CASE Tools for Database Engineering

C

Jean-Luc Hainaut
University of Namur, Belgium

Jean-Marc Hick
REVER S.A., Belgium

Didier Roland
REVER S.A., Belgium

Jean Henrard
REVER S.A., Belgium

Vincent Englebert
University of Namur, Belgium

1. 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, 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, that (2) each entity type, together with its attributes and relationship types translates into two tables on average, and that (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 databases building.

2. BACKGROUND: DATABASE ENGINEERING REQUIREMENTS

Database engineering mainly deals with data structures, data and code related to one or several databases, all of which 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, 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 DBMS. 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, 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 inter-relationships (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 requires several design and development teams that call for collaborative work support. Finally, no two organizations are alike and share the same methodological culture, so that CASE tool customization and extendability are often highly desirable.

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

3. FUNCTIONS OF CASE TOOLS

Schema management

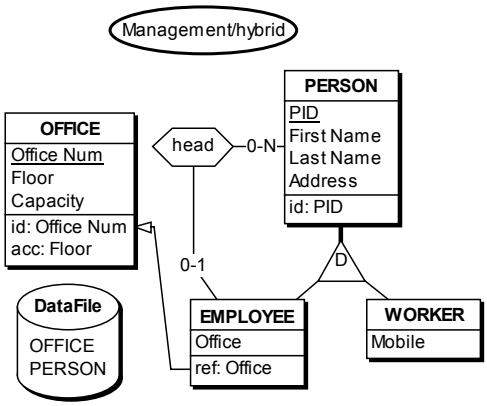
This is the basic goal of any CASE tool, namely allowing developers to enter schemas, including through graphical interfaces, to store them into a schema database, called *repository* or encyclopedia, to consult, browse through, and query them, to view them through adequate visualization modes, to output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

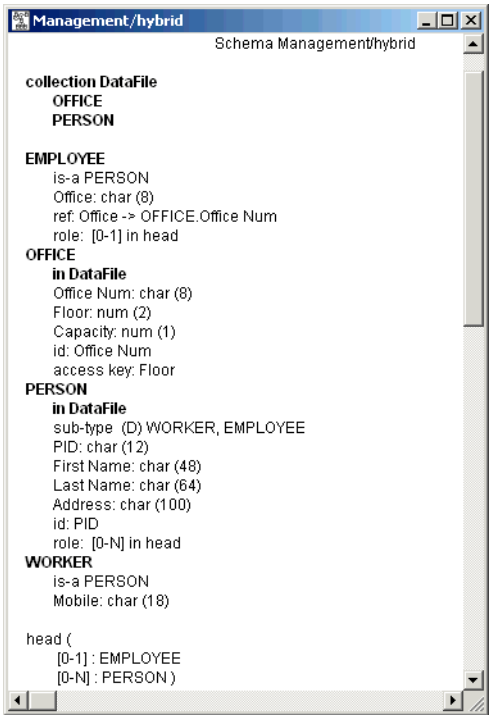
Schemas are expressed into a specific model, depending on the abstraction level and the concerned technology. At the conceptual level, most tools rely on some sort of Entity-relationship model. The UML class diagrams belong to this family and are gaining increasing popularity. However, their weaknesses and idiosynchrasies 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 (DB-UML), 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.

Figure 1 *- Graphical and hypertext views of a hybrid DB-MAIN schema. They show examples of entity types, IS-A hierarchy, attributes, relationship types and attributes, but also logical constructs such as a foreign key as well as physical constructs (data file and access key).*

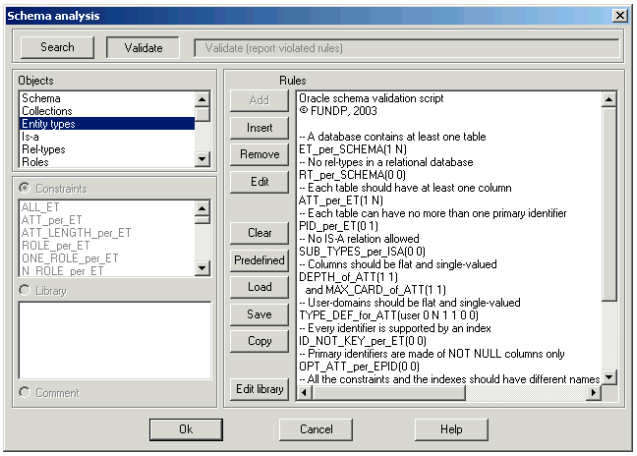




Schema checking and validation

Depending on the methodological standard adopted or the target technology, schemas are to meet specific requirements in order to be valid. In particuler, a schema can be analyzed against a definite model to check whether it complies with it.

