

Using Automated Database Reverse Engineering for Database Integration

M. R. Abbasifard, M. Rahgozar, A. Bayati and P. Pournemati

Abstract— One important problem in today organizations is the existence of non-integrated information systems, inconsistency and lack of suitable correlations between legacy and modern systems. One main solution is to transfer the local databases into a global one. In this regards we need to extract the data structures from the legacy systems and integrate them with the new technology systems. In legacy systems, huge amounts of a data are stored in legacy databases. They require particular attention since they need more efforts to be normalized, reformatted and moved to the modern database environments. Designing the new integrated (global) database architecture and applying the reverse engineering requires data normalization. This paper proposes the use of database reverse engineering in order to integrate legacy and modern databases in organizations. The suggested approach consists of methods and techniques for generating data transformation rules needed for the data structure normalization.

Keywords—Reverse Engineering, Database Integration, System Integration, Data Structure Normalization

I. INTRODUCTION

NOWADAYS integrated information systems are the necessity of the most organizations. The problem of integrating disparate information systems has been investigated in many research works [8, 19, 20, 22]. Many previous local applications in the organization will need to operate on the new integrated database. Integration of a given set of local databases into one global schema raises multiple challenges. This is due to various kinds of semantic incompatibilities and data inconsistencies that we will describe in the next section. Different database researches discuss various solutions for solving the problem of integrating systems [8, 19, 20]. In this paper we discuss not only integrating current databases but also the integration of legacy systems with the modern systems. In this method database reverse engineering (DBRE) is used as a part of local

Mohammad Reza Abbasifard is member of the Database Research Group, Faculty of Electrical and Computer Engineering School of Engineering, University of Tehran, Tehran, Iran (e-mail: m.abbasifard@ece.ut.ac.ir).

Masoud Rahgozar is member of the Control and Intelligent Processing Center of Excellence, Faculty of Electrical and Computer Engineering School of Engineering, University of Tehran, Tehran, Iran (e-mail: rahgozar@ut.ac.ir).

Ashkan Bayati is member of the Database Research Group Faculty of Electrical and Computer Engineering School of Engineering, University of Tehran, Tehran, Iran (e-mail: ashkanb@uisco.net).

Parisa Pournemati, Alzahra University, Tehran, Iran (e-mail: p_pournemati@alzahra.ac.ir)

applications reverse engineering. The purpose of DBRE is extracting data structure information by paying attention to existing applications database schema and programs source codes [4].

A large number of existing applications are running on mini and mainframe platforms [1]. These systems need to be able to evolve to new technology environments. That is why we focus mostly on DBRE in these environments [13]. Many works have been done about schema extraction from source code, but there has been limited research works on reverse engineering of legacy data files [10, 11, 12, 14, 15, 17, 18].

In legacy systems, most of the data are stored in indexed data files. Examples of such indices are VSAM, ISAM, etc. The semantics used for such data files is not suitable for relational database environment. That means, such files require particular attention and effort to be normalized, reformatted and moved to the modern database environments.

To design integrated (global) database architecture we need to apply reverse engineering for data normalization. The suggested approach of this paper presents a method for data structure normalization along with data transformation rules. In this approach the data structure is first extracted from the program source code, it is normalized and then the data transformation rules are generated. We also propose a mechanism to extract some of our rules from data. An example of such rules is determining the foreign key, relations among tables, etc. The normalization process should be done through interactions with a database design expert. The normalization of legacy data is the prerequisite for sharing and development of an integrated information system. In normalizing the legacy data, transformations such as splitting, merging and/or adding new items may be needed. There are cases that splitting, merging, and joining of the tables, in addition to the items, may be necessary.

The rest of this paper is organized as follows. Section 2 introduces database integration. In section 3 we discuss the fundamentals to database reverse engineering which consists of data structure extraction and normalization, and finally the paper ends with conclusion and recommending further works in section 4.

II. DATABASE INTEGRATION

In this section we initially discuss some of the incompatibilities that may exist between two different database schemata.

