Part 1: Introduction

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.
- Richard Barker: CASE*Method: Tasks and Deliverables. Addison-Wesley, 1990, ISBN 0201416972, ca. $69.
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German). Teubner, 1997.
- Udo Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.

Objectives

After completing this chapter, you should be able to:

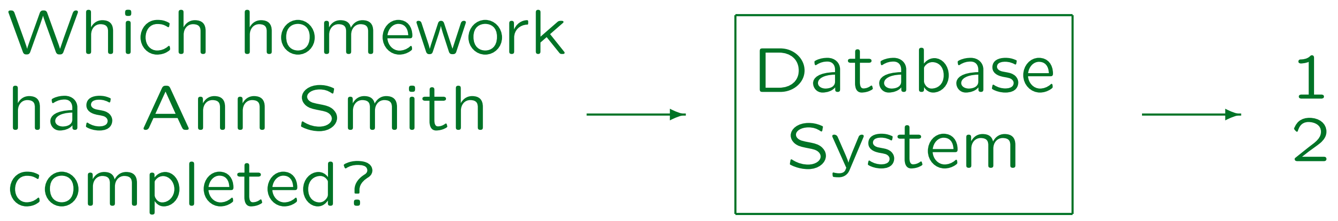
- explain correctness and quality criteria for database schemas, explain difficulties and risks.
- enumerate what else, besides the mere schema design, needs to be done during a database project.
- explain the relationship between application programs and database design.
- explain the three phases of database design.

  Why does one not directly start with a relational design?
- explain a system development lifecycle.

Overview

Basic Database Notions (1)

- The main task of a database system (DBS) is to answer certain questions about a subset of the real world ("domain of discourse"), e.g.



- This is done by selecting, aggregating and combining information that was previously entered.
- The system must know the structure of the data to support powerful queries.

Basic Database Notions (2)

- In the relational data model, the data is structured in form of tables (relations).
- Each table has a name, sequence of named columns (attributes) and a set of rows (tuples).

Solved		
Student	Homework	Points
Ann Smith	1	10
Ann Smith	2	8
Michael Jones	1	9
Michael Jones	2	9

DB Schema

DB State (Instance)

Basic Database Notions (3)

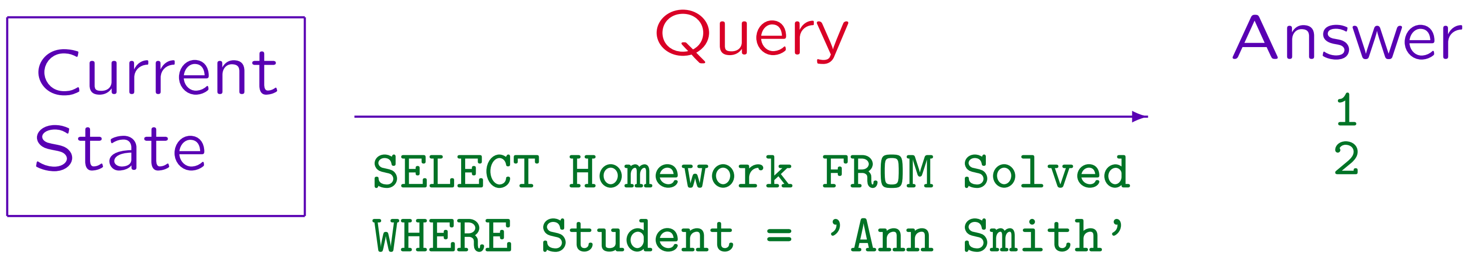
- In SQL, schemas are declared by means of `CREATE TABLE` statements:

```
CREATE TABLE Solved
(Student  VARCHAR(40) NOT NULL,
 Homework NUMERIC(2)  NOT NULL
                      CHECK(Homework > 0),
 Points   NUMERIC(2)  NOT NULL
                      CHECK(Points >= 0),
 PRIMARY KEY(Student, Homework))
```

- Schemas with 20-100 tables are medium size, tables with >100 columns are common.

