Contemporary Management Research
Pages 369-380, Vol. 4, No. 4, December 2008

Migrating to Relational Systems: Problems, Methods, and Strategies

Chang-Yang Lin
Eastern Kentucky University
E-Mail: cy.lin@eku.edu

ABSTRACT

In this review, migration refers to the process of moving from non-relational to relational without manually rewriting all existing applications. The paper illustrates the migration method and process, surveys the current migration products, and discusses managerial issues for making the migration process effective. The discussion of these issues will help the practitioner in planning migration projects.

Keywords: Relational Migration, Relational Data Bases, Legacy Systems, Data Conversion, Language Transformation, Data Propagation

INTRODUCTION

Over the last two decades, relational platform has been a de-facto standard for building both operational and analytical applications. For instance, current enterprise systems have utilized relational databases to integrate applications across functional areas; data warehouses have incorporated relational databases as a foundation to support data mining operations and analytical processing. In today's Internet era, relational systems are more effective than non-relational ones in supporting e-business processes due to their superior accessibility, scalability, and openness. Given the continued dependence on relational technology, migrating to relational systems is increasingly becoming a transition choice for over 10,000 major organizations that still employ non-relational data bases such as flat files, hierarchical or network databases for their legacy systems (Schwartz, 2005).

Legacy systems utilize a variety of non-relational database products (e.g., IMS, VSAM, Adabas, DataComm, CA-IDMS), are coded in 2nd or 3rd generation languages (e.g., Assembler, COBOL, JCL, PL/1), and often run on obsolete mainframe computers.

These non-relational legacy systems are problematic for several reasons. First, they are hard to maintain and to expand because there is a general lack of understanding of how the workflows and business rules are built into these non-relational systems. Second, they are difficult to integrate with newer systems in a modern platform because of non-extensibility, incompatibility, and less-openness of the underlying hardware and software of these legacy systems (Bisbal et al., 1999). Third, faster application development in a non-relational environment is hard to obtain because the non-relational systems operate at a lower level of abstraction and require extensive record-at-a-time programming (Lin, 1992).

Despite of these problems, some organizations still keep their legacy systems for mission-critical processes, and many continue to use non-relational technology for creating various applications. Costs for migrating to relational systems can be high, and organizations may not be able to afford abandoning their investment in existing non-relational systems. Table 1 summarizes risks and benefits on relational migration (Lin, 2001).

Two main initiatives may be considered for moving toward a relational platform from non-relational ones. For example, an organization can take an initiative to replace legacy systems with relational-based integrated packages such as an enterprise resource planning system from SAP or PeopleSoft. While this can be an appropriate approach in some instances, it often fails to capture the logic that defines how organizations work. An alternative is to have migration, where legacy systems are converted to modern information architectures that “allow systems to be easily maintained and adapted to new business requirements, while retaining functionality of the original legacy systems without having to completely redevelop them” (Bisbal, et. al., 1999). Although migration initiatives may target any new database platforms, many have been focusing on relational database management systems such as IBM’s DB2, Oracle’s Database, Microsoft SQL Server, or even the object-relational systems.

In this review, migration refers to the process of moving from non-relational to relational without manually rewriting all existing applications. The paper illustrates the migration method and process, surveys the current migration products, discusses managerial issues for making the migration process effective. The discussion of these issues will help the practitioner in planning migration projects. The three steps in relational conversion are suggested for migration.

