Providing Database Migration Tools
A Practitioner's View

Andreas Meier
CSS Insurance
P.O. Box 2568
CH-6002 Lucerne

Abstract:
Relational databases are widely used for end-user computing or developing new business functions. However, most applications with very large databases still need hierarchical or network database systems because companies which have already made significant investments wish them to be protected. Although the transition from one database system generation to the next is important, few promising migration strategies exist. This paper reviews and describes three of them: data and code conversion to assist present applications in running entirely on new generation database technology; language transformation to map one database language to another; and finally, data propagation to maintain the consistency between databases of different systems. The discussion of the pros and cons of these migration alternatives will give the practitioner and the researcher more insight for future work.

Keywords:
Database Migration, Data Conversion, Code Conversion, Language Transformation, Data Propagation.

1. Problems with Heterogeneous Database Environments

Database technology is a keystone for building information systems. Most companies are using not only one, but several database management systems such as hierarchical, network or relational; eventually, they may start to experiment with object-oriented database systems.

In practice, the variety of database systems leads to severe problems: end-users must deal with several database descriptions and different reporting tools; application developers have to be educated in distinct database languages (e.g. DL/1, CODASYL, or SQL); and system specialists need to apply a heterogeneous set of archive, recovery and restart procedures to keep databases consistent. To avoid some of these drawbacks, database migration has become a predominant issue for the practitioner although its potential as yet has not been extensively researched.

In this review, database migration means the process of *moving from one database technology to another* without manually rewriting all existing applications. Two technical problems need to be tackled. First, it has to be decided which database system is the target and how data can be transferred to it; and second, how applications can be converted smoothly without affecting availability or performance. It is important to not forget the significant investments already put into both data and application.

In contrast to multibase or client-server applications, migration strategies aim to give up one database system generation in favor of another. In practice, this means not only avoiding unnecessary licence costs but also *profiting from one homogeneous software environment.* Computer specialists as well as end-users are then no longer confronted with different methods, database descriptions, query languages and reporting tools.

This paper gives practical solutions to database migration by describing three important software strategies: data and code conversion, language transformation, and data propagation. The discussion of the pros and cons of these migration strategies will not only help the practitioner but hopefully will also stimulate the academic into doing prospective research.

**Proceedings of the 21st VLDB Conference
Zurich, Switzerland, 1995**

2. Survey of Database Migration Strategies

To apply progressive database technology, companies are faced with the challenge of migrating data and, in most cases, applications as well (Rodgers 1989). A few promising database migration strategies already exist which can protect the investment of present information systems and avoid the effort and risk involved in converting databases and applications. Figure 1 shows data and code conversion, language transformation and data propagation between two database management systems, A and B, with their respective query languages, AQL and BQL. Normally, A and B are two distinct database systems[1], e.g. hierarchical, network, relational or even object-oriented.

[1] If schema evolution and/or data replication are not supported by the database management systems, both systems A and B may be chosen from identical database technology. In this case, data and code conversion as well as data propagation are applicable. As an example, in relational technology, delta changes from a relation of system A could be propagated to the appropriate relation of system B, either synchronously or asynchronously.

