Datenbanken II A: DB-Entwurf

Chapter 3: Some General Remarks about Database Design

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/dd20/

Objectives

After completing this chapter, you should be able to:

- explain correctness and quality criteria for database schemas, explain difficulties and risks.

  Or vice versa, the basic kinds of design errors.

- enumerate what else, besides the mere schema design, needs to be done during a database project.

- explain the relationship between application programs and database design.

- Name some functions to expect from a database design tool.

  And advantages of using such a tool compared to using only a drawing program and a normal text editor.

Contents

The Task of DB Design (1)

- Database design is the process of developing a database schema for a given application.
- The requirements for a DB project can be specified by listing all of the questions which the DBS must be able to answer.

  Specific example values or variable names can be used.

- During DB design, a formal model of some aspects of the real world (a mini-world) must be built.

  The information which is needed to answer the required questions must be available.

The Task of DB Design (2)

- Building a model is done by abstraction:
  Details which are irrelevant for the given application are left out. Any model is a simplification of reality.

  Classification: On some abstraction level, objects are the same.

  Aggregation: Objects are seen as a unit (individual identity left out).

  Generalization: The same object on different abstraction levels.

- A model needs to be structured. Though text may contain all of the necessary information, it cannot be used by a computer for answering questions (except with higher AI).

The Task of DB Design (3)

- When a database schema is defined too closely to existing paper forms, text fields which can only be printed, but which cannot be used in statistical evaluations, may result.

  Suppose the goal is to determine from how many different countries there are students in this course. If the country is defined as a text field, entries might be e.g. “Germany”, “Federal Republic of Germany”, “Fed. Rep. Germany”, “FRG”. It will be necessary to eliminate synonyms manually.

- One must ask: What do I want to do with the data?

The Task of DB Design (4)

The three basic design errors are:

- There are situations in the real world which do not correspond to a database state.

  Data that actually occur cannot be entered.

- A legal question about the real world cannot be formulated as a query to the database.

  The needed information is missing in the database.

- Database states are possible which do not correspond to a legal state in the real world.

Constraints (1)

- Two kinds of errors must be distinguished:
  - Entering wrong data, i.e. the DB state corresponds a different situation of the real world than the actual one.

    E.g., 8 points given for Homework 1 in the DB vs. 10 in the real world. Then the DB state is wrong, but not the schema. What can be done to guard against such errors?
  - Entering data which do not make sense, or are illegal.

    E.g. -5 points for some homework, or two entries for the same student and same exercise. If such impossible data can be entered, the DB schema is wrong (design error).

Constraints (2)

- If the DB contains illegal/meaningless data, it becomes inconsistent with our general understanding of the real world.
- If a programmer assumes that the data fulfills some condition, but it actually does not, this can have all kinds of strange effects (including the loss of data).

  E.g. the programmer assumes that a certain column cannot contain null values. So he/she uses no indicator variable when fetching data. As long as there are no null values, this works. But if the schema does not prevents this, after some time, somebody will enter a null value. Then the program will crash (with a user-unfriedly error message).

Constraints (3)

- Given only the structural definitions (e.g. tables, columns, column datatypes), there are usually still many database states which do not correspond to states of the real world.
- Additional conditions which database states have to satisfy should be specified. In this way, invalid states are excluded.
- Such conditions are called "(integrity) constraints".

Constraints (4)

- Each data model has special support for certian common kinds of constraints, e.g. the relational model and SQL offer:
  - Keys: Unique identification of rows.
  - Foreign keys: Dynamic domain defined by a key.
  - `NOT NULL`: Entries for a column cannot be empty.
  - `CHECK`: Conditions that refer only to single rows.
- Arbitrary conditions can be specified as constraints (in natural language, logic, as SQL queries, programs, ...).

Constraints (5)

Why specify constraints?

- Some protection against data input errors.
- Constraints document knowledge about DB states.
- Enforment of laws / company standards.
- Protection against inconsistency if redundant data is stored.
- Queries/programs become simpler if the programmer is not required to handle the most general cases (i.e., cases where the constraint is not satisfied).