Table 1 Migrating to Relational Systems: Benefits and Risks

Benefits	Risks
Stay Strategic Competitiveness ▪ Relational databases are data solutions for data warehouses, enterprise resource planning systems, customer relationship management systems, supply chain management systems, and e-business applications. ▪ Relational systems provide the tabular data construct, the set operations, and ongoing enhancements for flexibility, compatibility, and scalability. Enable User-Developed Applications ▪ Relational systems permit flexible information retrieval with the use of SQL, which is much easier to code for users. ▪ Business specialists may exploit relational data marts with data mining tools. Faster Applications Development ▪ Many aspects of relational design and implementation require lower skill levels than do those of other techniques. ▪ IS professionals use front-end 4GL tools that support prototyping and RAD for the development of applications.	Disruption & Extra Costs ▪ The staff must divide their time between maintenance of existing non-relational systems, which requires 100% availability, and migration projects. ▪ The relational DBMS, migration tools, and additional design and programming are all extra and costly. ▪ The significant investments in non-relational systems would have to be scraped. Resistance to Change ▪ Relational systems require dramatic changes in business processes, and resistance to change can be expected. ▪ IS professionals must be familiar with the object-oriented languages such as Java and VB under J2EE or .NET environment. ▪ Business users will have to be trained in the use of SQL and other data mining tools. Poor Performance ▪ Performance degradation is expected for most relational migrations.

MIGRATION PROCESSES AND TOOLS

Migration consists of two separate but related processes: converting the data and converting the programs.

Converting data

Converting data and applications generally follows three stages of the migration process: capture the source data, map or create a relational database model, and migrate the source data (see Figure 1). The capture process performs metadata extraction and creates a source data model. In the map stage, source data model representing files and hierarchical or network databases is reverse-engineered into a conceptual or logical data model. This logical data model is in turn reengineered into a relational schema, which must then be transformed into a physical data base. During the mapping process, the data used in applications program can also be analyzed to derive a logical data model. A synthesis is required for these two types of derived data models. Finally, in the migrate stage, a stored database structure based on the relational database schema is created, and all data from the source database are moved to the relational database.

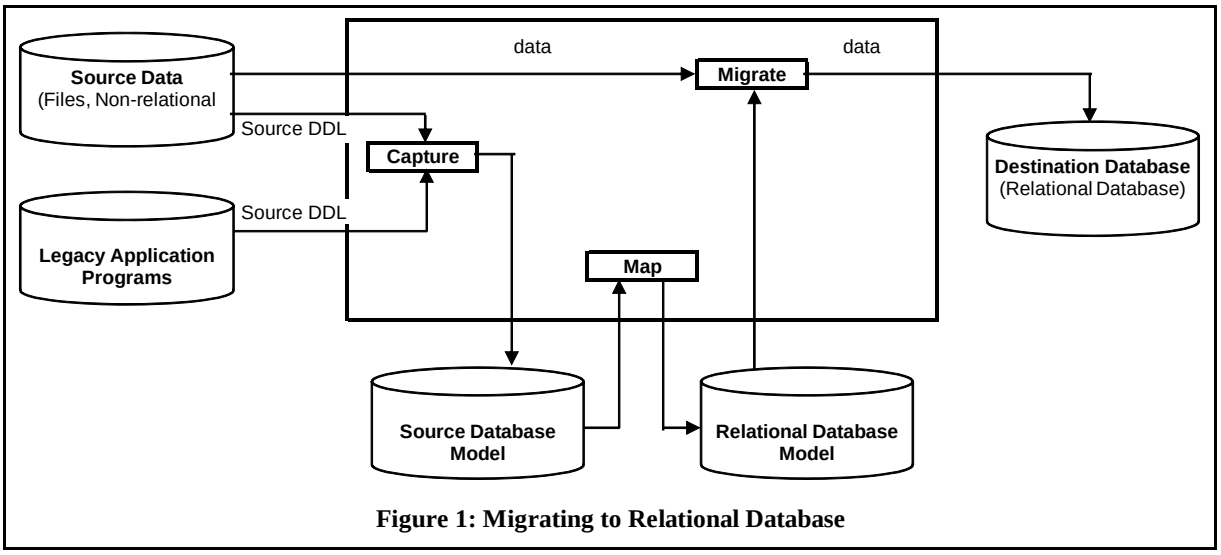

Figure 1: Migrating to Relational Database

The conversion of flat files is straightforward, particularly if the old files are well designed. Migration programs can be written to automatically convert files from the old to the new relational table structure. However, the database conversion effort cannot be performed automatically if record types are not in normal forms, fields for linking record

types are not present or formats of record types are significantly different from the new table structure. In these cases, extra effort using an analysis tool is required to identify common data items and to transform files into normal forms. A file is then mapped into two or more tables. Manual conversion is necessary if the analysis does not result in a satisfactory specification that is independent of the old file structure. An expert system is also needed to analyze the applications programs when a large amount of business logic about the data is built into the applications.