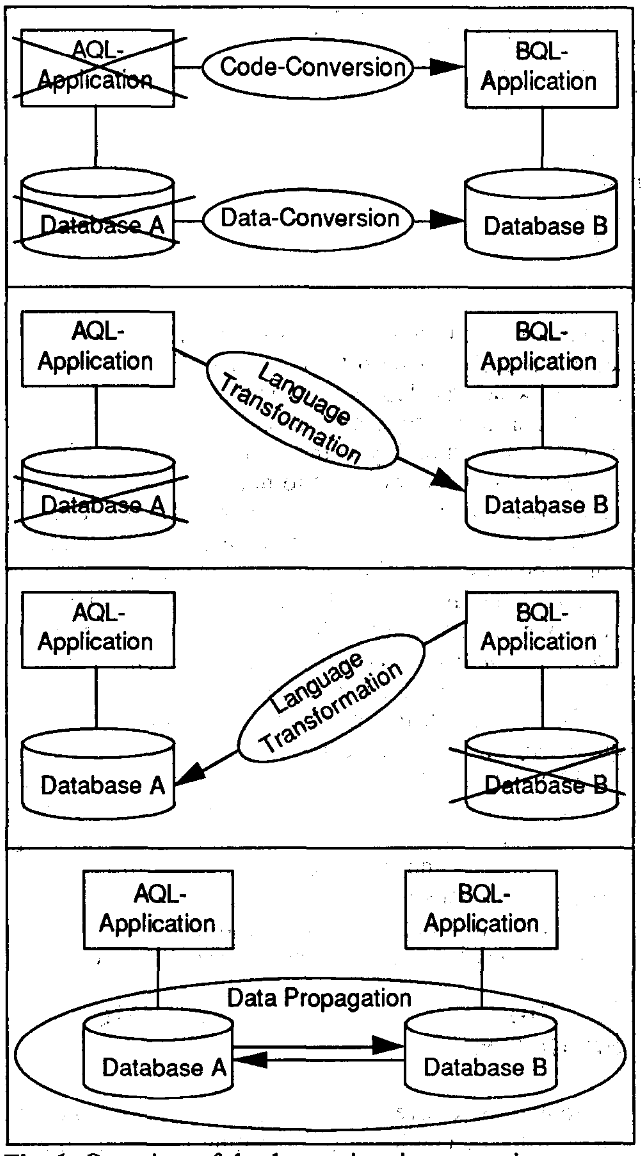

Fig. 1: Overview of database migration strategies

Data and code conversion for database application programs is a reasonable migration strategy, especially when applied to data which is accessed by a relatively small number of programs. First, the database itself has to be migrated from system A to system B. Second, database requests in application programs have to be inspected by a code-converter software package before calls in AQL can be replaced by new generation language statements in BQL. When source code conversion is applied to all application programs, the advantage of this migration strategy is that it needs only one single copy of data. However, converted source code is not very easy to maintain and has been known to exhibit performance difficulties.

Language transformation allows for the mapping of one database language to another and vice-versa, e.g. from AQL to BQL or from BQL to AQL. The direction of the language transformation is important. Mapping from AQL to BQL leads to the target database system B while the opposite direction keeps the existing database technology A. Both directions of language interfaces have proven to be very difficult and not applicable in general, and only partial solutions supporting a controlled and limited number of database commands are to be found on the market. Indeed, for very large databases, language transformation may not be adequate. In addition, unacceptable operational risks are associated with the necessary switch-over situation when using language transformation and where a large number of significant simultaneous changes are involved.

Data propagation aims to maintain the consistency between two or more data copies of different database systems by propagating only delta changes from one copy to the other. Forward data propagation enables automatic and selective transfer of changed data from one database generation A to another database generation B without having to convert existing applications. Reverse data propagation sends delta changes from the latest database generation B to A in order to maintain a company's investment in well-functioning application software. Data propagation allows for the performance of a stepwise migration of update programs but often takes years until it is completed. On the other hand, it avoids the efforts and risks involved in converting and interfering with well-tuned applications.

For all three migration strategies, commercial products or partial solutions can be found on the market which reflect the importance of database migration for the practitioner.

3. Steps for Data and Code Conversion

Instead of rewriting all or part of existing applications by hand, databases as well as database applications can be converted automatically by appropriate software tools (see, e.g. Gillenson 1990, Larson 1983, and Sockut 1985). To illustrate this approach, data and code conversion from record-based database systems (hierarchical, network) and languages (DL/1, CODASYL) to relational systems with the database language SQL will be described.

The first task to be solved is defining the conceptual schema of the target database system. In order to automate this process, general mapping types have to be applied.