  E.g., if columns are known to be not null: no indicator variable.

Constraints (6)

Constraints and Exceptions:

- Constraints cannot have any exceptions.
- A good DBMS will reject any attempt to enter data which violates a specified constraint.
- One can expect that eventually there will be exceptional situations in which the DBS seems unflexible because of the specified constraints.
- Only conditions that are unquestionable should be defined as constraints.

Flexibility and Computers (1)

- The introduction of a computerized system always changes the real world.
- With the old paper-based forms, it was always possible to scribble something at the border or bottom of the form.
- In a database, a field for notes or remarks must be added to the table.
- But still, there is the problem that programs evaluating the data cannot understand these remarks, so they will simply ignore them.

Flexibility and Computers (2)

- Computers are stupid:
  Every possible situation must be anticipated when developing programs or database schemas.
- This is what makes database design difficult.
- On the other hand, computers are very fast, very precise (act 100% according to the given rules), and do not complain about stupid tasks.
- The decreased flexibility is not necessarily fatal, if the users can accept it.

Contents

Interpretation of the Data (1)

- The formal database schema describes only data.
- Data becomes information by interpretation.
- This interpretation, i.e., the mapping between database states and situations in the real word, must be documented as part of the database design task.

  It must be clear what the stored data actually means and what the users of the system are supposed to enter in the table columns.

- The task of database design is certainly not complete when only a set of `CREATE TABLE` statements is delivered. Additional documentation is needed.

Interpretation of the Data (2)

- Part of the task can be solved by choosing good names for the schema elements (e.g., tables and columns).
- Names should be self-documenting (understandable without additional explanation), but also not too long (e.g. max. 18 characters), and not similar to names elsewhere in the schema.
- Choosing good names needs some thought.

  But the invested time will later pay off. Discussing the names with other people might help. The DB designer must talk about the schema with the future users, customers (domain experts) and programmers.

Interpretation of the Data (3)

- Abbreviations and other naming conventions should be documented and used consistently (especially important when developing in a team).

  E.g. table names: Some designers use the singular form of a noun, some the plural form. E.g. placement of underscores, capitalization.

- The documentation might include a small example DB state.

- There should be some explanation for every schema element (e.g. tables and columns).

  These can be used later in the help files for input fields.

Interpretation of the Data (4)

- The data inside the tables needs interpretation:
  - meaning of specific codes (elements of enumeration types),
  - units for physical measures,
  - format of strings that contain several pieces of data (this should anyway be avoided),
  - unusual/ambigous meanings for specific values,

    E.g., -1 point: student should resubmit. Really bad style!
  - meaning of null values.

Interpretation of the Data (5)

Examples of possible misunderstandings:

- General concept vs. concrete instances.
  - E.g., the course INFSCI 2711 "Database Analysis and Design" vs. this course given in a specific term (key: CRN).

    At the University of Pittsburgh, the five-digit course reference number (CRN) identifies a particular instance of the course, for which students can enroll and get grades. In Halle, we also have the "abstract module" and the "concrete module".
- Null values and strings of zero length should not be allowed for the same column (too difficult to distinguish).
- The two ends of recursive relationships.

  "parent_of" actually contains the ID of the child.

Interpretation of the Data (6)

- Entities in a context vs. globally unique entities.
  - If one student is in two classes that I teach, do I list him/her as one student or two separate instances of a student?

    Am I counting "total bodies" in each of my classes, or the number of unique students in all of my classes?
  - If two patients have bronchitis, is this counted as two different health problems or two instances of the same problem?

Interpretation of the Data (7)

- Non-existence of a relationship vs. existence with value 0:
  - If a student did not yet submit a homework, is there an entry in the results table with 0 points or is there no entry for this combination of student and homework?
  - Should the student grade be calculated as the average of 2 homeworks (95, 85) and ignore the missing homework, or should the grade be the average of 3 (95, 85, 0)?