The conversion of hierarchical or network databases is more difficult because records are related by embedded pointers rather than by common fields. Both reverse engineering and reengineering techniques are required to transform non-relational databases into relational databases.

Converting programs

Converting application programs to a relational platform is not as straightforward as converting data, because programs written in a non-relational environment typically use logic that is record oriented. These programs usually must be reverse engineered and rewritten to operate in a relational environment. There are three approaches to converting application programs:

1.Establishing a bridge linking non-relational programs with relational databases.

2.Redesigning or rewriting the access to data.

3.Rewriting the entire program.

The first approach to converting programs is to establish a bridge linking non-relational application programs with relational databases. In this approach, legacy programs remain unchanged, and the transparency layer becomes responsible for providing a bridge between the legacy programs and the relational databases (Hoey, 2006). Due to the incompatible characteristics between SQL and non-relational database languages (Meier, 1994) tremendous effort must be devoted to the development of a translator or a driver that can precompile existing non-relational programs to access relational databases. Such a translator could be used in conjunction with an expert system or CASE tools (e.g., reverse-engineering, reengineering, and code-generating tools) during the migration process.

The second approach is to redesign and rewrite the access of data for the existing non-relational programs. The SQL statements are developed to replace the record-at-time

I/O codes from COBOL or non-relational database languages such as CODASYL and DL/1. To identify such I/O codes, an analyst must isolate data access logic from the rest of the program logic into which the SQL statements will be placed. Because data access for some non-relational applications has similar logic, programs can be written to automatically convert data access for such applications.

Finally, some old programs may have to be completely rewritten from scratch. A comprehensive approach can be used to redesign and rewrite such non-relational programs.

Migration Tools and Products

In addition to computer-assisted software engineering (CASE), the so-called migration tools play a significant role in the conversion of legacy systems to a relational platform. For the earlier legacy migration tools, see the study by Gillenson for detail (Gillenson, 1990). This section identifies, from the Internet search, the current tools and products that automate legacy data and code conversion. Based on their capabilities, these tools are divided into three groups: data and application conversion, code conversion, and data propagation. Table 2 provides a general summary of current relational migration tools and products.

Software vendors such as BluePhoneix, SWS Software, and Anubex focus their attention on the tools that support a comprehensive conversion of the entire environment including analysis, database remodeling, code migration, data-migration program generation, and data propagation. From the products of the BluePhoneix family, the Discovery IT provides an automated, detailed mapping of system-wide IT activity, the DBMSMigrator converts a non-relational database to a relational database, and the LanguageMigrator moves legacy codes to more mainstream COBOL and Java. For example, in moving away from IDMS to the J2EE environment for the DaimlerChrysler, the project team began with BluePhoenix IT discovery tool to finalize a map for the code conversion and database migration. These were then performed using BluePhoenix DBMSMigrator that follows the approach illustrated in Figure 1.

In contrast, some vendors specialize in either data or code conversion. In the case of code conversion, the Acu4GL from Acucorp is used to translate COBOL I-O verbs into SQL, the I2C from m2o converts CA-IDEAL or CA-NATURAL to COBOL and Java with embedded SQL. In the case of data conversion, the TSI's tRelational/DPS allows

forward and reverse re-engineering to migrate an ADABAS system into relational-based data stores and data warehouses, and Fujitsu-Siemens' UDS/SQL enables application programs to access CODASYL and relational databases concurrently by propagating the updates of one database to the other.