General mapping types
Type 1 mapping brings a specific record type to one or several relations (see Figure 2). Type 1a is a one-to-one mapping between records and tuples, possibly restricted to specific fields or attributes. Type 1b separates the record type C in two or more relations based on a predicate. In practice, mapping of Type 1b is very helpful in order to allow a *redesign of existing hierarchical or network databases*. If, for instance, several entity types have been combined in one segment or record type for historical reasons, Type 1b may be used to satisfy the normal forms or to support a classification C1 and C2 of entity type C.

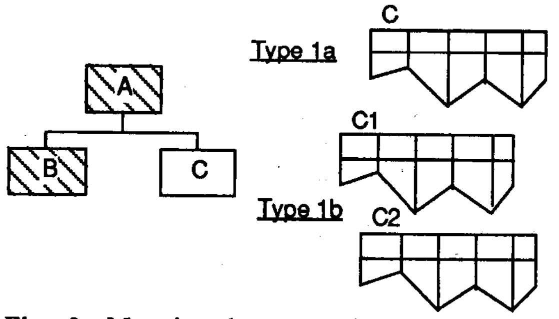

Fig. 2: Mapping between single record type and relation(s)

Type 2 is for mapping dependent record types to relations (see Figure 3). First, two record types, A and B, with a (1,c)-association are considered (c indicates conditional, i.e. c=0 or c=1). This means that for a specific record in A there exists, at most, one (c) in B, and every record in B has exactly one (1) corresponding in A. Normally, record type A is mapped to relation A (by applying mapping Type 1a of Figure 2) and record type B to relation B, where relation B includes a foreign key A# to reference relation A. Instead, by using mapping Type 2a of Figure 3, the two record types A and B are combined into one single relation A/B. Since not every record in A has a corresponding record in B, null values will be introduced in relation A/B.

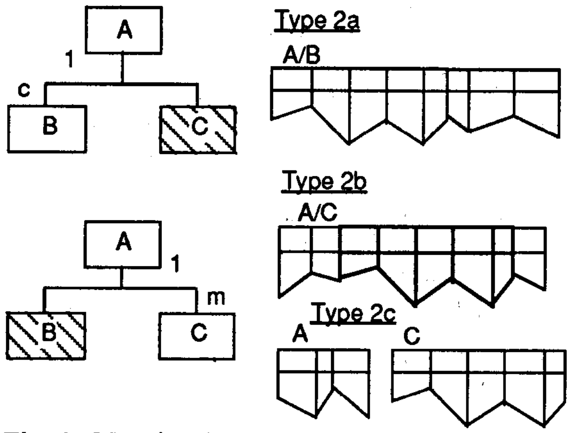

Fig. 3: Mapping between dependent record types and relation(s)

Although mapping Type 2a initially appears to be superfluous, it is necessary in practice. Very often, e.g. in hierarchical databases, segment extensions have been avoided in order to keep the application programs unchanged. Additional segment types have then been introduced as dependent segment types. Mapping Type 2a allows this *deficiency (inherited load) to be overcome* by combining two dependent record types from the same entity into one single relation.

Mapping Type 2b takes two record types with a hierarchical (1,m)-association and puts them into one nested relation (see Figure 3). If the relational database system does not support nested relations, record type A has to be mapped to relation A, and record type C to relation C including a reference through foreign key A#; this leads to mapping Type 2c.

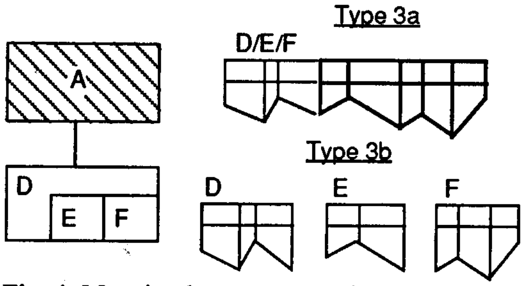

Fig. 4: Mapping between repeating group and relation(s)

Finally, mapping of Type 3 is necessary to transform *record types with repeating groups* into one nested relation (Type 3a) or to a set of relations (Type 3b) according to Figure 4. Repeating groups are very often used in hierarchical and network databases. With Type 3 they can be mapped into relations depending on the decision of the database administrator staff.

