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

Abstract. In the Forward Engineering phase, the designer modifies the database conceptual schema and produces a logical and physical schema containing some "spurious" relations dictated by design tricks and DBMS's limitations. Therefore the rebuilding of the database conceptual schema from the physical database structure is a fundamental issue in the re-engineering and design recovery processes. Some proposed methodologies try to get the necessary information from the schema and from the content of the database, paying little attention to the analysis of the usage that the programs make of the data, while this can be thought as the most reliable source of information. In this paper we present an approach to the reverse engineering based on the identification of schema, primary key, SQL and procedural indicators that lead to the assertion of Prolog facts and, by some heuristic rules, to the rebuilding of the conceptual schema.

1 Introduction

The amount of resources involved in the applications' maintenance and the number of investments made by the companies in the implementation of legacy systems unable to take vantage from the enhanced capabilities of the new technologies, give great importance to the recovery of the existing software and the reuse of both the code and the design meaningful aspects.
The Reverse Engineering (RE) is a system analysis process whose aims are the identification of the system components and their interrelationships and the creation of system representations in a different fashion or at a higher abstraction level. Usually, a RE process consists in the extraction of design elements or the synthesis of abstractions that are less dependent from implementation, but it can start from any abstraction level [5].
In this context, the re-engineering of database applications from the physical to the conceptual level seems to have great importance, especially if the process concerns obsolete or non documented data.

2 Database Forward Engineering in Practice

Database Reverse Engineering (DBRE) requires a full comprehension of the Forward Engineering (FE) process, as well as the knowledge of the decisions normally taken by designers and programmers.

The mapping from the database conceptual schema to the physical one can be seen as a sequence of transformations that induce a progressive degradation of the schema, that becomes less complete, simple, readable and expressive. Really, every database design methodology must face the problem of the difference between the semantic richness of the ER (or extended ER) model and the simplicity of the target relational model. In [13] the reader can find an analysis of the problems met in the mapping phase and a set of mapping rules.
The application of the mapping rules and the different semantic richness between the source and target models has the obvious consequence that many constraints must be implemented in a procedural way. Therefore it could be very difficult to capture the semantics of the conceptual schema by simply looking at the physical structures.

3 The Reverse Engineering Process

In spite of his relevance in the RE area, very few research efforts have addressed the DBRE activities. At first glance, it seems that even commercial products are able to solve the problem, but we must stress that almost all existing approaches are conditioned by a set of restrictive hypotheses, namely:

- all the conceptual requirements have been translated into data structures and constraints;
- the mapping from the conceptual to the logical and physical schema has been done applying the mapping rules rigorously, without any "trick";
- additional requirements dictated by the user or the host environment did not give raise to further restructuring of the schema;
- the company defined some "naming policy" for the elements of the schema.

However, we must think that a satisfactory Relational DBRE (RDBRE) should be capable of analysing several implementation solutions, both foreseeable and uncommon.
Furthermore, we must be able to analyse schemata implemented in the less recent DBMS environments, that show some well-known DDL limitations to the possibility of explicitly expressing the concepts of primary and foreign keys, referential integrity constraints, domain constraints, etc.
As we pointed out before, the FE phase produces two sets of specifications:

- a set of relations, hopefully in third normal form, that represent the information expressed in the ER conceptual model;
- a set of procedures that implement the constraints implicit in the conceptual model.

Obviously, a generic DBRE process must extract and abstract from these sources all the knowledge it needs to reconstruct the conceptual schema.

Batini, Ceri e Navathe [1] proposed an extremely simple and limited RDBRE process that sketches a series of steps for the analysis of the relations and the identification of the concepts to be taken by a reverse engineer having a good semantic knowledge of the source relational schema. It can be taken as a good starting point.
Another approach, described in [10], faced the problem of DBRE from a more experimental point of view by offering an enhanced set of methods, techniques and practical examples.