Interpretation of the Data (8)

- Need for historical information:
  - E.g. it should be stored who borrowed a book from the library.
  - Can the information be deleted when the book is returned?
  - Does a counter suffice how often the book was borrowed? Or is the complete history needed?
  - If the same student borrowed the same book several times, does this have to be stored?

Contents

Schemas: Quality Criteria (1)

- Syntactical Correctness: The schema is legal with respect to the given data model.
- Completeness: The necessary data can be stored.

  All the given questions about the real world can be answered from the database.

- Enforcement of Business Rules: Illegal updates are rejected.

  Data violating the given business rules cannot be entered.

Schemas: Quality Criteria (2)

- Sufficiently general: All situations that are possible in the real world can be represented in the database.

  I.e. all data that does not violate the business rules can be stored.

- Preciseness: The relation to the real world is exactly documented.

  The intention/interpretation of schema elements must be clear.

- Non-Redundancy: Every relevant aspect of the real world should be represented only once.

  The schema should be minimal, i.e. no schema element can be removed without violating the completeness requirement.

Schemas: Quality Criteria (3)

- Normality: If translated into the relational model, the schema will be in BCNF/4NF etc.

  This is related to non-redundancy and sufficient generality.

- Stability/Flexibility/Extensibility: The schema can be easily adapted to changing requirements.

- Simplicity and Elegance

  A solution with fewer, more generic schema elements might be preferable to a larger schema.

- Making good use of data model constructs

  E.g. EER-constructs ("Extended ER") reduce the need for general constraints.

Schemas: Quality Criteria (4)

- **Communication Effectiveness, Self-Documenting**

  Names of schema elements should be chosen well (to make the interpretation of data clear). Terms should be familiar to business specialists.

- **Readability**

  Diagrams should be drawn in a grid, line crossings should be minimized, symmetric structures should be emphasized, related concepts should be near in the diagram.

- **Uniformity**

  Style, naming conventions, abbreviations should be uniform.

Contents

Data vs. Programs (1)

The beginning of a DB project is the understanding that there are specific real-world tasks which need to be supported by computerization:

- The tasks require that data be collected and compiled so that they can be analyzed or summerized.
- Programs need to be created to facilitate the collection, compilation and querying of the data.
- DB design and application development are of equal importance (neither are subordinate to the other).

Data vs. Programs (2)

- In normal software-engineering projects, the programs are seen as the main goal and the data only as a means of implementation.
- Database projects are special:
  - There are usually many programs that access the same database.
  - The same data may be used by future programs.
  - Ad-hoc SQL queries and even updates can be used on the data.

Data vs. Programs (3)

- The specification of programs&data is intertwined:
  - The data must meet the information needs of the programs (no data is missing).
  - No unnecessary data (i.e. data not needed by any current or forseen program) should be collected.
  - Programs are needed to insert/modify the data.
- As ad-hoc queries and updates in SQL are possible, the second and third condition can have exceptions.

  It is important for a DB project to know whether there will be users knowing SQL or whether the goal is closed system.

Data vs. Programs (4)

- CRUD-analysis: Matrix that shows which program creates / retrieves / updates / deletes data for which schema elements.
- E.g. the old homework results database consisted of three tables and four programs:

Program	STUDENTS	RESULTS	EXERCISES
Registration	C		
Change Password	RU		
View Results	R	R	R
Import Points	R	C	R

Data vs. Programs (5)

- It is difficult to specify what application programs have to do without refering to a DB schema.
- The database schema determines already a large part of the needed programs.

  Basically, for every table a program is needed to enter/display the data. One program may do this for a small set of tables. Lookup tables don't need programs (fixed after DB creation).

- The database schema is smaller than the complete specification of the needed programs.

  It can be understood as a concise representation of the essential functions of a large subset of the required programs.

Data Independence (1)

- Before databases were widely used, data were stored in files directly managed by programs.
- The programs were considered the main thing, there was no independent documentation for the data.

