# Datenbanken II A: DB-Entwurf

# Chapter 4: Design Phases

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2024/25

http://www.informatik.uni-halle.de/~brass/dd24/

# Objectives

After completing this chapter, you should be able to:

- explain the three phases of database design.

  What has to be done in each of the phases?

- explain the advantages of doing a complex project in serveral phases.

  Why does one not directly start with a relational design?

# Contents

# Database Design Phases (1)

- There are usually three schema design phases:
  - Conceptual Database Design produces the initial model of the real world subset in a conceptual data model (like the Entity-Relationship-Model).
  - Logical Database Design transforms this schema into the data model supported by the DBMS (often the relational model).
  - Physical Database Design aims at improving the performance of the final system. E.g., indexes and storage parameters are selected.

# Database Design Phases (2)

## Why multiple design phases?

- Reduction in complexity

  If not all design decisions depend mutually on one another, problems can be separated and attacked one after the other.

- Protection against changes

  If design decisions do not depend on specific input parameters, they are not invalidated by changes to those parameters.

- Different tasks need different tools/techniques

- Milestones, Ceremony (accepted method)

  Easier to track the project's progress vs. the schedule.
  Project can celebrate (or submit bills for) each milestone.

# Database Design Phases (3)

- E.g., during conceptual design, there is no need to worry about limitations of a specific DBMS.

  Focus is on producing a correct model of the real world.

- DBMS features do not influence conceptual design, and only partially influence the logical design.

  This ensures that the conceptual design is not invalidated, if a different DBMS is later used.

- In the conceptual schema, non-standard datatypes for the attributes can be used.

  Of course, this makes the logical design more difficult. But object-relational systems do have an extensible type system.

# Database Design Phases (4)

- Only the physical design should depend on
  - sizes of database objects,
  - invocation frequency for each program,
  - performance of the hardware,
  - quality of the DBMS query optimizer.
- These parameters will change over time!
- If the logical design depends on them, it must be changed, which means that application programs must be changed, too.

# Database Design Phases (5)

- Don't accept compromises in the logical schema for the sake of performance.

  Unless experiments prove that the current design cannot deliver the required performance.

- Old DB designs are often heavily denormalized, which makes changes difficult and expensive.

  Each piece of redundant data (that is not completely managed by the DBMS, like, e.g. an index) makes application programs more difficult and inconsistencies possible. Denormalization also means that certain pieces of information can only be stored together, which makes the schema less flexible.

# Database Design Phases (6)

## View Integration:

- The conceptual design step is much more complicated than the other two.

  The logical design step can be largely automatic, and the physical design has a relatively limited set of options.

- Often, it is not possible to create the complete ER-Schema in one step, because this is very large.

- Then one starts with small ER-schemas which describe only the data necessary for one application or user (or a small group of related applications).

  For each application/user, one such schema is developed.