1) Semantic incompatibilities [8]

- Naming Conflicts: In any data model, the schemata incorporate names for various entities/objects represented by them. Since these schemata are designed independently, the designer of each schema uses his or her own vocabulary to name these objects. Objects in different schemata representing the same real world concept may contain dissimilar names. Resulting in problems of two types:

- Homonyms: This inconsistency arises when the same name is used for two different concepts.
- Synonyms: This type of naming conflict arises when the same concept is identified by two or more names.

Note that homonyms and synonyms can only be detected by external specification.

- Type Conflicts: These conflicts arise when the same concept is represented by different coding constructs in different schemata.
- Key Conflicts: Different keys may be assigned to the same concept in different schemata.
- Behavioral Conflicts: These conflicts arise when different insertion/deletion policies are associated with the same class of objects in different schemata.
- Missing Data: Different attributes may be defined for the same concept in different schemata.
- Levels of Abstraction: This incompatibility is encountered when information about an entity is stored at dissimilar levels of detail in two databases.
- Identification of Related Concepts: For concepts in the component schemata that are not the same but are related, one needs to discover all the inter-schema properties that relate to them.
- Scaling Conflicts: This incompatibility arises when the same attribute of an entity is stored in dissimilar units in different databases.

2) Quantitative Data Incompatibilities [8]

Data retrieved from two local databases for the same logical data item may be incompatible for the following reasons:

- Different Levels of Accuracy: Different databases may be storing an attribute at dissimilar levels of accuracy.
- Asynchronous Updates: Since each database is managed independently, all databases may not update the value simultaneously.
- Lack of Security: Due to lack of information security at component databases, unauthorized users may have changed the data.

Addressing all such incompatibilities is a part of DBRE and the normalization process (Fig. 1). By using this process a new global database is created that will contain the information coming from legacy systems. So we would not need to keep the data redundantly in two environments.

For accessing data new programs are directly connected to this database. Legacy applications use legacy views in the new

database with the help of legacy data access interfaces (wrapper). Fig. 2 illustrates this. The structure and logic of legacy programs is strongly tied to the legacy data access logic. The implementation or structure of the legacy data is mostly navigational or hierarchical and the logic of the legacy programs has been built around this structure. The simplistic approach of replacing isolated legacy data access statements by equivalent SQL statements will lead to significant and prohibitive performance degradation. An effective transformation of the legacy data access logic to the relational data access logic is not linear. Therefore the legacy data access logic should be considered as a whole and managed through a specialized data access interface. The main objective of creating the "Data Access Mapping Interfaces" is to avoid any alteration in the legacy data access logic in the programs.

