\AND \(a_s = \text{scalar\_exp}_s\)  
where  
function-ref::= COUNT(*) | distinct-function-ref | all-function-ref  
distinct-function-ref::={AVG|MAX|MIN|SUM|COUNT}(DISTINCT column-ref)  
all-function-ref::= {AVG|MAX|MIN|SUM|COUNT}([ALL]scalar-exp) |
| e | No statement contains:  
SELECT <selection>  
FROM T  
WHERE \(a_1 = \text{scalar\_exp}_1\) AND \...\AND \(a_s = \text{scalar\_exp}_s\)  
GROUP BY <column-ref-commalist>  
or  
SELECT <selection>  
FROM T  
WHERE \(a_1 = \text{scalar\_exp}_1\) AND \...\AND \(a_s = \text{scalar\_exp}_s\)  
ORDER BY <ordering-ref-commalist> |
| f | No statement contains:  
SELECT <selection>  
FROM T  
GROUP BY \(a_1, a_2, \ldots, a_s\) |
| g | No statement contains:  
WHERE <scalar-exp> [NOT] IN <subquery>  
or  
WHERE <scalar-exp><comparison> ALL|ANY|SOME <subquery>  
where <subquery> is like  
SELECT <selection>  
FROM T  
WHERE \(a_1 = \text{scalar\_exp}_1\) AND \...\AND \(a_s = \text{scalar\_exp}_s\) |
First phase:
- PK identified
- hypotheses about PK formulated
- existence of candidate keys indicated

Second phase:
- face the difficulties arising from the different semantic richness of ER and relational model
- we must consider:
  - mapping from an ER to a relational model is not unique
  - optimisation choices
  - poorness of the DDL
  - unusual implementation techniques
- we must adopt a “clued” approach
  (a conceptualisation phase will follow)
- the steps:
  - domains’ identification
  - FK’s identification
  - detection of integrity constraints
  - analysis of integrity constraints

Third phase (conceptualisation)
suitable combinations of indicators can lead to the identification of “probable concepts”
Domains’ identification

No ambiguities, thanks to the usage of the extended name:

\texttt{tablename.attributename}

Identification of the attributes defined on the same domain (synonyms):

- we can’t rely on identical types as defined in the catalog:
  \textit{(SQL type checking is weak!)}

Esp.

\textbf{Given the relations:}

\begin{itemize}
  \item \texttt{BOOKS (ID, TITLE, MAIN_AUT, PUBLISHER,\ldots)}
  \item \texttt{AUTHORS (ID, NAME,\ldots)}
  \item \texttt{SEC_AUTH (BOOK_ID, AUTH_ID)}
  \item \texttt{STUDENT (ID, FSTNAME, LSTNAME, COURSE,\ldots)}
  \item \texttt{LOAN (ID, STUD_ID, BOOK_ID, DATE,\ldots)}
\end{itemize}

\textbf{only a query like:}

\begin{verbatim}
SELECT NAME
FROM AUTHORS, BOOKS
WHERE AUTHORS.ID=BOOKS.MAIN_AUT AND BOOKS.PUBLISHER='X'
\end{verbatim}

or:

\begin{verbatim}
SELECT NAME
FROM AUTHORS
WHERE ID IN
    (SELECT MAIN_AUT
    FROM BOOKS
    WHERE PUBLISHER = 'X')
\end{verbatim}

can show that:

\begin{itemize}
  \item \texttt{AUTHORS.ID, BOOKS.AUT} \quad \textbf{synonyms}
  \item \texttt{BOOKS.ID, STUDENT.ID, LOAN.ID} \quad \textbf{not synonyms}
  \item \texttt{AUTHORS.ID}
\end{itemize}
## Domains’ identification (cont’d)

- **some typical patterns**

<table>
<thead>
<tr>
<th>Type</th>
<th>Pattern</th>
</tr>
</thead>
</table>
| equijoin      | ```SQL
SELECT ...
FROM T1, T2
WHERE ... T1.ATTR = T2.ATTR'...
``` |
| multiple join | ```SQL
SELECT ...
FROM T1, T2, T3, ...
WHERE ... T1.ATTR\{1\} = T2.ATTR\{2\} AND T2.ATTR\{2\} = T3.ATTR\{3\}...
``` |
| nested queries| ```SQL
SELECT ...
FROM T1, ...
WHERE ... T1.ATTR [NOT] IN (SELECT T2.ATTR'
FROM T2, ...
WHERE ...)
```
|               | ```SQL
or:
WHERE ... T1.ATTR\{\} = (SELECT T2.ATTR'
FROM T2, ...
WHERE ...)
``` |
| auto-join     | ```SQL
SELECT A.STAFF_ID
FROM STAFF A, STAFF B
WHERE A.SALARY > B.SALARY AND A.SUPERVISOR = B.STAFF.ID
``` |

- **some other cases of semantic equivalence:**

  INSERT INTO <table> (<column-commalist>)
  SELECT <selection-commalist>
  <table-exp>.  
  *(semantic equivalence of the corresponding attributes in <column-commalist> *and* <selection-commalist>)*

- **the usage of host variables induces some additional complexity (data dependences must be detected)**