Figure 2 *- The main screen of the Schema Analyzer of DB-MAIN. The script shown is a fragment of the Oracle model.*



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.

DB-MAIN includes a Schema Analyzer (Figure 2) that provides three main functions. First, it allows the designer to build, save and reuse specific models as a set of rules defined by structural predicates. Secondly, its engine can analyze the current schema against a specific model, and identify (report, select or mark) the schema objects that violate any rule of this model. Conversely, it can identify all the objects of the current schema that meet at least one of the rules of a model. DB-MAIN includes 12 predefined models, ranging from COBOL files structures and XML physical models to ERA and DB-UML conceptual models.

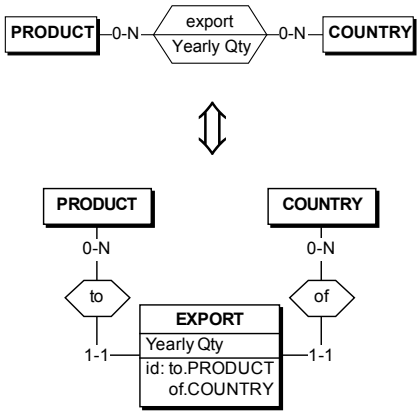
Schema transformation

Most database engineering processes intrinsically consist in producing specifications from existing ones, an activity that can be described as schema transformations [Rosenthal, 1994] [van Bommel, 2004] [Hainaut, 2005]. For instance, normalizing a conceptual schema, producing an SQL database or XML structures from a conceptual schema, optimizing a physical schema, or reverse

engineering standard files and CODASYL structures can be described mostly as schema transformations. In [van Bommel, 2005], the first author has shown that the whole database design process, together with other related activities, can be described as a chain of schema transformations. Schema transformations are essential to formally define forward et backward mappings between schemas, and particularly between conceptual structures and DBMS constructs, which ensures the traceability of the database engineering processes.

DB-MAIN includes a toolset of about 30 basic schema transformations together with facilities to build higher-level transformations. The basic transformations have been selected in order to cover the needs of most database engineering activities, including those mentioned above. In particular, they provide operators to materialize IS-A relations, to transform entity types into relationship types or into attributes, and to split/merge entity types. Attributes can be transformed into entity types and relationship types. Compound attributes can be disagregated, while multivalued attributes can also be replaced with serial or concatenated attributes. Non-set attributes, such as bags, lists and arrays, can be transformed into pure sets. Most of them are semantics-preserving, i.e., both source and resulting schemas describe exactly the same information (Figure 3).

Figure 3 *- A typical schema transformation that replaces a many-to-many relationship type with an entity type and one-to-many relationship types*



DB-MAIN introduces the concept of *predicate-driven transformation*, noted T(p), where T is any basic transformation and p a structural predicate as described in the previous section. Let us consider,

- RT_into_ET that denotes the transformation of a relationship type into an entity type (Figure 3);
- expression ATT_per_RT(i j), a predicate that, when applied to a relationship type, checks whether the number of its attributes is between i and j;
- expression ROLE_per_RT(i j), a predicate that, when applied to a relationship type, checks whether the number of its roles is between i and j.

Then, the expression

RT_into_ET(ATT_per_RT(1 N) or ROLE_per_RT(3 N))

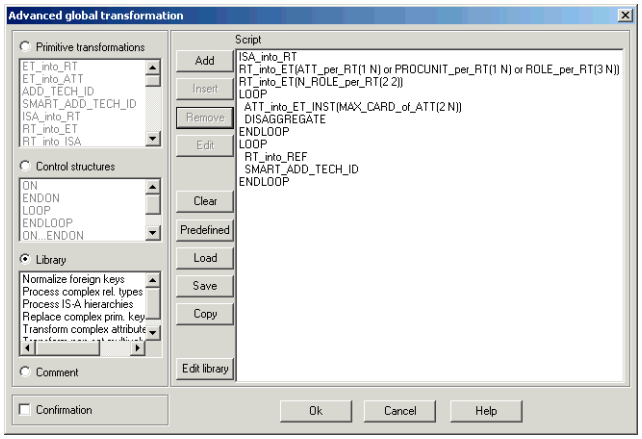
defines a predicate-driven transformation the effect of which, when executed on the current schema, replaces all complex relationship types (i.e., those which have attributes or at least 3 roles) by entity types.