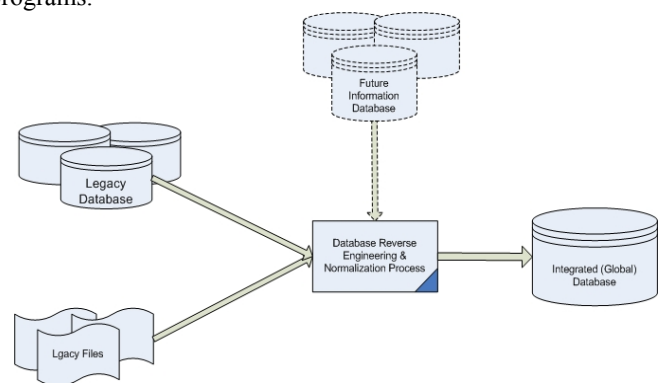


Fig. 1 Legacy Databases and files Integration Process

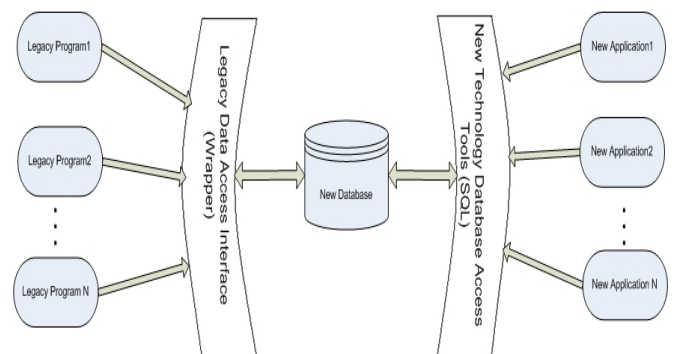


Fig. 2 The application of integrated databases in legacy systems and new systems

III. DATABASE REVERSE ENGINEERING

Many research works emphasize on reverse engineering of relational databases (Fig. 3) [2]. Surprisingly, the most important needs expressed by the industry and the administrations concern IMS, COBOL and CODASYL legacy systems [2].

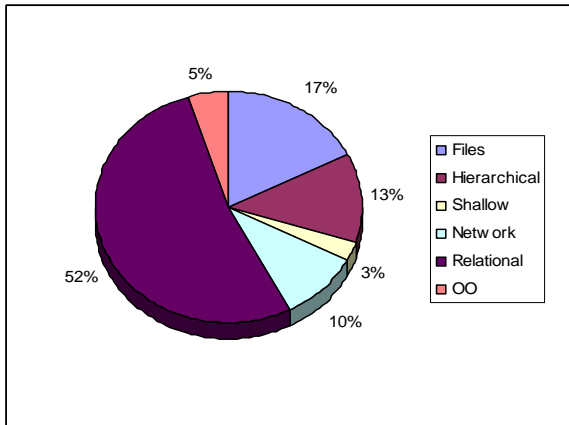


Fig. 3 Distribution of selected research publications according to the DMS model [2]

In our approach DBRE consists of building up the physical, logical and conceptual definitions of the old information system and then mapping the legacy data to the new information system schema. Reverse Engineering (RE) a piece of software consists, among others, in recovering or reconstructing its functional and technical specifications, starting mainly from the source text of the programs [5, 6].

In information systems, applications whose central component is a database or a set of permanent files, the complexity can be broken down by considering that the files or databases can be reverse engineered (almost) independently. In these systems, the semantic distance between the so-called conceptual specifications and the physical implementation is most often narrower for data than for procedural parts (a COBOL file structure is easier to understand than a COBOL procedure) [2].

Database reverse engineering is a suitable solution for recovering and integrating legacy databases and files. DBRE process is a part of software engineering and in reverse of Database Forward Engineering (DBFE) process. These processes can be shown using the following functions [2, 3]:

$$\text{Executable Schema} = \text{DBFE}(\text{Users Information Requirements})$$

$$\text{DBRE} = \text{DBFE}^{-1}$$

$$\text{Conceptual Schema} = \text{DBRE}(\text{Executable schema, domain semantics})$$

Our model, as illustrated in fig. 4, uses reverse engineering methodologies [2] along with focusing efforts on data structure extraction and normalization. The following subsections (3.A and 3.B) will introduce a method for rule mining. The normalization method is introduced in Subsection 3.C.