# Database Design Phases (7)

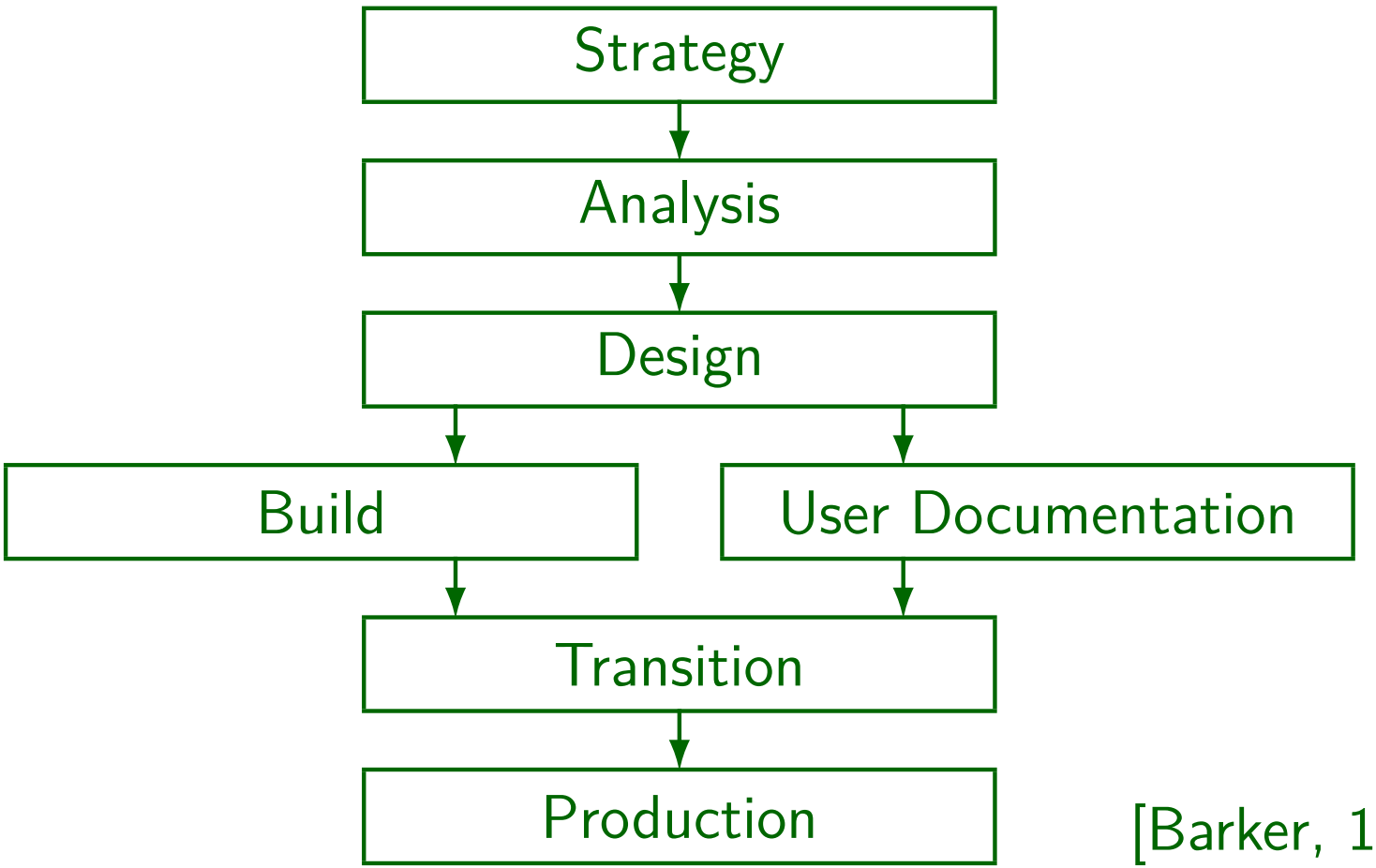
- In this case one starts with the design of the external schemas before the conceptual one.

  Of course, this is also done in the entity-relationship model. It is possible that not only the conceptual schema, but also the external schemas are translated into the relational model during the logical design and actually exist as views in the final database. But this is a design decision. One can also treat them as only temporary sketches for collecting requirements.

- These schemas must then be integrated to get the complete enterprise data model (i.e. the conceptual schema of the database).

# Contents

# Oracle CASE*Method




[Barker, 1990]

# Strategy Phase (1)

- The purpose of the strategy phase is to develop a plan for information systems development.
- The planned system must serve the organization's current and future needs.
- The plan must also take into account organizational, financial, and technical constraints.
- Of course, management wants to know "what will we get?" and "how much will it cost?" before the project goes into the next phase.

# Strategy Phase (2)

- The strategy phase results in a contract.
- However, even at the end of the strategy phase, estimates about the cost will not be very reliable.
- The contract can state that the price is
  - fixed.

    Then the price might be higher than necessary (the developers take the whole risk) and the finished product might be not very good (just satisfy the requirements in the contract).
  - an hourly rate.

    Then there is no incentive to ever finish the product.
  - a mixture of both or hourly rate paid at the end.

# Strategy Phase (3)

- Already in this phase, ER-diagrams and function hierarchy / business process diagrams should be developed.
- They do not yet have to be very detailed, but they should cover the whole area of the planned system.

  E.g. attributes might not yet be needed. But definitions/descriptions of all entities might be very useful. The more of the analysis that can be done in the strategy phase, the better (but time/money is limited).

- What will be the architecture of the proposed system?

# Strategy Phase (4)

- It is important to understand the company, e.g:
  - Business objectives,
  - Critical success factors,
  - Strengths, Weaknesses, Opportunities, Threats
  - Key performance indicators.
- Existing systems (legacy systems) and required interfaces must be understood and documented.
- Develop good working relationships with the people involved and understand the political environment.