The most powerful application of this concept is the *model-driven transformation*. It consists of a chain of, possibly iterative, predicate-driven transformations that is intended to transform any schema into a schema that complies with a definite model.

Both predicate-driven and model-driven transformations can be built, executed, saved and reused through the Advanced Transformation Assistant of DB-MAIN. A dozen of popular model-based predefined transformations are provided to derive relational, XML, UML schemas, but also to extract conceptual schemas from DBMS schemas such as CODASYL and standard file managers. Figure 4 shows a fragment of a script that expresses an RDBMS transformation. Though strongly simplified (the complete script comprises about 20 predicate-driven transformations), this fragment includes the most important operators to transform conceptual schemas into relational structures:

- materializing IS-A relations into one-to-one relationship types (ISA_into_RT),
- transforming complex relationship types (i.e., with attributes, or with methods, or with at least 3 roles) into entity types (RT_into_ET),
- same for many-to-many relationship types (with 2 *many* roles),
- transforming multivalued attributes (the maximum cardinality of which is at least 2) into entity types (ATT_into_ET),
- decomposing compound attributes (DISAG GREGATE),
- applying (LOOP) the later two until everything has been transformed,
- expressing remaining relationship types into referential attributes, i.e., foreign keys (RT_ into_REF),
- where the latter fails, adding a technical primary key to the concerned entity type (SMART_ ADD_TECH_ID),
- applying (LOOP) the later two until everything has been transformed.

***Figure 4** - The main screen of the Schema Transformation Assistant of DB-MAIN. The script shown is a fragment of the ERA-to-relational model transformation.*



Code generation

A schema that complies with the physical model of a DBMS can be transformed into a DDL script the execution of which will create the database structures. All CASE tools include generators for a wide list of SQL DBMS. Theoretically, this should be straightforward since basic generators are fairly easy to develop and since most DBMS are SQL2 compliant. The hard problem is to translate all the constructs of the conceptual schema into a poor DDL. Not only this means that the logical and physical schemas express these constructs, but also that the DDL generator is able to compensate for the weaknesses of the target DDL. For instance, IS-A hierarchies are not correctly translated by most popular SQL generators, and many useful constraints are ignored in the generation process. Consequently, the developer has to manually modify the DDL code to translate the discarded constructs, thus breaking the traceability chain between the conceptual schema and the SQL code.

DB-MAIN offers a collection of basic generators for SQL2 (including Oracle, Access, MySQL and InterBase), COBOL files, CODASYL (IDS2) and XML (DTD and Schema). It also includes a parametric SQL generator that takes in charge all the IS-A patterns as well as a large set of integrity constraints. The constructs that cannot be explicitly declared in SQL will be controlled through check predicates, views with check option, triggers or stored procedures.

Some CASE tools also provide generators for DB-related code such as J2EE or .NET components, wrappers, migrators, active components for special-purpose databases or GUI interface to databases.

Reverse engineering

The main goal of this process is to rebuild the logical and conceptual schemas of a legacy database the documentation of which is missing or outdated. Since it has been recognized that the availability of these schemas is an absolute prerequisite in database maintenance, evolution, migration or federation, reverse engineering has gained increasing interest during the last decade. Most CASE tools are able to extract a graphical physical schema from the SQL code or from system catalog tables. They also provide some elementary transformations to produce a basic conceptual schema, mainly by replacing foreign keys by relationship types. However, such an approach falls short for most actual databases. Indeed, it is based on a oversimplistic hypothesis according to which the conceptual schema has been completely expressed through explicitly declared SQL constructs, and, conversely, the conceptual schema only includes constructs that can be declared through SQL statements.