Hainaut et al. [7] exposed a short survey on the DBRE and data-oriented applications state of the art. They synthesised some issues related to the difficulties met during a DBRE process, that is independent from the adopted data model. According to the authors, we can see the DBRE problem as the research of a potential conceptual schema able to lead to the physical organisation constituted by the DDL, the host language data structures and the procedural specifications.
Therefore the process solving such a problem has two generally sequential phases:

- *Data Structure Extraction* (DSE), that reverses the physical phase by reconstructing the data structures from their DDL and host language representations.
- *Data Structure Conceptualisation* (DSC), that reverses the logical phase by the identification of a possible conceptual schema starting from the DSE output.

Chiang et al. [4] presented a methodology for extracting an extended ER model from a relational database. Their methodology analyses not only the data scheme, but also data instances. However we must note that the results of this technique not only are not completely decidible[1], but also are limited by the amount and the quality of the data stored in the database.
Moreover we have to remind that very often the knowledge is embedded in the code, without any up to date and reliable documentation, and none of these approaches take into account this peculiar aspect.

4 The Proposed Approach

In this section we present a general summary of the proposed RDBRE process, pointing out the goals, the range of applicability and the innovative aspects.
As in the other approaches, the goal is the reconstruction of the database specifications at an abstraction level pertaining to the analysis design phase, therefore the identification of the conceptual elements that constitute the ER schema.

4.1 Innovative Aspects

In a purely manual, human approach, many semantic indicators, as the name of the elements, similarities between the names of the relations and their keys, can play a significant role. In addition, the human being can make use of several information sources: documentation of applications (if existing) and other less formal, like naming constraints given by the organisation or simply naming traditions.
An essentially automatic approach must rely on structural clues at a very low abstraction level (data instances, elementary data types, structure and key of a relation, etc.).
The innovative aspect is mainly the "cognitive" approach. In fact, it is based on the acquisition of some clues derived from the analysis of the database schema and, primarily, the usage that the applications make of the data. We deduce this last information by identifying some SQL/host language patterns that can be considered significant for the identification of the properties to be found.

[1] For example, the attribute A cannot be a candidate key for the relation T if we find two tuples having the same value for A. Otherwise, we cannot assert anything.

These clues lead to assert facts that will be taken in input by heuristic rules. The resulting knowledge base can be incremented as the user acquires additional information.
Quite obviously, the data oriented applications are the most suitable area where we can apply the proposed methodology.

4.2 Underlying Hypotheses and Problematic

The aim of the proposed RE methodology is the rebuilding of the conceptual ER schema of a relational database. We extract the relevant information from the *relations' schemata* and *program code*.
In every DBMS we have a DDL that allows the definition of the schema. However, the DDLs very often do not completely conform themselves to the present standard, and this is especially true for the less recent DBMSs, whose database more likely can need a re-engineering process. In addition, very often the expressiveness of the schema depends on the choices taken by the designers in the implementation phase. On the basis of these considerations, we did not impose any special constraint on the DDL. We only adopted some concepts and conventions that generally agree with the ANSI and ISO standards, and their most common extensions [6, 9, 8].
Information pertaining to the relations' schemata can be found in the system catalog, whose structure is not substantially different among the various products.
We suppose that it is possible to extract from the catalog some undoubtedly information, like relations' names and attributes, attributes' types and the possible `NOT NULL` constraint. Furthermore, it is possible to detect for every relation the set of indexes, and their possible uniqueness (`UNIQUE` option). Finally, we do not exclude the possibility of having an explicit definition of the primary, candidate and foreign keys. As far as the programs are concerned, we suppose that they can be coded in any host language with embedded SQL.
We can therefore think that a DBRE methodology should consider at least three fundamental aspects: data models' theory, FE methodologies, non standard implementation techniques.
The first aspect does not raise relevant problems: both the relational and the ER models have a clear and well established theoretical soundness.
The FE methodologies show some differences, however, they make use of relatively well standardised methods and techniques. A complexity factor is introduced by the many-to-many relationship among the conceptual and the corresponding relational schemata, but we can think that this complexity is limited and easily manageable.
The last aspect is the most difficult: only a consolidated experience in designing and implementing database applications can help in the identification of the non standard decisions taken as effect of tricky optimisations or constraints imposed by the implementation environment or application requirements.
As an example, let us take the case of the identification of the primary key of a relation, that should correspond to the well-defined concept of the key attribute of an entity. We can have the following cases:

- the DDL didn't support the explicit definition of the keys;
- the designer did not define a unique index on the key's attributes;
- the designer rel
  tuples, and did r
  key's componen
- the primary key
- the tuple identifi

4.3 Guidelines

We move from the
for the definition of

1. We formulate a
   practical experi
   possible usage n
   the rules can lea
   - detection of
     define as ind
     sources (cata
     characterise,
   - Sure identi
     accomplishe
     attributes, a
     hierarchies, r
   - The formula
     rejected on t
2. The re-engineeri
   the user should
   code. So he/she
   assertions based
   knowledge of t
   fundamental rol
3. The results got
   be stored as *asse*
   in the subsequen

Indicators. The ind
categories accordin

- *schema indicat*
  while identifyin
  relation;
- *key indicators:*
  properties of the
- *SQL indicators:*
  about the way t
- *procedural indi*
  information pr
  patterns we use
  integrity constra

- the designer relied on the correctness of the procedures that manipulates the tuples, and did not defined at the schema level the NOT NULL constraint on the key's components;
- the primary key is derived from the identifier defined at the conceptual level;
- the tuple identifier is made by a *timestamp*.

4.3 Guidelines

We move from the discussed problems to present in the following some guidelines for the definition of the RDBRE methodology proposed in this paper.

1. We formulate a *set of heuristics* (or rules) based on the theory patrimony, the practical experiences and some realistic hypotheses on both the nature and the possible usage made of the elements of the relational schema. The application of the rules can lead to:
   - detection of peculiar properties of the analysed items, named *indicators*. We define as 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.
   - Sure identification, on the basis of reliable clues, of the functions accomplished by the items of the relational schema (foreign keys, multivalued attributes, associative tables, etc.) or of conceptual elements (entities, hierarchies, relationships, etc.).
   - The formulation of hypotheses, that afterwards should to be confirmed or rejected on the basis of subsequent investigations or user's explicit request.
2. The re-engineering process cannot be fully automatic, but we have to suppose that the user should have an interaction and the possibility of inspecting the source code. So he/she will confirm the proposed hypotheses or formulate alternative assertions based on the detected indicators, available documentation and semantic knowledge of the application. Evidently the experience of the user can play a fundamental role.
3. The results got by the recognition rules and the decisions taken by the user must be stored as *assertions*, and will populate a knowledge base, that will be available in the subsequent phase.

Indicators. The indicators used during the whole RE process can be classified in four categories according to the information source they can be deduced from:

- *schema indicators*: extracted from the catalog and from the knowledge captured while identifying the keys, provide information about the particular structure of a relation;
- *key indicators:* obtained from the primary keys analysis, help to define the properties of the primary key of a relation ;
- *SQL indicators*: deduced from the parsing of SQL statements, give suggestions about the way the relations are used during data access and manipulation;
- *procedural indicators*: resulted form the host language code analysis, supply the information provided by the SQL indicators by identifying data and control patterns we use to conditionally work on the database, i.e. fetch loops, referential integrity constraints' checks, operations on tables modelling hierarchies, etc.

Schema indicators	Primary key indicators	SQL indicators	Procedural indicators	CONCEPTS
	• doesn't correspond to PK of other relations • doesn't contain FK			**Fundamental entity** • without multivalued attributes • doesn't belong to any hierarchy
• possibile key dominance (PK + 1÷2 attributes) • referential integrity constraints on the FK (the DDL supports their explicit declaration)	• is composed by several FK	• join with the related relations	• referential integrity constraint checks (not explicitly defined by the DDL)	**Relationship**
• key dominance (PK + a single attribute) • non key attribute unique and not null	• doesn't contain FK	• is referred in a number of relations (i.e. its PK is the FK in several relations) • a few or no insert, delete or update (or with DB administrator grants)	• referential integrity constraint checks (not explicitly defined by the DDL)	**Enumerative type relation (decoding table)**
• key dominance (PK + a single attribute) • non key attribute unique and not null	• corresponds to PK of other relations	• the I/O procedures on the relations including the related PK are executed using it always for coding/decoding operations • the selection and manipulation statements on the relation including the related PK can use it for coding/decoding a key		**Key coding relation**
• "full PK" • referential integrity constraints on the FK if the DDL allows to define them explicitly	• PK includes a FK and a single attribute • usually FK and PK of the referred relations have the same name	• is referred by a single relation	• referential integrity constraint checks (not explicitly defined by the DDL)	**Simple type multivalued attibute**
• key dominance (PK + a single attribute) • non key attribute not null • referential integrity constraints on the FK (the DDL supports their explicit declaration)	• PK includes a FK and a single discriminant attribute • usually FK and PK of the referred relations have the same name	• is referred by a single relation	• referential integrity constraint checks (not explicitly defined by the DDL)	**Simple type multivalued attribute** (The single values for each entity are distinct by the value of the discriminant attribute)