Table 2 Relational Migration Products

Vendor (website)	Product	Features and Functions
Data and Application Conversion		
BluePhoneix (bphx.com)	DBMSMigrator LanguageMigrator	IDMS, IMS, ADABAS, VSAM --> DB2, SQL Server, Oracle COBOL, Natural, ADSO --> Java
Treehouse (treehouse.com)	tRelational/DPS	ADBAS --> RDMS-based data stores & data warehouses
SWS (sws.de)	HIREL, IXREL VREL, VIXREL	IMS, CODASYL --> DB2, Oracle VSAM --> DB2, Oracle
Anubex (anubex.com)	Anugen I/O	IDMS, ISAM, VSAM --> Oracle, DB2
Code/Program Conversion		
Computer Associates (ca.com)	Evolveware's S2T AllFusion	Capture business model from legacy code Code generator for J2EE, .Net, Web
Acucorp (acucorp.com)	Acu4GL	COBOL I-O verbs --> SQL
Semantic Designs (semdesigns.com)	DMS	COBOL, JCL, Natural --> Java, XML, PLSQL
m2o (move2open.com)	I2C	CA-Datacom, CA-IDEAL --> COBOL & JAVA with embedded SQL
Data Propagation		
Fujitsu-Siemens (fujitsu-siemens.com)	UDS/SQL	Enables application programs to access CODASYL & relational databases concurrently

PLANNING MIGRATION PROCESS

Relational database migration is an onerous, labor, time- and cost-consuming process. Since it is too risky to convert all applications at once, an organization should select one mission-critical application to convert first. This section discusses migration strategies and suggests the four-steps in relational conversion.

Migration Strategies

Proper strategies will have to be incorporated into a migration plan under different conditions. Three promising migration strategies - data and code conversion, language transformation, and data propagation - can avoid the effort and risk involved in relational conversion. Data and code conversion is an appropriate migration strategy under the condition where data is accessed by a relatively small number of programs (Meier, 1995). First, the legacy data stores are converted to a relational database. Second, the legacy codes are converted to modern languages such as SQL. This strategy has an advantage that it needs only one single copy of data – a target relational database.

Language translation involves building a bridge to link legacy programs with a target relational database. As discussed in the previous session, the bridge or the driver is extremely difficult to build; therefore language translation has not proved to be effective, especially for very large databases where a large number of significant simultaneous changes are involved (Meier, 1995).

Data propagation maintains consistency between legacy data stores and a target relational database by propagating only data changes from one data store to the other. The advantage of this strategy is that it does not convert legacy programs and therefore it avoids the efforts and risks involved in converting and interfacing with legacy applications (Meier, 1995). On the other hand, this strategy will slow the performance or availability of the operational application as the volume of data to be moved increases between two data stores.

Four Steps of Relational Conversion

The relational conversion process can be simplified into four important steps: converting data bases, converting programs, training, and documentation. In general, the conversion process is not strictly sequential. Tasks performed in each step can overlap. A non-sequential view of the conversion process (Lin, 2001) is shown in Figure 2.

The first step is to convert data bases. A set of programs must be used to read or to scan the legacy files to the new relational tables. Tables that are designed to store data about events or transactions are also created. The objective is to establish relational data bases that meet third or higher normal form. In this step, relational data bases and legacy data stores are maintained in parallel during the conversion steps.

The next step is to convert programs. The translators and the I/O routines must be developed to enable existing legacy programs to access relational data bases. For application programs that are not convertible, a total approach that redesigns and rewrites entire programs to fit into a relational structure is required.