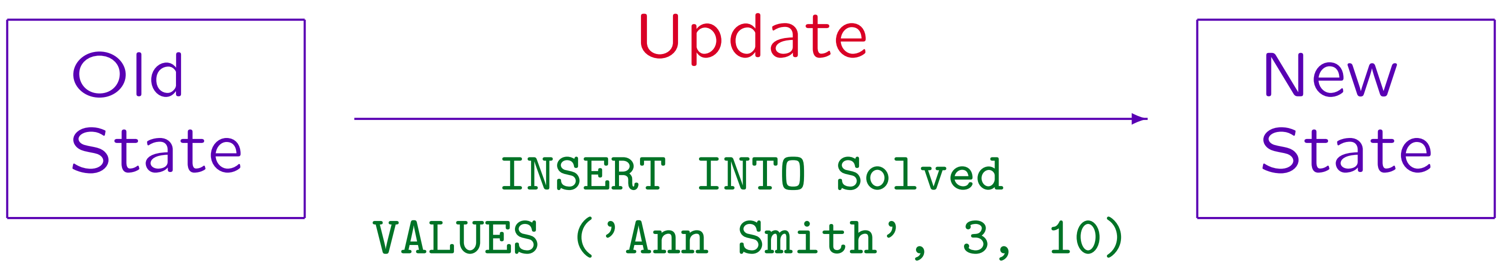
Basic Database Notions (4)

- The information is stored in the database state:



```
SELECT Homework FROM Solved
WHERE Student = 'Ann Smith'
```

- Entering, modifying, or deleting information changes the state:



```
INSERT INTO Solved
VALUES ('Ann Smith', 3, 10)
```

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 them.

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).
- 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?

Overview

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.

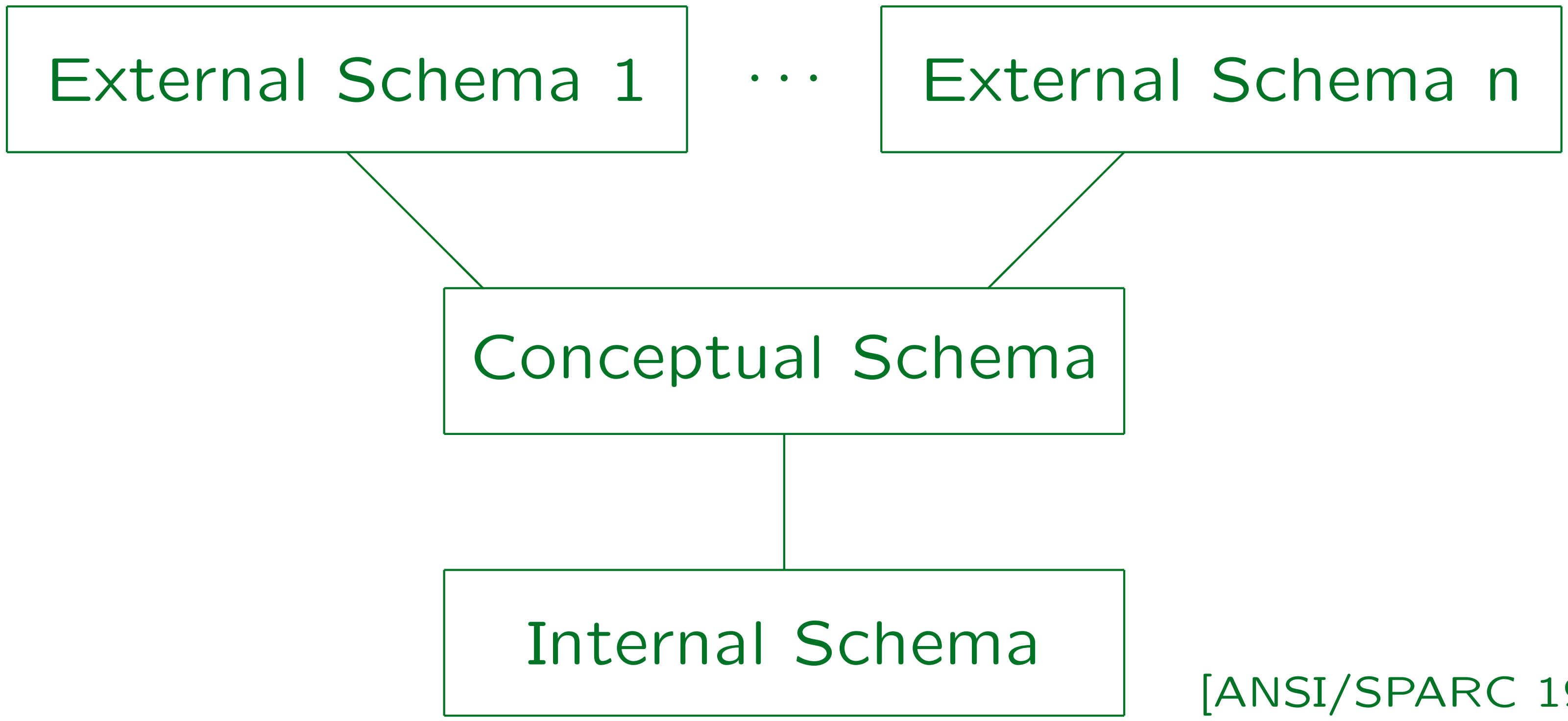
- This is reached by adding a third schema level, the “external schemas”.
- In this model, each application program (or group of programs/users) has a schema of its own.