Table 1. Matrix of indicators (part 1)

constraints on the FK (the DDL supports their explicit declaration)	same name			attribute)

Table 1. Matrix of indicators (part 1)

Schema indicators	Primary key indicators	SQL indicators	Procedural indicators	CONCEPTS
• "full PK" • referential integrity constraints on the FK (the DDL supports their explicit declaration)	• PK includes a FK and some attributes • usually FK and PK of the referred relations have the same name	• is referred by a single relation	• referential integrity constraint checks (not explicitly defined by the DDL)	**Complex type multivalued attribute**
• referential integrity constraints on the FK (the DDL supports their explicit declaration)	• PK includes a FK and a single discriminant attribute • usually FK and PK of the referred relations have the same name	• is referred by a single relation	• referential integrity constraint checks (not explicitly defined by the DDL)	**Complex type multivalued attribute** (The single values for each entity are distinct by the value of the discriminant attribute)
	• sets of relations (#≥2) with related PK	• each relation belonging to the set has autonomous join with other relations	• decision statements are present to conditionally operate on the tuples of the subsets by using DML statements	**Disaggregate hierarchy of subsets or partitions** (It includes several relations)
		• in general is characterized by associations with several relations	• decision statements are present to conditionally operate on the tuples of the subsets by using DML statements	**Aggregate hierarchy of subsets or partitions** (It includes several relations)
	• corresponds to PK of other relations	• has autonomous join with other relations	• decision statements are present to conditionally operate on the tuples of the subsets by using DML statements	**Generalisation entity of a hierarchy**
	• corresponds to PK of other relations	• has autonomous join with other relations	• decision statements are present to conditionally operate on the tuples of the subsets by using DML statements	**Subset entity**
• in general doesn't have key dominance	• contains FK	• in general has autonomous join		**Weak entity**

Table 1. Matrix of indicators (part 2)

Table 1 reports the set of indicators and the related concepts they are clues for. The table layout is the following:

- the rows correspond to different ER concepts, including both those having a direct mapping to the relational model (strong and weak entities) and those who, on the contrary, can't be modelled directly with single relational objects (hierarchies, relationships with attributes, n-ary relationships, etc.);
- the columns correspond to the categories of indicators;
- every matrix cell contains the indicators of the related column we can start from to deduce the concept of the related row.

4.4 Tools

The general features of the process exposed in the previous paragraph suggested the adoption of a logic programming language like Prolog to implement a prototype.
In fact, this environment allows quite easily to:

- represent the knowledge captured from each process step as a set of *facts* to be inserted in a Prolog database;
- transform the developed heuristics into Horn clauses and therefore in Prolog *goals* in order to recognise the meaningful properties of the schema elements;
- take advantage from the *backtracking* mechanism to repeat the process on the knowledge base updated manually by the user.

Furthermore, we want to outline that the RDBRE proposed approach must be considered as a coherent kernel capable to be adapted to the features of the chosen DDL, DML and host language patterns. Some additional possible refinements of the recognition capabilities can be added to the kernel.
In fact, in a Prolog program new assertions and rules can be progressively added and made immediately executable; thus exploiting all the *rapid prototyping* capabilities of the Prolog language [3, 2].
In addition, as the proposed approach is based on the recognition of particular SQL and programming language patterns, we guess the possibility of executing a code static analysis phase and having available all the necessary information according to the established format [12].