Some of these mapping types are supported by re-engineering tools. In the database area, the purpose of re-engineering tools is to recreate and enhance the specification of existing databases (Bachman 1989). Appropriate tools try to filter out the database schema information in such a way that the specification is independent of the database technology used in the original information system.

The BACHMAN toolset from Bachman Information Systems Inc., Cambridge with the DataAnalyst and the Database-Administrator components allow, for instance, reverse and forward database schema engineering for hierarchical IMS/DB and relational DB2 databases.

Most re-engineering tools in place today convert the data structure and schema information among different database technologies, i.e. between hierarchical, network or relational database systems. Because an inventory of a company's application network may consist of several hundred to a few thousand programs (see also Figure 8), the main task of database migration is to convert not only the data structure but also the application code.

Converting application code

In Figure 5, the process of code conversion for programs with a procedural database language to programs using the set-oriented language SQL is illustrated. The database description of the hierarchical or network database gives the starting point for the conversion process. The database administrator first chooses appropriate mapping types in order to produce the desired relational schema. An application of the migration tool not only produces the schema definition and stores it in the catalogue of the relational system, but also generates a corresponding SQL statement for every database call type (in DL/1 or CODASYL).

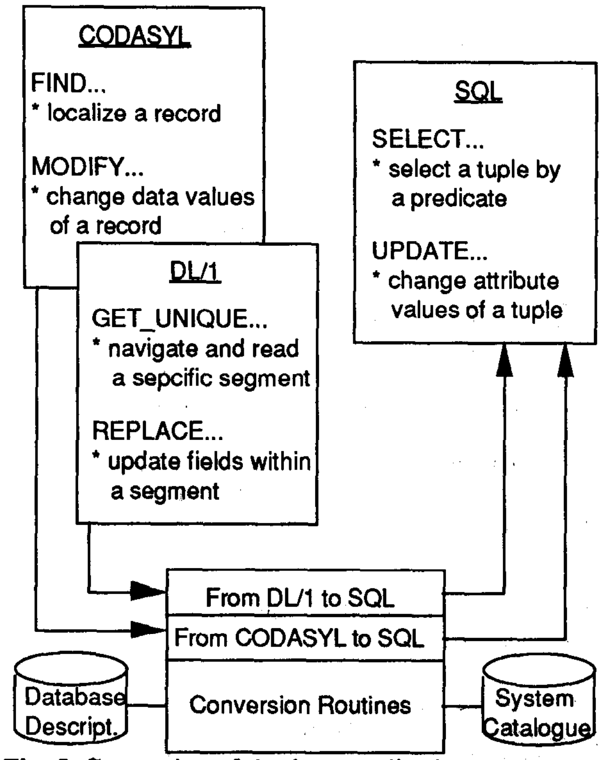

Fig. 5: Conversion of database application programs

Based on the mapping types, the converter software inspects the code of the record-based programs and replaces every database call with the corresponding SQL call. This process can be automated, as long as the record types in the source applications are not redefined individually and a set of type conversion routines are available (e.g., to convert basic data types and/or date formats). After having converted the application programs, function and stress tests have to be taken.

DL/1- or CODASYL-programs are record-based, allow navigation and perform a database request record by record. In contrast, a relational database system supports a set-oriented approach where sets of tuples can be performed with a single SQL statement. Most converter software tools, however, cannot profit from sets of tuples. In addition, physical aspects such as clustering, index definitions or access path selections are quite different for hierarchical, network and relational database systems. Therefore, performance-critical applications have to be tuned after the conversion process. In summary, data and code conversion is a practical migration strategy if used for non-critical and well-behaved applications.

Products from the SWS Software Services GmbH in Germany, such as HIREL (migration from IMS/DB to DB2 or SQL/DS) and IXREL (migration from IMS/DB to Unix-based relational database systems such as Informix) support data and code conversion.