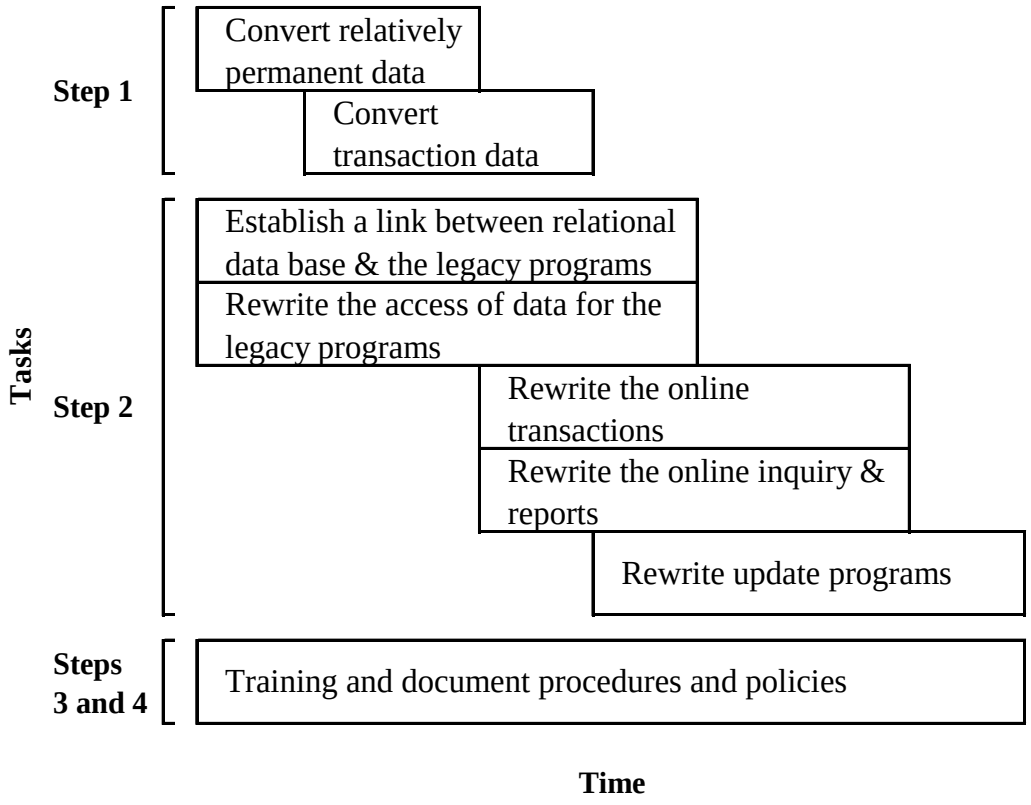

Figure 2 The Four Steps in Relational Conversion

Nevertheless, entire applications programs must be gradually migrated using relational database programming and Web programming. Online transaction programs using 4Gls and Java can be developed to collect data about events or transactions. A new

relational data base that uses visual 4GL components or simply SQL to retrieve information or generate reports should also be opened for user-developed applications. The conversion of the update programs then progresses to the final phase.

Finally, documentation of relational-specific policies and procedures is finalized. The documentation is actually a by-product of the conversion process. Among the most important components are the resolution of strategic issues and the adequate training of systems development professionals in the use of relational tools.

For all of the above steps, migration products (see Table 2) and CASE tools can be used to support the automation of the tasks to some extent.

MANAGERIAL ISSUES AND CONCLUSION

Relational migration is not only a technical task but also a managerial challenge for the organization. Several issues to be discussed below must be resolved for successful conversion to a relational environment. The discussion of these issues, migration strategies, and the suggested steps in relational conversion will help the practitioner in planning migration projects.

Selection of a Relational DBMS

The organization will have to choose an appropriate relational DBMS for its information infrastructure. Due to the increased tactical use of business intelligence, the infrastructure must accommodate mixed workloads of analytic, operational and transactional functions. For transactional functions, the infrastructure not only must support complete tasks for applications development but also must perform acceptably. For analytical functions, the infrastructure must be equipped with high-level visual software components to support user-developed applications.