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]

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.

Overview

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)

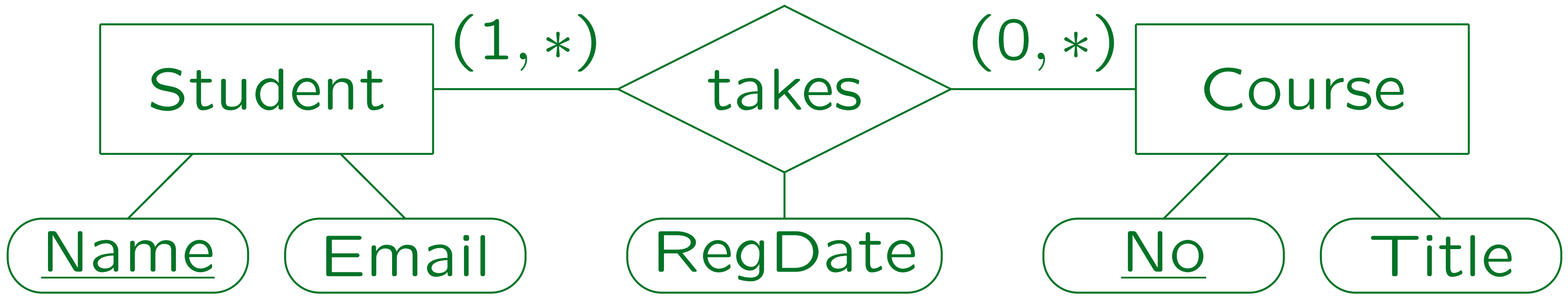
- In this case one starts with the design of the external schemas before the conceptual one.

  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).

Entity-Relationship Model (1)

ER-Schema in Graphical Notation:



- This mini-world contains students and courses (entities, objects).
- Students have a name and an email address (attributes, properties, data about objects).
- Students are identified by their name (key).

Entity-Relationship Model (2)

- Courses have an identifying number and a title.
- Students take courses (relationship).
- A student can take between 1 and arbitrarily many courses, and a course can be taken by 0 or any number of students (many-to-many relationship).