4. Language Transformation Interfaces

According to Figure 1, a relational database should provide a procedural language interface (DL/1 or CODASYL) and a hierarchical or network database should allow for access and modification of data by an SQL interface (Date 1986). For instance, a procedural language interface for relational databases would keep all existing applications untouched and protect prior investment. At the same time, one could profit from the relational database technology without any restriction. The only task would be to unload the hierarchical or network databases and to reload the data into relations. Unfortunately, both language interfaces are cumbersome for the following reasons:

- A lot of data types with DL/1, CODASYL and SQL are not compatible. Even worse, different data types for one field or for overlapping fields work with hierarchical or network databases but have, for good reasons, no analogy in SQL.
- Specific data types such as DATE (date format for year, month, and day), TIME (timestamp given by hours, minutes, and seconds), NULL (null values for data which is not yet known) or VARCHAR (character string of variable length) have no equivalence in DL/1 or CODASYL and therefore have to be simulated.
- With the exception of the ORDER-BY-clause, SQL does not support alphabetical, chronological or physical order of attribute values. In addition, temporal support for databases still remains a requirement for most commercial products. At least, hierarchical and network database systems allow for navigation with search arguments, application of GET_NEXT commands and insertion of physical records by order relations such as FIRST, LAST, or HERE.
- Some important SQL commands cannot be supported directly by a language interface for hierarchical and network database systems. Consider the following examples: predicates with the logical OR; comparison operators other than EQUAL and NOT EQUAL; LIKE and IS NULL; IN or EXISTS in a subquery; or GROUP BY and HAVING clauses.
- Integrity constraints and processing rules with DL/1 and CODASYL are quite different compared to referential integrity or trigger mechanisms in relational technology. For instance, the SQL commands for restricted deletion, cascaded deletion, or deletion with nullify are not exactly the same in DL/1 or CODASYL.
- External views of database schemes are different in definition and application in hierarchical, network, and relational technology. SQL, for instance, has restrictions for updating with views.
- If SQL commands are embedded in a programming language, the cursor concept is used in order to perform record by record. DL/1 or CODASYL give a direct control of hierarchical or network data structure as stated above. On the other hand, most relational database systems still fail to handle repeating groups or

nested relations, transitive closure, and complex objects which are in part possible with hierarchical or network databases.

The list of difficulties for both language interfaces could be extended by other examples, with the result that the transformation of language commands between non-relational and relational database languages cannot be solved entirely. Therefore, only partial solutions are currently available on the market.

As examples of the language transformation approach, the CODASYL-like database system UDS from Siemens supports an SQL interface called UDS/SQL, or the ADABAS-system from Software AG offers a family of language interfaces to different database systems, e.g. the ADABAS DL/1-Bridge.

5. Forward and Reverse Data Propagation

Data propagation maintains the consistency between two databases by propagating the updates (delta changes) of one database to the other. Data propagation can be implemented with application program logic. In practice, such an approach is both work and maintenance intensive, can be very error-prone and may require changes to numerous application programs. It is therefore preferable to propagate data using a generalized software package (Meier et al. 1994) which keeps data consistent without modifying existing application programs.

Data propagation enables the coexistence of applications from heterogeneous database management systems accessing the same data. For obvious reasons when considering market suitability, data propagation tools between hierarchical, network and relational systems are necessary. *Forward data propagation* is applicable if relational database technology is to be used for new applications or decision-support systems without converting existing applications. *Reverse data propagation* from relational to hierarchical or network allows for a smooth migration path by reflecting delta changes from SQL applications back to pre-existing applications. With *asynchronous data propagation*, changes are applied later, i.e. not in the same unit of work as the original update calls. It is typically used when decision-support applications need point-in-time data.

Forward, as well as reverse data propagation, work in three phases namely design, extract/load and propagation (see Figure 6). For simplicity, the three phases for forward propagation are described.