5 The RDBRE Process

The reverse process is performed essentially in three phases:

1. *Identification of primary keys.*
   We make use of all the indications that we can extract from the relations schemata. If it is not possible to identify the keys on this basis, we examine the SQL requests and the presence of fetch loops in the host language, so excluding from the set of the possible keys the set of attributes that do not uniquely identify a tuple. Finally, the human intervention can identify the primary keys by acting on the automatically detected clues.
2. *Detection of the indicators.*
   We look for some indicators we defined, that can lead to the identification, in the conceptual model, of one or more relational schema items. In particular, in this phase we try to identify the foreign keys that define the relationships among

relations, the properties of the relationships, the structure of the primary keys, clues on the hierarchies of set of relations, etc.. Even in this phase the human intervention can be necessary to take decisions in critical situations.

3. *Conceptualisation.*
   We analyse and compare the indicators to identify the "spurious" relations (those modelling relationships, multivalued attributes, decode tables), and formulate hypotheses on the conceptual meaning of the elements of the relational schema. The user can evaluate all these hypotheses, together with the conceptually not identified relations, and can decide to restart some RE process phases. Whilst doing this, he/she can make choices that are alternative to those previously taken.

5.1 Identification of Primary and Candidate Keys

If the DDL supports the explicit definition of primary and candidate keys, this knowledge is imported into the Prolog database by asserting the facts:

primary_key (T, K) or *candidate_key* (T, K)

where T is the name of the relation and K identifies the set of key component attributes.

If we do not find an explicit declaration of the key, we check if there are `UNIQUE` indexes. If there is only one set of attributes the `UNIQUE` index is defined upon, we assume this set as the primary key, otherwise, all the sets are taken as candidate primary keys.

If we do not succeed in the identification of any candidate primary key, we examine the code by acting on the table, searching for the attributes that satisfy the following conditions:

- at least one `WHERE` clause of a SQL statement must mention all them;
- the structure of the code must exclude that the selection returns a set of tuples (loop of fetches, aggregation operators, `ORDER_BY` or `GROUP_BY` clauses, etc.);

For each subset of attributes $S=\{a_1, a_2,..., a_s\}$ satisfying all the previous conditions, we generate the clause:

possible_candidate_key (T, S).

For each relation T not having a primary key, we consider the set of candidate keys and determine the *minimal candidate keys*. If we find only one, we take it as primary key, otherwise, for each minimal cardinality set Y_i we generate the facts:

minimal_candidate_key (T, Y_i).

We determine the primary key for the relation T in two steps:

i. for each Y_i we calculate the ratio:

$$U_T(Y_i) = \frac{S_T(Y_i)}{Q_T} = \frac{\text{total number of usages of } Y_i \text{ in } < \text{search\_condition} >}{\text{n° of SQL statements referring T}}$$

   that gives an estimate of the frequency of usage of the candidate key;

ii. the user can see all the minimal candidate keys Y_i and their $U_T(Y_i)$ to identify the primary key.

In Appendix we report a set of patterns that correspond to Prolog facts and the rule that can deduce from these facts the presence of a primary key.

5.2 Detection of the Indicators

This phase consists in the detection of some peculiar situations that implies the existence of associations between two relations. The first step is the identification of the synonyms, the following are concerned with the identification of the foreign keys and the existence of integrity constraint checks.

Searching for Synonyms. All the cases of homonyms can be easily solved just adopting the *extended name* of the attributes (`tablename.attributename`), while the detection of synonyms requires some additional processing.
Indeed, SQL is a language having simple data types and weak type checking, and therefore the type checking based on the content of the catalog cannot provide reliable information.
Because of this, we detect the synonyms by some SQL indicators that suggest domain identity of two attributes if we find a comparison in a `WHERE` clause (equi-join) and we annotate this fact as the Prolog fact:

synonym_attributes (Table1__Attribute1, Table2__Attribute2).