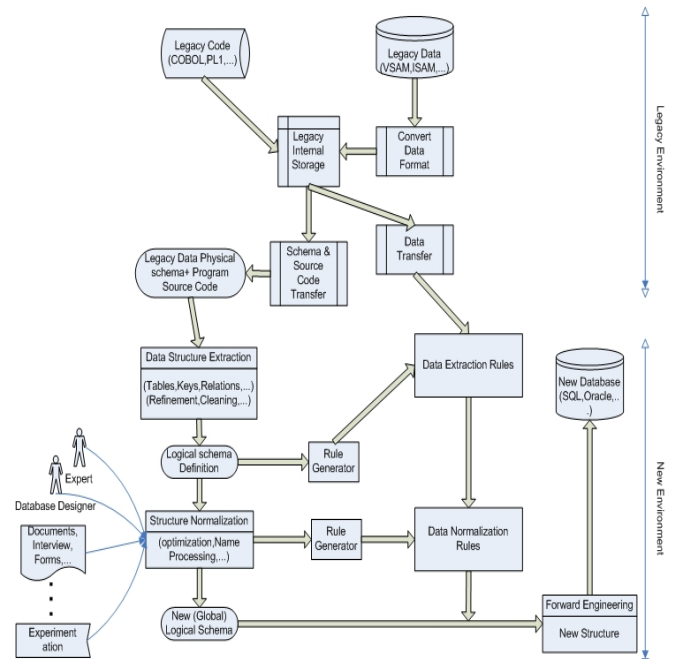


Fig. 4 Database Reverse Engineering & Normalization Process

A. Data Structure Extraction

In this section we describe how to recover the complete Database schema, including all the implicit and explicit structure and constraints. Real database systems generally supply a description of their schema in some readable and process able form. The problem is more complex for standard files, since in many cases they lack a Clear description of their structure. Analysis of many source programs provides a partial view of the file and record structure only. For most real-world applications, this analysis must go well beyond the mere detection of the record structures declared in the programs [9].

Legacy data and source codes contain the data structure that is used in the DBRE process. Legacy codes that do not have complete semantic information are imported to the new environment directly. In the next step, format of legacy data is changed and migrated to new environment. In this step, the available data structure is extracted from the source code via an automatic tool. The tool allows for refinement and cleaning of the data structure. After refinement and cleaning we are left with a data structure. The data structure helps to generate rules for extracting the legacy data.

The next sections describe schema extraction by rule mining, schema refinement and schema cleaning, which are parts of the data structure extraction process. The schema extraction by rule mining idea mentioned in this paper is used to improve schema extraction and to find the relations among extracted tables. To the best of our knowledge this idea has not been mentioned in any of the previous works in literature.

1) Schema refinement

The schema refinement process is a complex task through which various information sources are searched for evidence of implicit or lost constructs. The explicit physical schema obtained so far is enriched with these constructs, thus leading

to the complete physical schema. The complexity of the process mainly lies in the variety and in the complexity of the information sources. Indeed the implicit constraints are hidden, among others, such as in procedural sections in the application programs, JCL scripts, GUI procedures, screens, forms and reports, triggers and stored procedures. To perform reverse engineering of one application, usually more than one of those potential sources of information need to be analyzed and for each one there exists more than one way to express a constraint. For example, in a COBOL program there are at least six different ways to validate a reference constraint. In addition, the non encoded part of the system must be analyzed as well because it can provide evidence for lost constructs. This part includes file contents (the data), existing documentation, experimentation (execution of the program), user and programmer interviews as well as the environment behavior. Environment's behavior is the constraint that is enforced by the environment of the application. For example, the list of the customers is provided by another application that verifies all the constraints and thus the current application does not validate that the customer number is a unique identifier. So it is impossible to discover the identifier of the customer by the analysis of the current application [2, 3, and 9].

2) *Schema cleaning*

This process transforms the complete physical schema into a complete logical schema by removing or transforming all the physical constructs into logical ones. All the physical constructs can be discarded at this point because they do not provide any information about the database's logical structure. These constructs were useful for technical reasons such as optimizing the performance of the database or to implement access mechanisms [2, 3].

B. *Schema Extraction by Rule Mining*

To the best of our knowledge all previous works in literature have done very little research on extracting the rules from the legacy data. Where as, such legacy data contains vital information about the rules that can be retrieved implicitly. Our proposal is to use the legacy data in order to extract a more complete rule. From the legacy data we can extract rules such as primary key, foreign key, relations among tables etc. The following examples will further illustrate this point:

1) *Example 1 (Foreign key extraction):*

Assume we have two tables, A and B, where the fields are defined in both. As well the primary key of A is also defined. To find the foreign key of table B, we only consider fields, in B, that are same type as the primary key in A as candidates of foreign keys. From the candidates retrieved we determine the foreign key of table B, by choosing the field(s) where their values are repeated as primary key in A. The following equation determines the above explained:

$$\text{Foreign key}_{Table B} = \left\{ \begin{array}{l} \text{Type}(PK_{Table A}) = \text{Type}(FK_{Table B}), \\ \forall V \in \text{Values}(FK_{Table B}) \Rightarrow V \in \text{Values}(PK_{Table A}) \end{array} \right\}$$