Several questions should also be addressed before selecting a relational infrastructure: How effective the relational DBMS supports the installations such as mainframe, UNIX, and Linux? If an organization already has several different relational systems, which of these systems should be selected for the migration project? Should an organization choose from one of the top three relational systems (i.e., DB2, Oracle, and Microsoft SQL Server) dominating more than 85% of the market or an open source system like MySQL supporting lower-end or simpler web-based applications development?

Selection of Legacy Migration Tools and CASE Tools

Organizations have typically used automated migration tools to support the conversion process as illustrated in Figures 1 and 2. As each step is being automated, a commensurate drop in manual errors is expected to be realized. It can be a challenge to select the appropriate tools for migration.

When selecting tools to support migration, an organization should evaluate them in terms of their capabilities, extensibilities, openness, and standards. For example, does the tool automate information capturing of existing applications, reverse reengineering, and forward reengineering? Does the tool allow for the third-party add-on components? Does the tool interconnect all other software components? Does the tool support industry-standard metadata exchange formats such as CASE Definition Interchange Format (CDIF)?

CASE tools, which facilitate the conversion of non-relational systems to relational systems, perform highly automated reverse engineering and forward reengineering functions. An organization should also follow applications development standards to select the proper CASE tools for conversion.

The Simultaneous Use of Multiple Databases

The simultaneous use of the source data stores and the target relational database must be planned for when a parallel installation is to be adopted during the conversion process. Procedures necessary for the effective coordination of two databases must be established so that the integrity of the databases is maintained and business is conducted as usual.

Policies and Guidelines

Systems development management must develop relational-specific policies, procedures, methodologies, standards, and guidelines to support the new functions supported by the relational infrastructure. For example, techniques for the program design and construction phase should be updated so that developers must take advantage of the set-at-a-time processing enabled by SQL. In addition, the use of a specific CASE-supported development methodology must be enforced.

Training

Systems development professionals and end users must prepare for relational technology by accepting additional training. Systems development professionals should receive comprehensive training with an emphasis on data management concepts, CASE-driven systems development, set-at-a-time database programming, and web programming under the J2EE environment. End users should learn the use of 4GL and SQL for generating reports and for retrieving information from a relational database or a data warehouse.

REFERENCES

Bisbal, J.; Lawless, D.; Wu, B.; and Grimson, J. (1999). Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issue. *IEEE Software*, 6(5), 1-18.

Comella-Dorda; Wallnau, K.; Seacord, R.; Robert, J. (2000). *A Survey of Legacy System Modernization Approach*. Retrieved October 8, 2008 from http://www.sei.cmu.edu/publications/documents/00.reports/00tn003.html

Gillenson, M. L. (1990). Physical Design Equivalencies in Database Conversion. *Communications of the ACM*, 33, 120-131.

Hoey, S. (2006). IMS to DB2 Migration: Exploring the Options. *Z/Journal*. Retrieved February 21, 2007, from http://www.zjournal.com

Lin, C. (1992). Reengineering to a Relational Data Base Structures. *Software Engineering, 3(1)*, 17-21.

Lin, C. (2001). Relational Database Conversion: Issues and Approaches. In Sanjiv Purba (Ed.), *High-Performance Web Databases: Design, Development, and Deployment* (pp. 559-568). Florida: CRC Press LLC.

Meier, A. (1995). Providing Migration Tools: A Practitioner's View. Proceedings of the 21st VLDS Conference, Switzerland, 635-641.

Meier, A. (1995). Providing Migration Tools: A Practitioner's View. *Proceedings of the 21st VLDS Conference*, 635-641.

Meier, A.; Mercerat, J.; Muriset, A.; Untersinger, J.; Eckerlin, R.; and Ferrara, F. (1994). Hierarchical to Relational Database Migration. *IEEE Software, 11(3)*, 21-27.

Schwartz, E. (2005). Expanding Legacy Apps. *InfoWorld.* Retrieved November 15, 2006, from http://www.InfoWorld.com