It must be noted that in some cases the semantic equivalence of the domains could be deduced from the usage of the host variables, as it happens when a join is implemented by separate `SELECT` on different tables. This aspect will be considered in the future, as it requires the correct identification of the data dependencies.

Detection of Foreign Keys. Once the synonymies have been detected, we can look for the foreign keys, that model the associations between tables. The process takes place in three consecutive steps:

1. *Annotation of the explicitly declared foreign keys.*
   This is the simplest case. If the DDL possesses the appropriate mechanisms we have simply to assert:
   foreign_key (Referencing_table, FK_attributes_list, Referenced_table)
   for each table (Referencing_table) where there is the definition of a set of attributes (FK_attributes_list) referring to another table (Referenced_table).
2. *Detection of non explicitly declared foreign keys.*
   Given a relation T having a known primary key PK, we single out the synonyms of the components of PK that all belongs to a relation T'. They are the components of a foreign key, defined in T', referring T. Therefore, we can assert the fact:
   foreign_key (T', FK_attributes_list, T).
3. *Detection of foreign keys referring uncertain primary keys.*
   For all the relations having only an indication of possible primary key (PPK), we have to apply the same process described in the previous step, but we have to transfer the uncertainty of the information about the primary key to the result. Therefore, if we can find a set of attributes of T' that are synonym of the components of the PPK of T, we can assert the fact:
   possible_foreign_key (T', PFK_attributes_list, T).

Referential Integrity Constraints. The uncertainty of the information gained in the previous step makes necessary to attempt to confirm the indications looking at

referential integrity constraints' checks done in the code. To do this, we have to check the references made from the PPK to the PFK.
The existence of triggers, that can be fired at insertion, deletion or updating of tuples, can reveal a referential integrity constraint checking.
We can envisage the possibility of automatic detection of some simple procedural patterns of frequent usage. Alternatively, a manual approach can take place.

5.3 Conceptualisation

In this phase, we can use the previously detected indicators to identify the "spurious" relations that the designer had to introduce in the logical and physical design phase to represent the elements of the ER schema that do not have a direct equivalent (relationships with attributes, n-ary relationships, multivalued attributes, etc.).
We have to refine the knowledge acquired in the previous phases, to identify the features of relations that implement some peculiar conceptual elements. As we have pointed out before, the indicators can be classified in four categories according to the source they derive from. Such a classification is the foundation of a simple and extensible paradigm, the *matrix of indicators*, aimed to the identification of the elements belonging to conceptual schema (Table 1).
The phase of populating the matrix of indicators moves from both the theoretical knowledge about data models and the mapping rules, and the practical knowledge derived from implementation experiences. We cannot give general rules for the identification of the conceptual elements, as it depends on the amount and quality of the indicators in the matrix. As an example, the structure of the primary key can be a "strong" indicator of an associative table, while a high join frequency with the associate tables can be taken as an indicator that confirms a hypothesis. If for a relation we are not able to detect any indicator for any row of the table, we have to:

1. show that it is impossible to clearly state the conceptual meaning of the relation;
2. display all the detected indicators;
3. ask the user to formulate a hypothesis.

Some inconsistencies can arise; in these cases, it can be necessary to re-execute some phases of the whole process.
In this phase it is possible to recognise the abstraction hierarchies and the cardinality of the associations.
Regarding IS-A hierarchies, we have to remind that they can be mapped in different ways: creation of an aggregation table, that contains all the attributes of the generalisation and specialisation classes, or as many relations as are the classes and subclasses. In both cases, we can find some typical patterns, as shown in fig. 1-2 (with obvious meaning of the variable names).