  The data was organized in a way which was well-suited only for the single program which used the file.

- It was difficult to use the data for other purposes than the one for which they were orginally collected.

  It is frustrating if one knows that the "information is in there", but the new evaluation would be too difficult to program (or even require manual analysis). [Good from the data privacy standpoint . . .]

Data Independence (2)

- Data often lives longer than the programs.

  New versions of programs are developed relatively frequently, but the data collected with the old program cannot be thrown away, it has to be migrated to the new system (might require considerable effort).

- Thus, data must be seen independent from a specific program.
- Vice versa, programs should not depend on the way the data is stored (data organisation/file format).
- The independence of programs from the data organization is called "physical data independence".

Data Independence (3)

- It might be necessary to change the data organisation from time to time, e.g. because
  - the number of rows in a table has grown so much that a sequential scan of all rows to find one with a specific value takes too long.

    An index must be added (e.g. a B-tree). A indes over attribute A of relation R speeds up queries that search for rows in R with $A = c$, where c is a constant (plus possibly other queries).
  - certain application programs are executed so often that a single disk cannot support the required number of accesses per second.

Data Independence (4)

- It would be bad if one had to change all application programs when the data organization is changed.

  When programs directly access files, this is of course necessary.

- In relational DBMSs, indexes can be added or deleted without any change to an application program.
- SQL is a declarative language: One specifies only which conditions the result must satisfy, but not how it should be computed.
- The query optimizer automatically uses indexes.

Data Independence (5)

- This has led to the distinction between two schema levels:
  - The Conceptual Schema describes the logical information contents of the database.

    E.g. in the relational model.
  - The Internal Schema (or Physical Schema) describes the way the data is actually stored.

    E.g. relations plus indexes, disks, and many storage parameters.
- Users can refer (in SQL queries) only to the conceptual schema.

Data Independence (6)

- The query optimizer translates the SQL query into an internal query program which is evaluated on the actually stored instance of the internal schema.
- In most systems, the storage parameters are defined as part of the `CREATE TABLE` statement, and most have a `CREATE INDEX` command in their SQL.

  Theoreticians would have wished a clearer separation. But since the internal schema normally must repeat the information in the conceptual schema and add its own parameters, this is not very practical.

- However, these are not part of the SQL standard and highly system dependent.

Data Independence (7)

- Database systems are subject to constant evolution: The information requirements change and there are structural changes in the part of the real world being modelled.

  "The only programs that never change are those that are not used."

- Often, additional application programs are developed for an existing DB. These programs access the existing data, but might also need additional data.

- E.g., columns must be added to existing tables.

Data Independence (8)

- The goal of logical data independence is that the existing application programs do not have to be changed when the conceptual schema is extended.

  Or modified in other information-preserving ways.

- This is reached by adding a third schema level, the "external schemas".