  Many projects are bound to fail because of the political environment.

# Strategy Phase (5)

- Develop a timeline for the development (project plan) and an estimate of the needed resources.

  Time and money are important resources. But also the access to stakeholders and users (interview partners) is an important resource. The valuable time of people within the company is critical to the project, but must be listed as a project cost. Also access to hardware and to the data must be discussed.

- Is the project feasible in the given limits?

- Prioritize the project goals: Not everything that would be nice to have is worth the effort.

  If it should turn out later that time or budget is insufficient: What can be sacrificed and what is essential?

# Strategy Phase (6)

- Think about risks to the project and what can be done to manage them.
- Develop a cost-benefit analysis and provide sufficient motivation
  as to why the proposed project is worth the effort.

  Quantify the impacts on the business.
- One method to estimate the complexity of a project is the Function Point Method.

  See: Software engineering textbooks, `http://www.ifpug.org/`.

# Analysis Phase (1)

- In the analysis phase, all system requirements are gathered in complete detail ($\approx$ conceptual design phase).

  This builds on the results of the strategy phase.

- The final ER-diagrams are developed, including all attributes and business rules/constraints.

- The function hierarchy/business process diagrams are further developed. Dataflow and entity usages are analyzed.

# Analysis Phase (2)

- Legacy systems must be carefully analyzed and a strategy for transition and data migration must be developed.

  Don't underestimate the effort of data migration (from the old system into the new system). How will data be handled that violates the constraints? Is data cleaning possible?

- Describe required interfaces with other software.
- Collect information about the expected data volumes, function frequencies, and performance expectations.

# Analysis Phase (3)

- Collect security requirements.
- Collect requirements for backup/recovery.
- "It is not possible to meet a user's need that was never discovered." [Koletzke/Dorsey]
- "A thorough requirements document can easily fill several thousand pages." [Koletzke/Dorsey]
- Describe what is needed, but do not yet think too much about how it should be done.

  The focus is still on the user, not on the system.

# Design Phase (1)

- The focus now shifts from the user to the system.
- The relational database design is developed based on the given ER-model ($\approx$ logical design phase).

  Probably denormalization should already be considered (if really necessary), but other physical design decisions (e.g. indexes) can be deferred until the build phase. When defining the tables, you should work together with an experienced DBA (preferable the one who has later to live with the design).

- Functions are mapped into modules (application programs) and manual procedures.

# Design Phase (2)

- "The Design phase is where the blueprints are drawn for building the system. Every detail should be laid out before generation." [Koletzke/Dorsey]
- Design standards must be set. This includes the development of screen concept prototypes.

  All programs should have the same look and feel.

  User documentation should have a similar structure.

  Programming styles should be uniform (naming standards).

# Design Phase (3)

- "Design is complete when the design documents could be handed over to another team to build, with each application having its own screen (or report) design, list of detailed functionality, and create-retrieve-update-delete (CRUD) report."
  [Koletzke/Dorsey]

  This is an exact specification of the applications, similar to blueprints of an architect which are given to a contractor for building a house.

# Build Phase (1)

- In the Build phase, the working system is created.
- E.g. tables, views, procedures, triggers and other database objects are created, the final decisions of physical design are made.

  Storage parameters for tables including the partitioning among tablespaces/disks, indexes, clusters, etc.

- The database should be filled with example data of the same size as the production database will be.

  Only in this way performance can be tested and tuned.

# Build Phase (2)

- The application programs are developed (hopefully, many programs can be generated with a tool like Oracle Designer out of specifications developed during the Design phase).
- Of course, testing the developed programs is mandatory.

  First, every developer will test his/her program in isolation. But then also other people including real users must test it, and the integration with other programs must be tested. A test plan should be developed during the design phase.

# Build Phase (3)

- "Whenever systems are built, apparently small constraints and limits get introduced during the build stage:
  - I can't imagine them ever needing more than 255!
  - The biggest one I've ever seen had only seven line items.
  - I think I'll code those codes directly into the program to make it work faster!" [Barker, 1990]

# Documentation (1)

- "Documentation should be an ongoing process occurring throughout the system development process. It should accompany the first prototype the user sees and every other software deliverable." [Koletzke/Dorsey]
- "We all know the nightmare stories of developers who come in to modify an existing system for which there is no documentation." [Koletzke/Dorsey]

# Documentation (2)

- "By preparing careful system and user documentation throughout the life cycle of the project, developers are not left with a major task at the end. In addition, frequently no client money is left at this point to pay to extend the development process further." [Koletzke/Dorsey]
- System documentation will be mainly developed during the Design phase. User documentation (and the help system) can only be developed when the design is complete.

