Chapter V
Data Reengineering of Legacy Systems

Richard C. Millham
Catholic University of Ghana, Ghana

INTRODUCTION

Legacy systems, from a data-centric view, could be defined as old, business-critical, and stand-alone systems that have been built around legacy databases, such as IMS or CODASYL, or legacy database management systems, such as ISAM (Brodie & Stonebraker, 1995). Because of the huge scope of legacy systems in the business world (it is estimated that there are 100 billion lines of COBOL code alone for legacy business systems; Bianchi, 2000), data reengineering, along with its related step of program reengineering, of legacy systems and their data constitute a significant part of the software reengineering market.

Data reengineering of legacy systems focuses on two parts. The first step involves recognizing the data structures and semantics followed by the second step where the data are converted to the new or converted system. Usually, the second step involves substantial changes not only to the data structures but to the data values of the legacy data themselves (Aebi & Largo, 1994).

Borstlap (2006), among others, has identified potential problems in retargeting legacy ISAM data files to a relational database. Aebi (1997), in addition to data transformation logic (converting sequential file data entities into their relational database equivalents), looks into, as well, data quality problems (such as duplicate data and incorrect data) that is often found with legacy data.

Due to the fact that the database and the program manipulating the data in the database are so closely coupled, any data reengineering must address the modifications to the program's data access logic that the database reengineering involves (Hainaut, Chandelon, Tonneau, & Joris, 1993).

In this article, we will discuss some of the recent research into data reengineering, in particular the transformation of data, usually legacy data from a sequential file system, to a different type of database system, a relational database. This article outlines the various methods used in data reengineering to transform a legacy database (both its structure and data values), usually stored as sequential files, into a relational database structure. In addition, methods are outlined to transform the program logic that accesses this database to access it in a relational way using WSL (wide spectrum language, a formal language notation for software) as the program's intermediate representation.

RELATED WORK

In this section, we briefly describe the various approaches that various researchers have proposed and undertaken in the reengineering of legacy data. Tilley and Smith (1995) discuss the reverse engineering of legacy systems from various approaches: software, system, managerial, evolution, and maintenance.

Because any data reengineering should address the subsequent modifications to the program that the program's data access' logic entails, Hainaut et al. (1993) have proposed a method to transform this data access logic, in the form of COBOL read statements, into their corresponding SQL relational database equivalents.

Hainaut et al. (1993) identify two forms of database conversion strategies. One strategy (physical conversion) is the physical conversion of the database where each construct of the source database is translated into the closest corresponding construct of the target database without any consideration of the semantic meaning of the data being translated. One of the problems with this strategy is that the resulting target database produced is of very low quality. The second strategy (conceptual conversion) is the recovery of precise semantic information, the conceptual schema, of the source database through various reverse engineering techniques, and then the development of the target database, using this conceptual schema, using standard database development techniques. This strategy produces a higher quality database with full documentation as to the semantic meaning of the legacy data, but this approach is more expensive in terms of time and effort that it entails (Hainaut et al., 1993a). Hainaut et al.'s approach first uses the physical conversion strategy to convert data and then uses a trace of the program, which accesses the legacy data, in order to determine how the data are used and managed. In this way, additional structures and constraints are identified through the procedural code. Through an analysis of the application's variable dependency graph and of the record and file definitions, data fields are refined, foreign keys are determined, and constraints on multivalued fields are discovered. During the database conceptualization phase, the application's physical constructs of indexes and files are removed and the program's objects of arrays, data types, fields, and foreign keys are transformed into their database equivalents (Hainaut et al., 1993a).

Initially, database reengineering focused on recognizing the legacy database structure and transforming these structures into a new model (Aiken & Muntz, 1993; Joris, 1992; Pomerlani & Blaha, 1993; Sabanis & Stevenson, 1992). The values of legacy data were used solely to identify the legacy system's dependencies in terms of keys between records (Pomerlani & Blaha).

Aebi and Largo (1994), in the transformation of database structures, recognize that the transformation of structural schemas involves many issues. The first issue is that the different attributes and entities of the old system must be mapped to the new schema of the transformed database. Constraints, during the migration from the old to the new system, may be added, dropped, or changed. Entity sets in the new system may be identified by new attributes or by old attributes with changed domains or data types.

Wu et al. (1997), with their "butterfly" approach, assume that the legacy data are the most important part of the legacy system and it is the schema rather than the values of this legacy data that are the most crucial. This legacy data are modified in successive iterations with the legacy data being frozen and used for reading purposes only. The "Chicken Little" strategy allows the legacy system to interact with the target system during migration, using a gateway to serve as a mediator. This gateway is used to translate and redirect calls from the legacy system to the target database system, and then the gateway translates the results of the target database for use by the legacy system and by the legacy database. Although the legacy system is allowed to interact with the target database during migration, each data access involves two database accesses: one to the target database and another to the legacy database (Bisbal, Lawless, Wu, & Grimson, 1999).