  However, between each student and each course at most one connection can exist (relationships are sets of pairs).
- For each combination of student and course, such that the student takes the course, a registration date (RegDate) is stored.

Entity-Relationship Model (3)

- Proposed by Peter Pin-Shan Chen (1976).

  An International Conference on the Entity-Relationship Approach has occurred almost every year since 1979.

- Standard tool for conceptual design.

  Every professional DB designer must know it well.

- The graphical notation helps to establish a better overview; to "see" the structure of the data.

  It is also useful for communicating with the future users. This notation was probably an important success factor.

- UML class diagrams are based on the same ideas.

  Plus methods. The ER-model describes only the static structure.

Entity-Relationship Model (4)

- The entity-relationship model is used only for the design, afterwards a schema transformation into the relational (or object-relational) model is done.

  There is no commercial entity-relationship DBMS. Object-oriented database management systems are similar, but for standard business applications they are seldom used. Of course, one can translate an ER-schema also to other data models, e.g. XML.

- Many variants/extensions of the ER-model have been proposed. Several different graphical notations are used.

  If you know one notation, it is easy to learn another one, since the basic concepts are the same.

Entity-Relationship Model (5)

- The ER-model is called a “semantic data model”, because it more closely resembles the real world than e.g. the relational model. For instance:
  - ◇ In the ER-model, persons are modelled. In the relational model, only names/numbers are found.

    The ER-model is an abstraction of the real world, whereas the relational model is an abstraction of files on a computer.
  - ◇ In the ER-model, there is a distinction between entities and relationships. In the relational model, both are represented by relations.

Entity-Relationship Model (6)

- Since ER-schemas can be translated into relational schemas, the expressiveness/semantical richness of the ER-model is not needed to satisfy the information requirements of the applications.
- But it makes the correspondence between the DB schema and the real world clearer (like a comment).
- Extended ER-models have e.g. specialization (subclasses), which is a very useful feature. The translation into the RM is possible, but often less elegant.

Entity-Relationship Model (7)

- (Unfair) comparison: C is translated into assembler, but one prefers to write programs in C.

  The comparison is unfair, because the language level difference between C and assembler is much greater than between the ER-model and the relational model. In the end, most entity types correspond to tables and vice versa. Conceptual models that are as high above the relational model as C is above assembler still have to be defined. Also portability is very important for C, whereas assembler depends on the machine type. A bit, this also appears in conceptual design, since an ER-schema does not depend on the features of a specific DBMS. But again, the effect is smaller than in the C-Assembler case (unless one also translates into OODBMS, XML, etc.).

Entity-Relationship Model (8)

- Since there is anyway no implementation, one can extend the ER-notation if necessary.
- However, the ER-notation acts as a communication tool between designers, programmers, customers.

  This is endangered if one does arbitrary changes to the notation.
- Of course, if one uses a CASE tool for managing ER-diagrams, one has to stick to the notation supported by the tool.

  Modern CASE tools have some support for user-defined extensions.

Overview

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.

CASE-Tools (2)

- There are special CASE-Tools for database projects (data modeling tools), e.g.
  - ◇ Oracle Designer,
  - ◇ CA ERwin,
  - ◇ Sybase PowerDesigner,
  - ◇ Embarcadero ER Studio,
  - ◇ DB-MAIN (LIBD Laboratory, REVER S.A.).
- A specialized graphical editor for ER-diagrams is a standard component of such tools.