Design phase
The database administrator has first to specify which databases, record types and fields should be propagated and how they should be mapped to corresponding relations, tuples and attribute. The result of the mapping definitions will be stored in a mapping directory. For this definition phase, general mapping types are provided in order to map data consistently from non-relational to relational and vice-versa. It is important to note that the same mapping types 1 to 3 (see Figures 2, 3, and 4) may be used for forward as well as for reverse data propagation, if fully concatenated keys are enclosed with the relations.

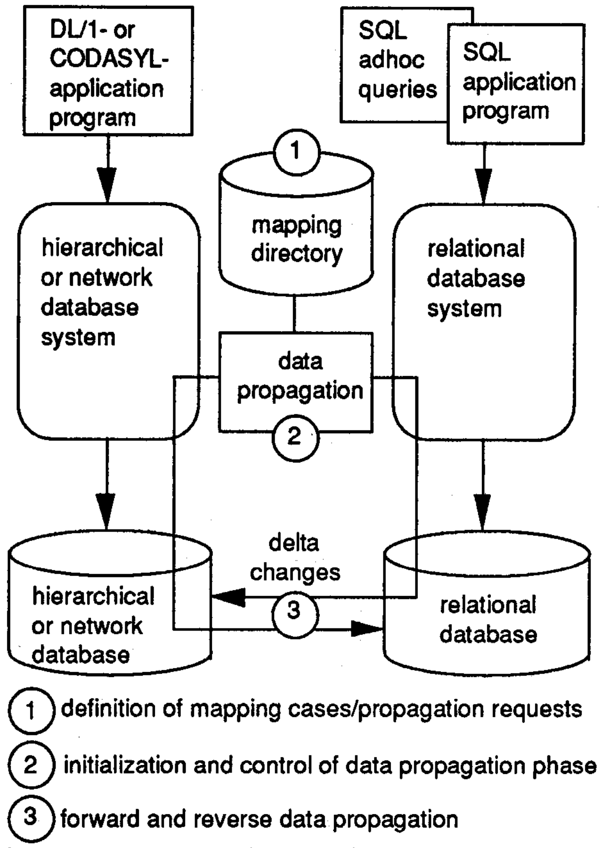
DL/1- or CODASYL-application program
SQL adhoc queries
SQL application program
1
mapping directory
hierarchical or network database system
relational database system
data propagation
2
delta changes
3
hierarchical or network database
relational database
1 definition of mapping cases/propagation requests
2 initialization and control of data propagation phase
3 forward and reverse data propagation

Fig. 6: Data propagation provides system-controlled consistency

After having selected the hierarchical or network databases, mapping types and corresponding relations, the database administrator has to inform the system of these definitions by one or more propagation requests. A *propagation request* has to be specified for each entity type in order to declare the propagation type (forward or reverse propagation), record type and corresponding relational structure, and the desired mapping types. It should also be possible to specify multiple sets of propagation requests. In a computer configuration for example, one may wish to propagate records to a first set of relations used for operational applications; and/or to propagate some of the same record types using a second set of propagation requests to a second set of relations used for decision-support applications.

The major task of the design phase is to generate and maintain the propagation request definitions in the mapping directory. This can be done by a mapping verification and generation utility which acts as a driver, validates the propagation requests, stores the mapping definitions into the directory and generates SQL update modules. For performance reasons, it is very important that, for each request and every call type, the corresponding SQL update module is generated in advance.

Extract and load phase
The data extract and load phase have to *synchronize the chosen databases* of the hierarchical or network and the relational system prior to propagating changes. To start this phase, the database administrator has to run a specific status change utility. This sets those databases involved to read-only status and ensures that all subsystems have released their update authorization. After having extracted the designated databases by an extract utility and having generated the input for the load utility, this data is then loaded into the appropriate database.

Propagation phase
Once the mapping definitions and initial extract and load phase have been completed, the hierarchical or network databases will be ready for updating, and data propagation using an SQL update program can begin.