  ```SQL
SELECT NAME
FROM AUTHORS A, BOOKS B
WHERE A.ID = B.AUT AND B.TITLE = :book
```

  is equivalent to:

  ```SQL
SELECT AUT
INTO :aut_code
FROM BOOKS
WHERE BOOKS.TITLE = :book
```

  ```SQL
SELECT NAME
INTO :aut_name
FROM AUTHORS
WHERE ID = :aut_code
```
Foreign Keys

Three steps

a) Annotate explicitly defined FK
   A trivial case

b) Identification of not explicitly defined FKs
   Given a relation $T$, having a primary key $PK$, we select the synonyms of $PK$ that all belong to a relation $T'$. They are the components of a FK, defined in $T'$, that references $T$.

c) Identification of the FKs that refer an uncertain PK
   For all the relations that only have a Possible Primary Key (PPK) we apply the same procedure. The result is affected by the same uncertainty that affects the PPK.
Referential integrity constraints

Identifying the referential integrity constraints checking embedded in the code can help in validating the ambiguous cases.

Referential integrity constraints checking:
• in less recent DBMSs was a programmers’ task
• in more recent DBMSs can be defined at the schema level (triggers)

Identifying the procedural patterns that implement the constraints’ checking can be of valuable help in re-engineering phase (clean up of the code, homogeneity, etc.)

An example
(procedural pattern to assure the referential integrity when inserting a tuple in the referencing table)

| PROFESSORS (LSTNAME, FRSTNAME, BIRTHDATE, ADDRESS,...) |
| COURSES (COURSE_ID, CLASSROOM, PROF_LSTNAME, PROF_FRSTNAME, PROF_BIRTHDATE,...) |
| EXEC SQL BEGIN TRANSACTION; |
| EXEC SQL |
| SELECT * |
| FROM PROFESSORS |
| WHERE LSTNAME = :prof_lstname AND FRSTNAME = :prof_frstname AND BIRTHDATE = :date; |
| if (SQLCODE == 0) |
| { |
| EXEC SQL |
| INSERT INTO COURSES (COURSE_ID, PROF_LSTNAME, PROF_FRSTNAME, PROF_BIRTHDATE) |
| VALUES (:course, :prof_lstname,:prof_frstname :date); |
| EXEC SQL COMMIT WORK; |
} else <call of the error_handling routine>
Referential integrity constraints: analysis

Identifying FKS and referential integrity constraints can help in recognising the relationships.

The procedural indicators can be used to:

- identify the FKS
- confirm or reject the hypotheses

Some patterns are very similar:

```sql
EXEC SQL BEGIN TRANSACTION;
EXEC SQL
   SELECT *
FROM CUSTOMERS
WHERE COUNTRY = :zone;
if (SQLCODE == 0)
{
   EXEC SQL
   INSERT INTO AGENTS (AGENT_ID, ..., ZONE)
   VALUES (:agent, ..., :zone);
   EXEC SQL COMMIT WORK;
}
else <call of the error_handling routine>
```

This pattern implements a constraint, but not a referential integrity constraint, as the existence check is performed on a non-key field.
Referential integrity constraints: the checking algorithm

LEGENDA
- T_ed: referenced table
- K_ed: referenced key
- T_ing: referencing table
- K_ing: referencing key

Detection of a control pattern

1. \( \exists P: \) possible_primary_key(T_ed, K_ed)
   - T
   - F

2. Assertion
   - foreign_key(T_ing, K_ing, T_ed)

3. Annotate the control of a dynamic constraint
   - possible_primary_key(T_ing, K_ing, T_ed)

4. Annotate the presence of the control:
   - it could be a control of a dynamic constraint
   - it could be wrong the fact possible_primary_key(T_ed, K)
     where \( K \neq K_ed \)

5. Assertion
   - foreign_key(T_ing, K_ing, T_ed)

6. Confirmation of:
   - primary_key(T_ed, K_ed)
   - foreign_key(T_ing, K_ing, T_ed)
Third phase: conceptualisation

A simple and extensible paradigm:

the indicators’ matrix

- rows correspond to ER concepts (with or without direct mapping)
- columns corresponds to indicators’ categories
- every cell $C_{ij}$ contains the indicator of $Class_j$, that can be used for the identification of the $Concept_i$

Preceding phases populate the cells making use of:
- Models and mapping rules knowledge
- practical knowledge deduced from the implementation experience

Quality and quantity of the indicators affect the concepts' identification.
The indicators’ matrix
## The indicators’ matrix
The IS-A hierarchies

### Conceptual schema

<table>
<thead>
<tr>
<th>Conceptual schema</th>
<th>The relations' schemas</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE (EID, D1,...,Dn)</td>
<td>EXEC SQL</td>
</tr>
<tr>
<td>MANAGERS (EID, M1,...,Mp)</td>
<td>INSERT INTO EMPLOYEE (EID, D1,...,Dn) VALUES (:id, :d1,...,:dn);</td>
</tr>
<tr>
<td>TECHNICIANS (EID, T1,...,Tq)</td>
<td>switch (role)</td>
</tr>
<tr>
<td>SECRETARIES (EID, S1,...,Sr)</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>case '01':</td>
</tr>
<tr>
<td></td>
<td>EXEC SQL</td>
</tr>
<tr>
<td></td>
<td>INSERT INTO MANAGERS (EID, M1,...,Mp) VALUES (:id, :m1,...,:mp);</td>
</tr>
<tr>
<td></td>
<td>break;</td>
</tr>
<tr>
<td></td>
<td>case '02':</td>
</tr>
<tr>
<td></td>
<td>EXEC SQL</td>
</tr>
<tr>
<td></td>
<td>INSERT INTO TECHNICIANS (EID, T1,...,Tq) VALUES (:id, :t1,...,:tq);</td>
</tr>
<tr>
<td></td>
<td>break;</td>
</tr>
<tr>
<td></td>
<td>case '03':</td>
</tr>
<tr>
<td></td>
<td>EXEC SQL</td>
</tr>
<tr>
<td></td>
<td>INSERT INTO SECRETARIES (EID, S1,...,Sr) VALUES (:id, :s1,...,:sr);</td>
</tr>
<tr>
<td></td>
<td>break;</td>
</tr>
</tbody>
</table>

A typical insertion pattern for a disaggregate hierarchy
The IS-A hierarchies

EMPLOYEE

MANAGERS
SECRETARIES
TECHNICIANS

Conceptual schema

EMPLOYEE (EID, D1,..., Dn, M1,..., Mp, T1,..., Tp, S1,..., Sr)

The relation's schema

switch (role)
{
  case '01':
    EXEC SQL
    INSERT INTO EMPLOYEE (EID, D1,..., Dn, M1,..., Mp)
    VALUES (:id, :a1,..., :an, :m1,..., :mp);
    break;
  case '02':
    EXEC SQL
    INSERT INTO EMPLOYEE (EID, D1,..., Dn, T1,..., Tq)
    VALUES (:id, :a1,..., :an, :t1,..., :tq);
    break;
  case '03':
    EXEC SQL
    INSERT INTO EMPLOYEE (EID, D1,..., Dn, S1,..., Sr)
    VALUES (:id, :a1,..., :an, :s1,..., :sr);
    break;
}

A typical insertion pattern for an aggregate hierarchy
## Associations’ detection

<table>
<thead>
<tr>
<th>Type</th>
<th>Pattern</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>NULL &lt;foreign_key&gt;</td>
<td>total association</td>
</tr>
<tr>
<td></td>
<td>NOT ALLOWED IN &lt;table&gt;</td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>NULL &lt;foreign_key&gt; ALLOWED IN &lt;table&gt;</td>
<td>partial association</td>
</tr>
<tr>
<td>SQL</td>
<td>SELECT ...</td>
<td>partial association</td>
</tr>
<tr>
<td></td>
<td>FROM ...&lt;table&gt;, ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHERE ...T.FK IS [NOT] NULL</td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Joins FK-PK have clauses:</td>
<td>multiple association</td>
</tr>
<tr>
<td></td>
<td>FROM T</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHERE FK1=:host_var1 AND...AND FKn=:host_varn</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FROM T, T'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHERE T.FK1 = T’.PK1 AND...AND</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T.FKn=T’.PKn</td>
<td></td>
</tr>
</tbody>
</table>

*Typical patterns for the detection of the associations*
Conclusion

As the DB Conceptual Schema is semantically much richer than the Physical DB Schema, when reconstructing an ER schema we must look at the constraints that are maintained at the procedural level, too.

More recent DBMSs offer enhanced possibilities for defining and maintaining the constraints.

We described a RDBRE methodology that makes use of information taken from:
- catalog
- source code

Innovative aspects:
- interpreting how applications make use of the data
- using procedural patterns

Pros:
- the “cognitive” approach
- easy recognition of new patterns

Limitations:
- the methodology must be refined
- no user friendly interface at present

A prototype has been implemented

Future developments:
• integration in TROOP: a reverse engineering tool under development