The cardinality of the associations can be inferred from the existence of some specific DDL options or from some typical procedural patterns. For example, we can deduce the totality or partiality of an association by the specification of the `NOT NULL` constraint or of an outer join implies. The presence of a loop of fetches can suggest that the association is multiple.

```
EMPLOYEE (EID, D1,…,Dn)
MANAGERS (EID, M1,…,Mp)
TECHNICIANS (EID, T1,…,Tq)
SECRETARIES (EID, S1,…,Sr)

EXEC SQL
   INSERT INTO EMPLOYEE (EID, D1,…,Dn)
   VALUES (:id, :d1,…,:dn);
switch (role)
{
   case '01':
      EXEC SQL
         INSERT INTO MANAGERS (EID, M1,…,Mp)
         VALUES (:id, :m1,…,:mp);
      break;
   case '02':
      EXEC SQL
         INSERT INTO TECHNICIANS (EID, T1,…,Tq)
         VALUES (:id, :t1,…,:tq);
      break;
   case '03':
      EXEC SQL
         INSERT INTO SECRETARIES (EID, S1,…,Sr)
         VALUES (:id, :s1,…,:sr);
      break;
}
```

Fig. 1. A typical insertion pattern for a disaggregate hierarchy

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

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;
}
```

Fig. 2. A typical insertion pattern for an aggregate hierarchy

6 Conclusion

In the reconstruction of the ER schema of a database, we can identify the constraints that are maintained at the procedural level, too. Therefore it becomes possible to re-engineer the applications getting all the advantages provided by the most recent DBMS, that allows the definition of many constraints directly at the schema level.
In this work we described the problematic and the guidelines of a RDBRE that takes in input the DBMS catalog and the source code, and is able to deduce the database conceptual schema.
The proposed methodology shows some innovative aspects in respect with others presented in the literature. In fact, the particular "cognitive" approach recognises specific properties of the relations not only taking into account the knowledge that can be deduced from the database structure, but also attempting to interpret how the applications make use of the data. This is done searching for some SQL and procedural patterns that can be considered significant for the detection of the searched properties.
A Prolog prototype of an "expert system" implements the methodology. The knowledge base containing the pattern recognition rules can be easily modified, both adding new rules to increment its potentiality and tailoring the rules to the specific DDL, DML and host language.
As the proposed approach requires a user interaction, it is evident that the recognition potentiality and the quality of the knowledge base depend on the user's experience in data oriented applications development and his/her knowledge of the application environment.
As a future development, to make a more extensive test on a suitable number of applications, we foresee the integration of the tool in TROOP [11], a reverse engineering tool currently under implementation.

References

1. Batini C., Ceri S., Navathe S.B.: Conceptual Database Design: An Entity-Relationship Approach, The Benjamin/Cummings Publishing Company,Inc., 1992.
2. Canfora G., Cimitile A., Munro M.: A Reverse Engineering Method for Identifying Reusable Abstract Data Type, Proceedings of IEEE Working Conference on Reverse Engineering, Baltimore 1993 (ISBN 0-8186-3780-3)
3. Ceri S., Gottlob G.: Normalization of Relations and Prolog, Communications of the ACM, June 1986, Vol.29, No. 6.
4. Chiang R.H.L., Barron T.M., Storey V.C.: Reverse engineering of relational databases: Extraction of an EER model from a relational database, Data & Knowledge Engineeering, Vol. 12, No. 2 (March 1994), pp. 107-142
5. Chikofsky E.J., Cross II J.H.: Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software, 7(1): 13-17, January 1990.
6. Date C.J., White C.J.: A guide to DB2 - Second edition, Addison-Wesley(1987)
7. Hainaut J-L., Chandelon M., Tonneau C., Joris M.: Contribution to a Theory of Database Reverse Engineering, Proceedings of IEEE Working Conference on Reverse Engineering, Baltimore 1993 (ISBN 0-8186-3780-3)
8. Information technology - Database languages - SQL2; ISO standard N. 9075

9. Korth H.F., Silberschatz A.: Database System Concepts, McGraw-Hill International Editions, Second Edition, 1991.
10. Premerlani W.J., Blaha M.R.: An Approach for Reverse Engineering of Relational Databases, Proceedings of IEEE Working Conference on Reverse Engineering, Baltimore 1993 (ISBN 0-8186-3780-3)
11. Signore O.,Loffredo M.:Re-Engineering towards Object-Oriented Environments: the TROOP Project, Proceedings of The Eighth International Symposium on Computer and Information Sciences (ISCIS VIII), November 3-5, 1993, Istanbul, Turkey (Sponsored by IEEE)
12. Signore O., Loffredo M.:A Repository based Tool for Re-Engineering towards an Object Oriented Environment, ERCIM Database Research Group Workshop 4, May 3-5, 1993, ICS-FORTH, Crete, Greece
13. Teorey T.J., Yang D., Fry J.P.: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, ACM Computing Surveys, Vol.18, No.2, June 1986.

Appendix

In the following, we list a set of SQL and procedural clues revealing that a certain subset S of the attributes of a table T could candidate itself as a possible primary key for T. In particular, the first predicate asserts that the attributes of S are used to select a certain subset of data tuples of T; whilst the remaining facts assert that the selection is expected to return only one tuple.
For each indicator, we show the asserted Prolog fact.
The term `commalist` simply identifies a list of elements of the specified type, separated by a comma.

- *rule_a(T, S)*
  At least one SQL statement contains:
  `WHERE a1=<scalar_exp1> AND...AND as=<scalar_exps>`

- *rule_b(T, S)*
  No declaration of a cursor like:
  ```
  DECLARE <cursor_id> FOR
  SELECT <selection>
  FROM T
  WHERE a1=<scalar_exp1> AND...AND as=<scalar_exps>
  ```
  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.

- *rule_c(T, S)*
  No statement contains:
  ```
  SELECT ALL|DISTINCT <selection>
  FROM T
  ```

