CASE Tools for Database Engineering

Jean-Luc Hainaut  
University of Namur, Belgium

Jean Henrard  
REVER S.A., Belgium

Jean-Marc Hick  
REVER S.A., Belgium

Didier Roland  
REVER S.A., Belgium

Vincent Englebert  
University of Namur, Belgium

INTRODUCTION

Designing and implementing a database comprising a few tables require a level of expertise that is readily found among most experienced users, provided they are somewhat keen on office productivity tools. Playing a dozen of hours with Microsoft Access should give clever and motivated users sufficient feeling and technical skill to develop small workable databases.

However, things quickly get harder for larger databases, particularly when they have to be integrated in different environments and to meet the needs of several applications. Very large and complex databases often include distributed heterogeneous components that have been developed by dozens of more or less skilled developers at different times, with different technologies and through different methodologies (or absence thereof). Mastering, integrating, maintaining, renovating, evolving, and migrating such complex systems and developing new components for them are highly complex engineering processes that are far beyond the capacity of individuals, whatever their experience and skill. Such problems cannot be addressed without the help of robust methodologies, supported by powerful tools. The following conservative figures illustrate the need for a disciplined, tool-based, approach to database engineering. Considering that (1) each conceptual entity type should ideally be accompanied by a two-page documentation giving its precise semantics and properties, (2) each entity type, together with its attributes and relationship types, translates into two tables on average, and (3) each table is coded by 150 lines of SQL statements, including structural and active components; a standard conceptual schema comprising 1,000 entity types will be described by a 2,000-page documentation and will produce 300,000 lines of SQL-DDL code. Since database engineering is a special branch of software engineering, the concept of database-centered computer-aided software engineering tools—DB CASE tools for short—appears quite naturally. As a matter of fact, most CASE tools include a strong component intended to help developers design and produce (mainly relational) databases.

This paper discusses some important aspects and functions of DB CASE tools through the description of DB-MAIN, a programmable general-purpose CASE tool that has been designed to support most complex engineering processes for large databases. In particular, it includes, besides standard functions, powerful capabilities for reverse engineering and federated database building.

BACKGROUND: DATABASE ENGINEERING REQUIREMENTS

Database engineering mainly deals with data structures, data and code related to one or several databases, all of which must be adequately documented in order to be exploited, maintained, and modified in a reliable and efficient way all along the lifetime of these databases. The core of the documentation of a database is a hierarchy of schemas, each of which describes in a precise way its information/data structures at a definite level of abstraction. Most design approaches agree on a four-level architecture comprising a conceptual schema, a logical schema, a physical schema, and code (Batini, Ceri, & Navathe, 1992). The conceptual schema describes the information structures that are of interest in
the application domain in a formalism that is technology-independent, such as the entity-relationship model. The logical schema is the translation of the conceptual schema according to the model of a family of DBMSs. The physical schema adds to the logical structures DBMS-specific technical constructs intended to give the database such properties as efficiency and reliability. The code translates the physical schema into a compilable DDL program.

Once a database has been deployed in its production environment, it enters a maintenance and evolution cycle, through which its schema (and consequently its contents) is continuously modified to follow the evolution of functional, organizational, and technological requirements. Both modern and legacy databases are migrated to more advanced platforms; they are integrated to form federations, generally through wrapping and mediation techniques; and they are interfaced with the Web and feed data warehouses. These processes all require the availability of a precise, up-to-date documentation of the concerned databases. Whenever this documentation is missing, the databases must be redocumented, a process called reverse engineering.

DB CASE tools are to support these processes, ranging from traditional analysis and design to reverse engineering and federation. They collect, store, and manage not only schemas and their interrelationships (ensuring traceability), but also pieces of code and multimedia documentation related to each step of the database life cycle.

A large organization can include several hundreds of databases, each described by several schemas. Each schema appears to be a graph, made up of hundreds of thousands of vertexes (abstracting entity types, attributes, relationship types and roles, constraints, etc.) This size factor implies specific representation modes and powerful tools to consult, analyze, and transform schemas. Large collections of databases require several design and development teams that call for collaborative work support. Finally, no two organizations are alike and share the same methodological culture, so CASE tool customization and extendability are often highly desirable.

In the next section, we will discuss some important functions of DB CASE tools that derive from these requirements.

FUNCTIONS OF CASE TOOLS

Schema Management

This is the basic goal of any CASE tool, namely, allowing developers to enter schemas, including through graphical interfaces; to store them into a schema database, called a repository or encyclopedia; to consult, browse through, and query them; to view them through adequate visualization modes; to output printable documentation; and to exchange specifications with other CASE tools, notably through interchange formats such as XMI (Object Management Group, 2003).

Schemas are expressed into a specific model, depending on the abstraction level and the concerned technology. At the conceptual level, most tools rely on some sort of entity-relationship model. The UML class diagrams belong to this family and are gaining increasing popularity. However, their weaknesses and idiosyncrasies make them a poor support to express large and complex schemas, specially in nonstandard processes such as reverse engineering.

DB-MAIN is based on the wide-spectrum model GER (generic Entity-relationship) that encompasses most commonly used models, whatever their abstraction level and their technology (Hainaut, 1996). An operational model M is defined by a specialization mechanism through which the constructs of the GER pertinent in M are selected and renamed, and the assembly rules for valid M schemas are stated. The GER includes such constructs as entity types, domains, attributes, relationship types, methods, and a large variety of integrity constraints. It also comprises logical and technical constructs, such as foreign keys, access paths, access keys, and files. The tool includes the rules for the most popular models such as Entity-relationship, extended UML class diagrams (DBUML), relational, object-oriented, CODASYL, COBOL files, and XML (DTD and Schemas). Additional personalized models can be defined thanks to a method engineering environment. Figure 1 shows a hybrid schema that includes constructs from several technologies and several abstraction levels. Such a schema is usual in ongoing reverse engineering projects.

DB-MAIN offers several ways to query and browse through a schema. In addition to six customizable views of schemas, two of them being illustrated in Figure 1, objects can be selected by name patterns, keywords in object annotations, and structural patterns. Selected objects can be marked in order to be retrieved later or to be passed as arguments to further processes.

Schema Checking and Validation

Depending on the methodological standard adopted or the target technology, schemas are to meet specific requirements in order to be valid. In particular, a schema can be analyzed against a definite model to check whether it complies with it. For instance, a conceptual schema that has been transformed into XML structures should be validated against the XML schema model before passing it to the code generator.