# Documentation (3)

- Time and money invested in good documentation will later pay off by
  - less phone calls of users who need help,
  - less time lost by the users for trying to find a way to do what they need to do,
  - a better impression by the users about the software quality,
  - easier (cheaper) maintainance/modifications.
- A user manual can even say “This is no bug, this is a feature”, and the users might accept that.

# Documentation (4)

- Few people read a big manual before they start using the software.
- There should be a short introduction ($\leq$ 20 pages).
- After that, a good table of contents, a good index, and good cross-references are essential.

  It should be possible to understand a section without reading all the previous ones. However, a few users do want to read more than the introduction in a sequential manner. Repeating again and again the same things is not nice for them. Sequential readers also can expect that concepts are defined before they are used.

# Documentation (5)

- Manuals are always missing when they are needed.

  Thus, there should be a good online help system. Documentation should be available in electronic form.

- Documentation might also include the preparation of training courses.

- Also, a web site might be developed that contains an FAQ and a list of bugs and other problems that are currently being resolved.

  A good website might mean that less support/help desk people are needed at the telephones.

# Transition Phase (1)

## Big Bang (vs. Gradual/Phased Transition):

- One one day, all tasks are switched to the new system.
- Clean solution, no development effort into temporary interfaces.
- Risky: What happens if the software does not quite work?

  Developers will always promise that it works tomorrow and only minor details are missing (99% effect). When do you switch back to the old system? Can you switch back to the old system?

# Transition Phase (2)

## Big Bang, continued:

- Needs a lot of training.

  Even with training, it will look different when the employees have to do real work with it. In the days after the switch, there might be not enough staff to answer all questions. And the development team will be busy removing real errors.

- Companies can go bankrupt this way.

  The productivity will go down for a while. There must be financial reserves to survive this.

# Transition Phase (3)

## Gradual/Phased Transition:

- Temporary interfaces between new parts and old parts are needed. (These will be thrown away in the end.)

  Thus, the overall development cost is certainly bigger.

- Certain tasks (e.g. copying data between systems) might need to be done manually (extra work, possible errors).

- One can get an impression of the software quality and the transition problems first for a smaller part of the company.

  But this might be able to paralyze the rest of the company.

- Users who already switched to the new system may help in training users which still have to switch.

# Contents

# Risks / Risk Management (1)

Consider possible risks and what to do about them:

- The collected requirements are wrong or not complete.

  In order to reduce this risk, one can invest more time and money into the requirements analysis: One can do more interviews, study more existing standard solutions, have more thorough presentations and discussions of the solution, play through more example scenarios, develop more prototypes. Of course, one can also hire more experienced data modellers. There is a tradeoff between risk and money, but sometimes relatively little money or simply doing things in a different way can significantly reduce the risk.

- The requirements change.

# Risks / Risk Management (2)

- The software is not ready on time.
  The budget is insufficient.
- The software does not work correctly, it might actually destroy data or enter incorrect data.
- The DBMS is down (not available).

  E.g. because of hardware faults, software bugs, bad adminstration (this includes the case that a disk is suddenly full).
- The system does not deliver the required performance.

# Risks / Risk Management (3)

- The DBMS vendor goes bankrupt and and the software is no longer supported.
- The DBMS vendor changes the licensing terms and the system gets more expensive (at least updates).
- A disk fails. There is a fire in the computer room.

  Although it might be possible to restore the latest DB state, this might takes hours (downtime).

- The DBA accidentally deletes an important table.

# Risks / Risk Management (4)

- The programmers or the DBA do not sufficiently know the DBMS software.
- Important people leave the project.
- Employees (users of the system) do not know it well enough to use it correctly and efficiently.
- Employees accidentally enter incorrect data.
- Employees accidentally delete important data.

# Risks / Risk Management (5)

- A hacker tries to access or damage the data.
- Somebody who leaves the company takes information from the database with him/her.

  In the extreme case, an export file of the entire database.
- The employees do not like the new system.
  The worker's union protests against it.
- The system violates data privacy laws.
  Or the company gets a bad reputation because of questionable practice regarding personal data.

# 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.