Unfortunately, most existing databases do not meet this requirement. On the one hand, they often use legacy technologies, such as RPG, CODASYL, IMS, IMAGE or COBOL files, for which few robust extractors have been developed. On the other hand, many constructs and constraints have not been translated into DDL code, but, rather coded in the application programs, notably in data validation sections scattered through millions of lines of code. Discovering them involves in-depth procedural code analysis that are out of the scope of standard text and character stream analyzers. Finally, many database structures are not straighforward translation of a clean conceptual schema, but include complex optimization patterns that often are difficult to identify and interpret.

DB-MAIN [Hainaut, Englebert, Henrard, Hick & Roland,1996] includes a powerful toolset for large database reverse engineering. In particular it offers DDL code extractors for SQL, RPG, IMS, CODASYL (IDMS and IDS2), COBOL files and XML, programmable text analyzers to identify programming *clichés*, dependency and dataflow graphs analyzers and a sophisticated program slicer to identify program sections that process database components. It also includes specific assistants to discover implicit foreigns keys and scripts to analyze large schemas and to extract conceptual structures from logical schemas. The extensibility services of DB-MAIN make it possible to develop additional tools to cope with new problems such as unknown legacy technologies.

Meta-CASEs

Strictly speaking, a meta-CASE is an environment in which one can describe and generate customized CASE tools for a specific engineering area [Kelly, 1996] [Englebert, 1999]. However, most CASE tools offer some

restricted capabilities for developing additional specific, user defined, functions.

DB-MAIN provides developers and method engineers with five meta-level extensibility services.

1. Method fragments (models and model-based transformations) can be developed via the scripting facility of the Schema Analysis and Global Transformation assistants.
2. The repository can be extended by associating user-defined meta-properties with all the object types.
3. New processors and functions can be developed either in Java or through the 4GL Voyager 2.
4. After and before triggers can be defined for all the repository modification and transformation primitives.
5. MDL, a method engineering environment allows companies to develop specific methodologies [Roland, Hainaut, Hick, Henrard, & Englebert, 2000]. A method comprises specific models, product types and engineering processes defined by their strategies. The method engine of DB-MAIN guarantees the strict application of the chosen methodology. It also ensures complete traceability thanks to its history manager.

For instance, customized versions of DB-MAIN have been developed for datawarehouse design, database evolution, database reverse engineering, active database generation, XML engineering, federated database development and generation, and temporal database design and generation.

4. FUTURE TRENDS

Despite much effort from laboratories and industry (ISDOS, one of the first CASE tool was developed by Teichrow and Hershey in 1977), CASE tools permeate more slowly than expected among practitioners. In their in-depth analysis, Lundell and Lings (2004) identify three factors that can explain this phenomenon, but fails to spot the poor quality of the artefacts and code that CASE tools produce from users specifications.

Future tools are expected to provide better code generators, tools interoperability, notably through the CDIF [ISO, 2000] and XMI standards, intelligent assistance and support for maintenance and evolution. Inter-model synchronization and transformational approaches will be better supported, as witnessed by the increasing popularity of the OMG MDA.

5. CONCLUSION

Any attempt to write a stable state of the art in database CASE technology would be hopeless. Indeed, this domain is highly volatile (Lundell, 2004) and is the subject of rapid merging and buying, so that even the name of well established CASE tools can change in a short time.

Nevertheless, we can mention, among the main tools at the present time, *Rose* and *XDE* from Rational, *Together* from Borland, *PowerDesigner* from Sybase, *ERwin* from Computer Associates, *MEGA Database* from MEGA International, *MetaEdit+*, a meta-CASE from MetaCase, *Designer/2000* and *Developer/2000* from Oracle and *Silverrun* from Magna Solutions. Interested readers can also consult the site http://www.qucis.queensu.ca/Software-Engineering/tools.html. Information on DB-MAIN can be found at http://www.info.fundp.ac.be/libd. The free Education edition can also be downloaded from this address.

6. REFERENCES

Batini, C., Ceri, S. et Navathe, S., B. (1992). *Conceptual Database Design - An Entity-Relationship Approach*, Benjamin/Cummings.

Englebert, V. and Hainaut, J-L. (1999). DB-MAIN: A Next Generation Meta-CASE, *Information Systems Journal*, 24(2), Elsevier.

Hainaut, J-L. (1996). Specification preservation in schema transformations - application to semantics and statistics, *Data & Knowledge Engineering*, 11(1).