Figure 6 focuses on synchronous data propagation because some system components act differently in the asynchronous case (Meier et al. 1994). If the database description of a hierarchical or network database includes a data propagation exit, a data capture function presents all changed records to the appropriate SQL update program. While processing the first update to a specific record type of the executing application program (i.e. insert, replace, or delete), the SQL update program receives mapping information. After having determined the current propagation status of the desired propagation requests, it then evokes the relational database system for updating the corresponding relations.

The data propagation software package has to guarantee system-controlled consistency. If propagation fails following a specific propagation request, the SQL update program should not only provide diagnosis information but has to back out all updates performed by the updating program since the previous commit-point.

With data propagation, a stepwise migration for database management systems can be performed. Forward propagation is used while read-only applications may access the data copy in the relational databases. If read and write on the relational system are necessary and the same data is affected by existing applications, reverse propagation is recommended. Once data propagation is established, it is possible to convert one application program at a time. This can last for years but can conveniently be done when an application area needs a major redesign. Over time, data propagation allows an *evolutionary and low-risk migration* of applications to new generation database technology.

The DataRefresher and DataPropagator MVS/ESA product family from IBM, for instance, allows forward and reverse data propagation between the hierarchical database system IMS/DB and the relational DB2, synchronously or asynchronously. The InfoReplicator and InfoPropagator products from Platinum Technology perform a total refresh or incremental data changes of IMS and DB2 databases.

6. Planning a Database Migration Process

The main trend of database migration is to take increasing advantage of current relational database technology (see Figure 7). An empirical study for Europe, carried out by the Plenum Institute and the University of Würzburg (Hildebrand 1992), illustrates that today's data is mainly stored in file systems and partly kept in relational or hierarchical database systems. However, in the next few years, relational systems will become the predominant storage technology.

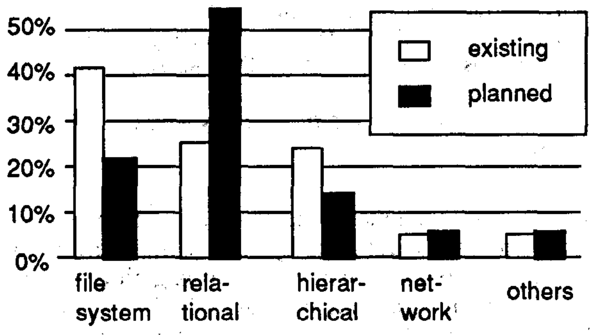

Fig. 7. Relevance of database technology in practice

Database migration is not a single task but a process which can last several months or even a few years. The reason for this time- and cost-consuming process can be seen by some relevant parameters reflecting the investigations companies have put into their information systems. Figure 8 illustrates typical figures of the workload and the complexity of a possible database migration process. Here, mid-range companies are assumed to be those institutions with at least a thousand employees, and large companies are those which provide employment for more than ten thousand.

The migration parameters show that company investments in their information and database systems may reach several million dollars. Therefore, a database migration path has to follow some rules in order to succeed.

1. Study the key business areas of the corporate strategy and derive corresponding data architecture.
2. Define the mapping between the installed databases and the databases which support the corporate-wide data model.
3. Choose the most appropriate migration strategies from data and code conversion, database language transformation or data propagation.
4. Ask for a steering committee to accept the business plan for database migration with time-table, costs, human resources and skills needed.

There is no doubt, that database migration is not only a technical but also an organizational challenge. It does not make sense to convert data structures and business functions into a new technology platform without *considering the key success factors of the company*. This

migration parameters	mid-range company	large company
# of database products	4 - 6	10 - 12
data volume in Gigabytes	≈ 100	> 1'000
# of record types	≈ 300	> 1'000
# of data elements	≈ 5'000	> 10'000
# of application programs	≈ 1'000	> 10'000
# of transactions per second	50 - 80	> 100

Fig. 8: Important parameters for database migration

leads to the discussion of a corporate-wide data architecture. The core of that architecture is a data model which is independent of existing hardware or software solutions yet is based on the business plan of the company.