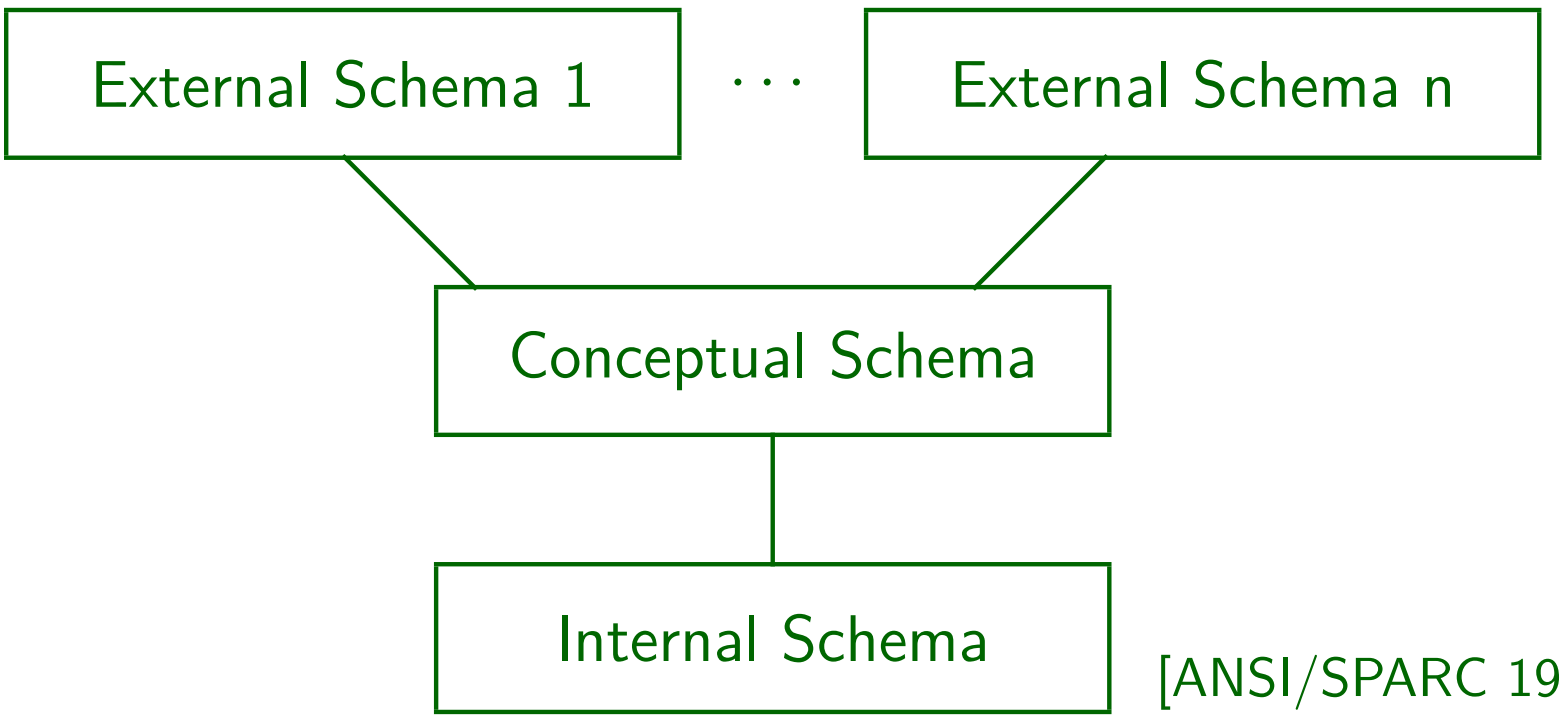
- In this model, each application program (or group of programs/users) has a schema of its own.

Data Independence (9)

- The data for the external schema are not actually stored. They consist of views (virtual tables) that are computed from the data for conceptual schema.
- In this way, it might be possible to keep the existing external schemas stable when the conceptual schema has to be changed.
- In current systems, it is possible, but not required to separate conceptual and external schemas.

  Design decision: Should there be one account (schema) per application (filled with views) and one (or more) for the base tables?

Data Independence (10)



[ANSI/SPARC 1978]

Contents

CASE-Tools (1)

- CASE = Computer Aided Software Engineering.
- CASE tools support the development of software, e.g. by
  - managing design documents (multiple versions),
  - enforcing syntax rules,
  - performing consistency/style checks,
  - managing dependencies between components,
  - translating between different views of a system,
  - generating code,
  - supporting project management and team work.
- There are special CASE-Tools for database projects (data modeling tools).

  See the software list in the Syllabus (Chapter 0 of this course).

CASE-Tools (2)

- Standard features of database CASE-Tools:
  - A specialized graphical editor for ER-diagrams is a standard component of such tools.
  - Support for different kinds of diagrams, e.g. ER-diagrams, diagrams of relational schemas, business process diagrams.
  - Repository for storing all design documents.

    This should include version management and consistency checks. Normally, many ER-diagrams must be managed. A single one would be too big (could only be used as wallpaper).
  - Automatic translation from the ER-model into the relational model (and vice versa).
  - Automatic generation of software prototypes.