CASE-Tools (3)

- Standard features of database CASE-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)

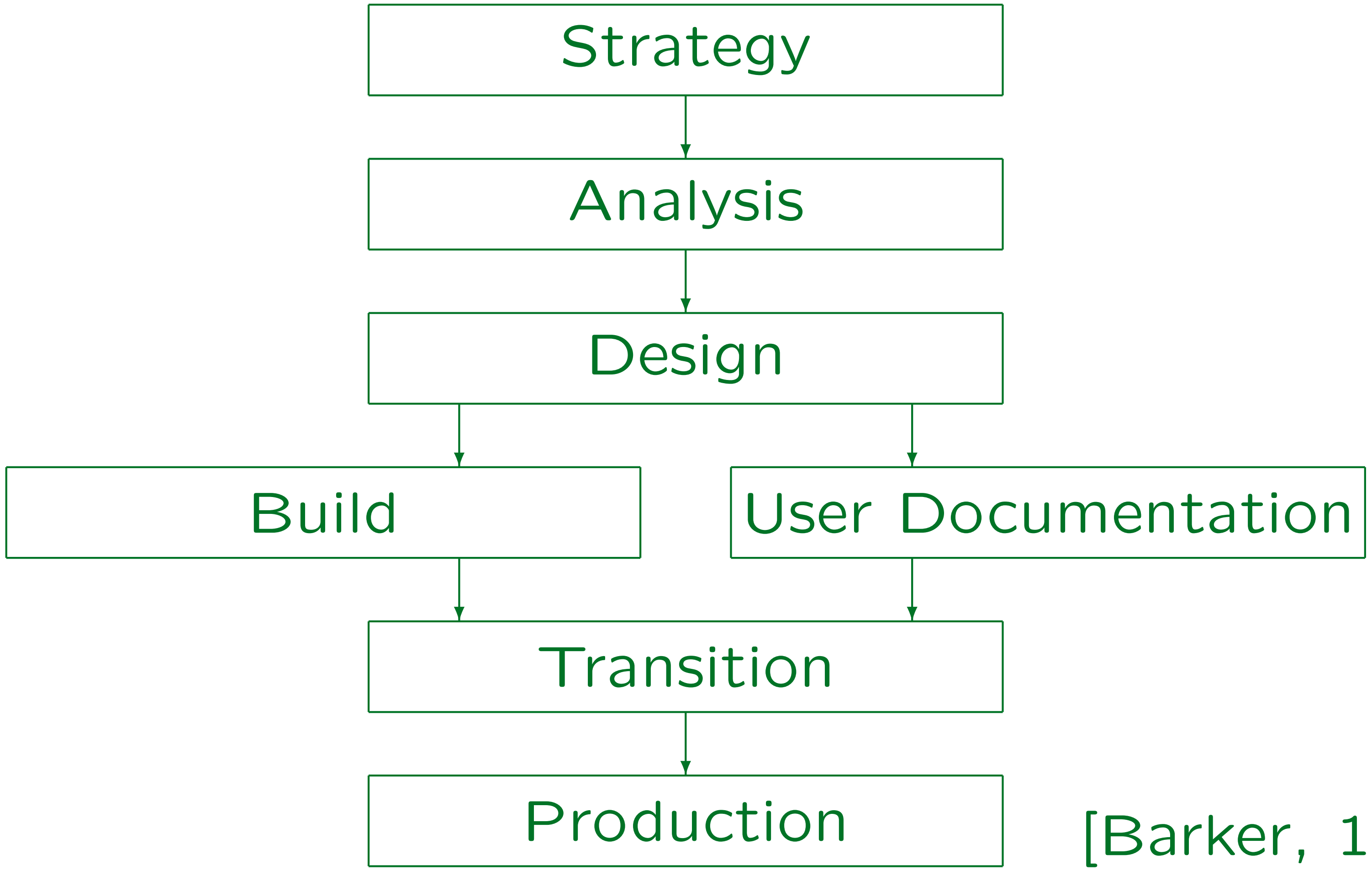
- The translation from an ER-schema to a relational schema is quite tedious if done by hand.

  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.

Overview

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 (≈ 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 <u>what</u> is needed, but do not yet think too much about <u>how</u> 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 table-spaces/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).

Transition Phase (4)

Gradual Transition, Continued:

- 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.

Overview

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.

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 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.

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 into the relational model, reverse engineering.
  - ◇ 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]

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.