7. Suggestions for Future Research

Database technology remains the basis for developing new applications and for optimizing business processes. As relational, object-oriented and expert database products come into increasing demand from both the industrial and service sectors, database migration will need to evolve. The following key areas should therefore be extensively studied, both by practitioners and researchers:

Data and application re-engineering will become a hot issue in the second half of the ninety's, along with the business process redesigns of competitive companies. If a company has invested hundreds of person years for its applications, it doesn't have the time, money and human resource to redevelop these applications from scratch. On the other hand, time to market, customer orientation and core business concentration will force companies to invest in future database and information technology.

As mentioned above, database migration is not only a technical task but also an organizational challenge for most companies. Therefore, the *design of a corporate-wide data architecture becomes a strategic necessity* before new database or information technology can be implemented. Most companies, however, still fail to develop and maintain a corporate-wide data model derived from their business plan.

More specifically, *schema evolution is a database research domain still in its infancy*. Even with object-oriented technology, little progress has been seen. With relational database technology, at least new relations can be created, existing ones dropped, and new attributes added. The semantics of a class hierarchy in object-oriented technology however is more complicated and not yet successfully adopted for schema evolution.

Finally, *data replication and data propagation should become common functions* of commercial database systems. While data replication has been studied in depth, this is not the case with data propagation. In practice, huge amounts of unstructured and badly documented data is kept on different platforms. Most application programmers, specialists of information centers, as well as field experts and casual users of a company are totally lost when database queries, reports, or documents need to be generated for business decisions. Extracting and copying data without strong system control of periodicity and data semantics leads to complete data chaos.

Acknowledgements: Most of this work is based on experience gained from the migration of all DATACOM/DB-applications to the relational DB2 at CSS Insurance Company as well as from the design and development of the software package DataPropagator MVS/ESA (which has become an IBM program product) at Swiss Bank Corporation. I would like to thank all my professional colleagues for the successful database migration phases in practice. For obvious reasons, I can mention only a few by name: I send special thanks to Peter Bachmann, Urs Bebler, Camillo Ceppi, Rolf Dippold, Bob Eckerlin, Toni Kreienbühl and Pete Uhrowczik. Peter Lockemann and Rudolf Marty encouraged me to submit this paper to the industrial track of VLDB'95. I am also grateful to an unknown referee of the VLDB program committee and to Liz Gosselin who was not tired to reshape my text.

Literature:

Bachman, C. W.: A Personal Chronicle - Creating Better Information Systems, with Some Guiding Principles. IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 1, 1989, pp. 17-31.

Date, C. J.: Why Is It So Difficult to Provide a Relational Interface to IMS. In: Date, C. J.: Relational Database - Selected Writings. Addison-Wesley, 1986, pp. 241-257.

Hildebrand, K.: Information Management - Status Quo and Future Issues (in German). Wirtschafts-informatik, Vol. 34, No. 5, 1992, pp. 465-471.

Gillenson, M. L.: Physical Design Equivalencies in Database Conversion. Communications of the ACM, Vol. 33, 1990, pp. 120-131.

Larson, J. A.: Bridging the Gap between Network and Relational Database Management Systems. IEEE Computer, Vol. 16, 1983, pp. 82-92.

Meier, A., Dippold, R., Mercerat, J., Muriset, A., Untersinger, J.-C., Eckerlin, R., Ferrara, F.: Hierarchical to Relational Database Migration. IEEE Software, Vol. 11, No. 3, 1994, pp. 21-27.

Rodgers, J.: Database Coexistence - Requirements and Strategies. Proc. of the 18th Mini-G.U.I.D.E. Conference, Florence 1989, pp. 117-142.

Sockut, G. H.: A Framework for Logical-level Changes within Database Systems. IEEE Computer, Vol. 18, 1985, pp. 9-27.