CASE Tool Advantages (1)

- All documentation in one place.
- Easy modifications. Version Management.
- Graphical editors for various types of diagrams that enforce the syntax of the respective diagram type.

  It is easier to use a graphical editor that knows about ER-constructs than a general graphical editor. In addition, the resulting ER-diagram is guaranteed to be well-formed.

- Information in the repository is structured, and not pure text.

  This makes searches more selective, and allows complex evaluations (e.g. average number of attributes per entity type).

CASE Tool Advantages (2)

- Browsing of the information is possible (hyperlinks).

  E.g. one can click on an entity type in an ER-diagram and gets a dialog box that contains additional information.

- Cross-references are enforced (no broken links).

- Consistency between documents.

  E.g. if the same entity type appears in two diagrams, it has the same attributes: It is stored only once in the repository.

- Different views on the same data: Design documents at different levels of detail.

  The same piece of information has to be entered only once instead of once for every document that contains it.

CASE Tool Advantages (3)

- The translation from an ER-schema to a relational schema is quite tedious if done by hand.

  It is also easy to make mistakes in a manual translation. The consistency between the ER-diagrams and the relational schema is guaranteed if an automatic translation is used. If the two get out of sync the ER-diagrams become misleading and a useless documentation.

- A CASE tool contains knowledge
  - how software should be developed and
  - which documentation is needed.

Contents

Business Rules (1)

- Business rules are similar to constraints, but
  - they refer to the real world, not to the DB.

    Constraints can only be specified after the structure of the database state is defined (e.g. tables, columns). Business rules describe restrictions in the real world.
  - they are more general.

    They not only restrict states in the real world, but also "who is allowed to do what?" and temporal constraints and procedures that must be followed ("if the invoice is not paid after 30 days, a letter is sent to remind the customer").
- Business rules are what prevents the business from chaos (not everybody can do what he/she wants).

Business Rules (2)

- Like constraints, business rules cannot have any exceptions.

  This might be difficult for business people to understand, but "flexible" business rules are basically not relevant for database design. They might be useful for programs (default values, warnings).

- It is also important which business rules are likely to change in future and which ones are very stable.

  "I watched a large insurance company struggling to introduce a new product. The hold-up was the time required to develop a supporting information system. Meanwhile, one of the company's competitors was able to introduce a similar product, making use of an existing information system, and win a major share of the market." [Simsion/Witt, 2001]

Business Rules (3)

During DB design, business rules are transformed into:

- Structural elements of the schema

  E.g. if every student can have only one contact email address for this course, it can be stored as an attribute of the `STUDENTS` table. Otherwise, an extra table is needed.

- Constraints

  If each student must have an email address, this attribute must be `NOT NULL`. If there cannot be two students with the same first and last name, these two attributes form a key.

- View Definitions

  The weighting of points for a course is 30% homeworks, 35% project, 35% final exam.

Business Rules (4)

Results of Business Rule Transformation (continued):

- Programs

  If first and last name are unique, they can be used to identify students in program inputs. Otherwise a more complicated selection procedure is necessary. Programs can also be used to check more general constraints than can be declared in current database systems.

- Triggers
  (procedures that are executed at certain updates).

  E.g. if the quantity on hand is smaller than 5, the item is reordered.

Business Rules (5)

Results of Business Rule Transformation (continued):

- Programs that are executed in certain time intervalls (e.g. every day, at the end of each month).

  E.g. if the homework is not submitted one week after the deadline, the student gets 0 points for it.

- Database Accounts, Access rights, Views

  E.g. if there is a global homework results database (for all courses of the department), but each professor may see only the results of his/her students, there probably will be accounts for every professor who wishes SQL access and a view that selects the data the current user may see. Without direct SQL access, the checking can be done in the application programs. But this means that the programs must do the user management instead of letting the DBMS do this work.

Contents

Tasks of a DB Project (1)

- Development of the database schema.

  Including physical parameters and external views.