2) *Example 2 (Relations among Tables):*

Assume, using the above method, we determined the foreign key in table B. Given the primary key in A we want to determine the relations among tables A and B. If the values in our foreign key are unique we can find determine the relation among table A and B to be one-to-one, else the relation is one-to-many. The following equation determines the above explained:

$$\text{Relation}(Table A, Table B) = \left\{ \begin{array}{l} \forall i, j; V_i, V_j \in FK_{Table B}, \\ \forall i \neq j; V_i = V_j \Rightarrow \text{Relation}(A, B) = Table A \overset{1}{\leftrightarrow} Table B \\ \forall i \neq j; V_i \neq V_j \Rightarrow \text{Relation}(A, B) = Table A \overset{1}{\leftrightarrow} Table B; \end{array} \right\}$$

C. *Data Structure Normalization*

The next process is the normalization of the extracted structures. It is necessary to use normalization process to better integrate the legacy databases files. Legacy data normalization is a prerequisite for sharing and extending of the unified Information System.

As mentioned before, for many legacy systems the data stored in indexed files are at least as important as those stored in database tables. They need particular attention because they need more normalization along with the need to be reformatted and moved to relational data bases. In an ideal information system schema's data needs to be stored in a unified and integrated format. The following guidelines should be considered when data is to be normalized and imported into an integrated database [1,7]:

- The indexed files are going to be evolved to a RDBMS environment,
- Their definitions are to be evolved to a normalized model,
- They are to be integrated again into the legacy programs environment,
- The legacy programs codes (i.e. their structures and data access logics) are to be fully respected,
- The same data is going to be shared with future applications using new technology tools.

Reverse engineering the procedural part of an application is much easier when the semantic structure of the data has been elicited [7]. For most legacy information systems, the conceptual and logical designs of the legacy indexed files are rather poor [16]. In this step because of insufficient semantic data in extracted legacy codes, the need to define new data for new database and the lack of automation it is necessary to use an expert(s). In order for the expert to determine a more complete semantics easier resources such as documents, interview with employees of organization, forms, reports and even various experiments are used. The design of legacy data needs significant changes in order to be converted to RDBMS environment and fully normalized. Legacy data normalization

is a prerequisite for sharing and extending of the unified Information System. This includes many topics to deal with, such as, splitting, atomizing or adding new data items or splitting, joining and merging of the tables.

Then above mentioned tool gives the necessary information, with which we can automatically define new rules. These new rules help in normalizing the data. It should be mentioned that the whole processes performed in the normalization step is saved for future use.

After normalization we reach the new logical schema level, a prerequisite for the forward engineering process, name processing should be performed.

1) Name processing

Name processing consists in changing names to make them more expressive and /or standardized. The name of the objects are the names given by the programmers (as recovered during data structure extraction), who have used some naming rules. Now the names can be changed to give more information on the named objects:

a) Remove common prefixes

A common naming conversion in COBOL consists in prefixing each attribute name by the name (or a short name) of the entity type. This is useful in large programs to ensure the uniqueness of the attribute names. Those prefixes do not give any information, so that they can be removed.

b) Meaningful name

The names of the collections are more meaningful than the corresponding entity types names, so that the entity types name can be replaced by the collections name.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we describe an approach for automated DBRE and data normalization for integrating systems. Reaching an integrated database for organizations is highly precious. Local schemata need to be completely normalized during their integration in a global schema. In our approach we focus deeply on this normalization process. The suggested model provides a schema with complete semantics of legacy databases. This model is also very flexible. Along with normalization process we determine constraint rules, such as foreign key and primary key, by observing patterns in data.

Future works will include completing and improving theories and solutions to perform automated DBRE process.

ACKNOWLEDGEMENTS

This work is supported by grants from TAKFA (National Information and Communication Technology Agenda; High Council of Informatics; Iran). We would like to thank Pouyesh Bargh Co. about preparing test platform for our work.