Hainaut, J-L, Englebert, V., Henrard, J., Hick J-M., and Roland, D. (1996b). Database Reverse Engineering : from Requirements to CASE tools. *Journal of Automated Software Engineering*, 3(1).

Hainaut, J-L. (2005). Transformation-based Database Engineering, *in this book*.

ISO (2000). Information technology - CDIF framework - Part 1: Overview, ISO/IEC FDIS 15474-1, Final Draft, ISO/IEC JTC 1/SC7/WG11, 2000-02-14.

Kelly, S., Lyytinen, K. and Rossi, M. (1996). MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors, *Proceedings of CAiSE'96*. Lecture notes in computer science: Vol. 1080 Springer-Verlag, Berlin/Heidelberg, Germany, pp. 1-21.

Lundell, B. and Lings, L. (2004). Changing perceptions of CASE technology, *Journal of Systems and Software*, 72(2),Elsevier.

OMG (2003). XML Metadata Interchange (XMI) - Specification, v2.0, May,2003, OMG, (http://www.omg.org/cgi-bin/doc?formal/2003-05-02; September 2004)

Roland, D., Hainaut, J-L., Hick, J-M., Henrard, J. and Englebert, V. (2000), Database Engineering Processes with DB-MAIN, in *Proc. of the 8th European Conference on*

Information Systems (ECIS2000), Vol. 2, Vienna University, Austria, pp. 244-251.

Rosenthal, A. and Reiner, D. (1994). Tools and Transformations - Rigourous and Otherwise - for Practical Database Design, *ACM TODS*, 19(2).

Teichrow, D. and Hershey, E. (1977). PSL/PSA: a computer aided technique for structured documentation and analysis of imformation processing systems, *IEEE TOSE* , 3(1).

van Bommel, P., (ed.) (2005). *Transformation of Knowledge, Information and Data: Theory and Applications*, IDEA Group Pub.

7. KEY TERMS

CASE tool: Software tools that help software designers and developers specify, generate and maintain some or all software components of an application. Most CASE tools provide functions to allow developers to draw database schemas and to generate the correspondant DDL code.

Conceptual schema: A structured technology-independent description of the information about an application domain such as a company or a library. By extension, it also an abstract representation of the existing or project database that is made up of the data of this domain.

Database Reverse Engineering: The process through which the logical and conceptual schemas of a legacy database, or of a set of files, are recovered, or rebuilt, from various information sources such as DDL code, data dictionary contents, database contents, or the source code of application programs that use the database.

DB-MAIN: An experimental CASE tool developed at the University of Namur since 1993. It supports most database engineering processes, among which, information analysis, database design, reverse engineering, federated database design and database migration. Its generic structure model and method engineering environment allow users to build their own methodologies.

Logical schema: The description of the data structures of a database according to the model of a specific technology, e.g., a RDBMS. The logical schema of a database is the implementation of its conceptual schema. Application programs know the database through its logical schema.

Models and schemas: In the database realm, a model M is a formal system comprising a closed set of abstract object categories and a set of assembly rules that states which arrangements of objects are valid. Since M is supposed to describe the structure, the properties and the behaviour of a class S of external systems, the semantics of M is specified by a mapping of M onto S. Any arrangement m of objects which is valid according to M describes a specific system s of class S. m is called a *schema* while s is the *application domain* or the *universe of discourse*. Among the most popular conceptual models we can mention the Entity-relationship, Object-role, relational and UML class models. Among DBMS models, the SQL, CODASYL, IMS, Object-relational and XML models are curently the most widely used.

Physical schema: The technical description of a database all the physical constructs (such as indexes) and parameters (such as page size or buffer management policy) are specified. The physical schema of a database is the implementation of its logical schema.

Repository: A database in which the specification and the description of the various artefacts of a software system are stored. As far as database engineering is concerned, the repository includes the schemas of the project database at all the abstraction levels as well as the correspondence between them. All CASE tools rely on some sort of repository.

Traceability: The property of software design and development that makes it possible to link any abstract artefact to the technical artefacts that implement it, and conversely. In addition, this link explains how and why this implementation has been chosen. In the database realm, traceability allows a programmer to know exactly which conceptual object a definite column is an implemention of. Conversely, it informs him/her on how a conceptual relationship type has been implemented. The transformational paradigm is one of the most promissing approaches to formally guarantee traceability.