- Development of the application programs.

  Including interfaces to other systems.

- May require a redesign of business processes.

  This may actually help the business.

- Migration of old data, cleaning old data.

  The old data might not fully satisfy the new constraints. The necessary "data cleaning" can take a lot of time.

Tasks of a DB Project (2)

- Entering/buying additional data.
- Ensuring that performance requirements are met.
- Defining access rights.
- Writing documentation.
- Training users.

  In the transition phase, many questions must be answered.

- Developing procedures for backup and recovery.

  A knowledge transfer to the DBA might be a project goal.

DB Design is not Easy (1)

- The designer must learn about the application domain.

  Domain experts often don't bother to say the obvious (obvious to them) or mention rare exceptions. They use technical terms which the database designer must learn.

- Exceptions: The real world is very flexible.

  One must somehow extrapolate from the single state (or the few states) one observes to all possible states.

- Size: Database schemas can be very big.

  Despite the name "miniworld", schemas with more than 100 entity types are still quite normal.

DB Design is not Easy (2)

- The solution is usually not unique.
- Sometimes there is no perfect solution, one can choose only between two bad things.
- Existing software or data might reduce the choices.
- Such a project brings changes into the company, but the users might fear changes (only management wants it).

DB Design is not Easy (3)

- It is a mistake to assume that once you know the syntax of the ER-model, you can work as DB designer for large projects.
- What else is needed (besides experience)?
  - Translation from the ER-model into the relational model, reverse engineering (from existing relations to ER-diagram).
  - Normal form theory and the intuition behind it, redundancy, constraints.
  - Having seen many DB designs, knowning typical patterns.

DB Design is not Easy (4)

- Things a DB designer should know, continued:
  - Interviewing techniques.
  - Basic business knowledge.
  - Form analysis, text analysis, view integration, schema condensation, ...
  - Business process modeling, CRUD-analysis, ...
  - CASE-tools, software engineering techniques.
  - SQL, current database software, programming knowledge.

DB Design is not Easy (5)

- "Data quality is almost certainly the biggest problem you're going to have in a legacy-bound project. If your schedule doesn't include a big chunk of time for analyzing, fixing, and testing the data from the legacy system, your schedule is wrong." [Muller, 1999]
- "If the system you're proposing to build is an order of magnitude greater in size than the ones you have built previously, it is a good bet your culture isn't capable of doing it." [Muller, 1999]
- "There was a reuse organization (in yet another building) that needed to have a say in making sure everything was reusable, or was at least contributing to the concept of reuse. The head of this organization did not like the head of the application organization, so nothing ever got done." [Muller, 1999]

References

- Ramez Elmasri, Shamkant B. Navathe: Fundamentals of Database Systems, 3rd Ed., Ch. 16, "Practical Database Design and Tuning".
- Toby J. Teorey: Database Modeling & Design, 3rd Edition. Morgan Kaufmann, 1999, ISBN 1-55860-500-2.
- Graeme C. Simsion, Graham C. Witt: Data Modeling Essentials, 2nd Edition. Coriolis, 2001, ISBN 1-57610-872-4, 459 pages.
- Robert J. Muller: Database Design for Smarties — Using UML for Data Modeling. Morgan Kaufmann, 1999, ISBN 1-55860-515-0, ca. $40.
- Peter Koletzke, Paul Dorsey: Oracle Designer Handbook, 2nd Edition. ORACLE Press, 1998, ISBN 0-07-882417-6, 1075 pages, ca. $40.
- Martin Fowler, Kendall Scott: UML Distilled, Second Edition. Addison-Wesley, 2000, ISBN 0-201-65783-X, 185 pages.
- Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide. Addison Wesley Longman, 1999, ISBN 0-201-57168-4, 482 pages.
- Carlo Batini, Stefano Ceri, Shamkant B. Navathe: Conceptual Database Design. Benjamin/Cummings, 1992, ISBN 0-8053-0244-1, 470 pages.
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German). Teubner, 1997.
- Udo Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.