REFERENCES

[1] Rahgozar M., Oroumchian F., An effective strategy for legacy systems evolution. Journal of Software Maintenance & Evolution. Issue 5, Volume 15, September 2003.

[2] Jean-Luc Hainaut, Introduction to Database Reverse Engineering. LIBD-Laboratory of Database Application Engineering Institut d'Informatique - University of Namur; May 2002.

[3] Jean Henrard, Program understanding in Database Reverse Engineering. Thesis Submitted for the degree of Doctor of Science , University of Namur- belgique; August 2003.

[4] Xiaomin Wu, Murray A., Storey M.-A., Lintern R., A reverse engineering approach to support software maintenance. version control knowledge extraction, Proceedings. 11th Working Conference on 8-12 Nov. 2004.

[5] Chikofski E., Cross J., Reverse Engineering and design recovery: A taxonomy. IEEE Software, Jan. 1990.

[6] Special issue on Reverse Engineering. IEEE Software, January, 1990.

[7] Rahgozar M., Oroumchian F., Migrating Legacy Indexed Files to Unix-RDBMS environment. Journal of Software Maintenance & Evolution. sept. 2003.

[8] M.P. Reddy, B.E. Prasad, P.G. Reddy, Amar Gupta. A Methodology for Integration of Heterogeneous Databases. IEEE Transactions on knowledge and data engineering. Vol 6 . No. 6. Dec 1994.

[9] J-L. Hainaut, J-M.Hick, J.Henrard, D.Roland, V.Englebert, Knowledge Transfer in Database Reverse Engineering A Supporting Case Study. Institut d'Informatique, University of Namur, rue Grandgagnage, 21 - B-5000 Namur; IEEE 1997.

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

[11] Casanova M., Amaral De Sa., Mapping uninterpreted Schemes into Entity-Relationship diagrams, two applications to conceptual schema design. IBM J. Res. & Dev., Vol. 28, No1, 1984.

[12] Nilsson E., G., The Translation of COBOL Data Structure to an Entity-Rel-type Conceptual Schema. Proceedings of ERA Conference, IEEE/North-Holland 1985.

[13] Rahgozar M., Oroumchian F., Automatic Evolution of Legacy Data Objects. WSEAS Int. Conf. on Applied Math. & Comp. Science (AMCOS'02), Rio De Janeiro, Oct., 2002.

[14] Davis K., Arora A., A Methodology for Translating a Conventional File System into an Entity-Relationship Model. Proceedings of ERA, IEEE/North-Holland 1985.

[15] Sabanis N., Stevenson N., Tools and Techniques for Data Remodelling Cobol Applications. Proceedings of the 5th International Conference on Software Engineering and Applications, Toulouse, 7-11 December, pp. 517-529, EC2 Publish 1992.

[16] Blaha M.R., Premerlani W., Observed Idiosyncrasies of Relational Database designs. in Proc. of the 2nd IEEE Working Conference on Reverse Engineering, Toronto, IEEE Computer Society Press, July 1995.

[17] Hainaut Jean-Luc, Chandelon M., Tonneau C., Joris M., Transformational techniques for database reverse engineering, Proceedings of the 12th International Conference on ER Approach, Arlington-Dallas, E/R Institute and Springer-Verlag, LNCS 1993.

[18] Edwards H. M., Munro M., Deriving a Logical Model for a System Using Recast Method. Proceedings of the 2nd IEEE WC on Reverse Engineering, Toronto, IEEE Computer Society Press 1995.

[19] C. Batini, M. Lenzerini, A methodology for database schema integration in the entity relationship model. IEEE Trans. Software Eng. vol. SE-10, no. 6, 1984.

[20] D. M. Dilts, W. Wu, Using knowledge-based technology to integrate CIM databases. IEEE Trans. Knowledge Data Eng., vol. 3, no. 2, June 1991.

[21] Jeong-Mi Kim, JuHum Kwon, Doo-Kwon Baik, RGSN Model for Database Integration. International Conference on Information Systems, 2005.