```
WHERE a1=<scalar_exp1> AND...AND as=<scalar_exps>
```

- *rule_d(T, S)*
  No statement contains:
  ```
  SELECT <function-ref>
  FROM T
  WHERE a1=<scalar_exp1> AND...AND as=<scalar_exps>
  ```
  where
  ```
  function-ref::= COUNT(*) | dist-function-ref | all-function-ref
  dist-function-ref::= {AVG|MAX|MIN|SUM|COUNT}(DISTINCT column-ref)
  all-function-ref::= {AVG|MAX|MIN|SUM|COUNT}([ALL]scalar-exp)
  ```

- *rule_e(T, S)*
  No statement contains:
  ```
  SELECT <selection>
  FROM T
  WHERE a1=<scalar_exp1> AND...AND as=<scalar_exps>
  GROUP BY <column-ref-commalist>
  ```
  or
  ```
  SELECT <selection>
  FROM T
  WHERE a1=<scalar_exp1> AND...AND as=<scalar_exps>
  ORDER BY <ordering-ref-commalist>
  ```

- *rule_f(T, S)*
  No statement contains:
  ```
  SELECT <selection>
  FROM T
  GROUP BY a1, a2,..., as
  ```

- *rule_g(T, S)*
  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 a1=<scalar_exp1> AND...AND as=<scalar_exps>
  ```

In fig. 3, we present the Prolog predicate that warns the user about the existence of a subset S verifying all the previously listed assertions.
All the Prolog facts pointing out the presence of a primary key for a table T compose the predicate in fig. 4. This assertion considers both the information resulted from the analysis of the database catalog and those deduced by the references to table T in SQL and procedural patterns.

```
rules_verify(T,[S|LAS]) :-
   nl,nl,write('  ATTRIBUTES : '),
   write(S),nl,nl,
   rule_a(T,S),
   rule_b(T,S),
   rule_c(T,S),
   rule_d(T,S),
   rule_e(T,S),
   rule_f(T,S),
   rule_g(T,S),
   (ass_possible(T,S) ->
      (assert(proposed_possible_key(T,S)),
       nl,nl,
       write('      Warning: possible key!'),
       nl);
      true),
   rules_verify(T,LAS).

rules_verify(_,[]).
```

Fig. 3. Predicate `rule_verify`.

```
table_key_detect(T) :-
   nl,nl,nl,write('      TABLE  : '),
   write(T),nl,
   write('      ----------------------------------'),
   nl,nl,
   index_detect(T),
   (primary_key(T,_) -> true;
      (candidate_key(T,_) ->
           primary_key_def(T,candidate);
           (setof(A,not_null_column(A,T),LA),
            subsets(LA,LAS),
            rules_verify(T,LAS),
            acquire_information(T)))).
```

Fig. 4. Predicate `table_key_detect`.