Part 9: More Design Techniques

References:

- Batini/Ceri/Navathe: Conceptual Database Design. Benjamin/Cummings, 1992.
- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Ed., Section 16.2, "The Database Design Process"
- 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.
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German), Teubner, 1997.
- Koletzke/Dorsey: Oracle Designer Handbook, 2nd Edition. ORACLE Press, 1998, ISBN 0-07-882417-6, ca. $40.
- Paul Dorsey, Joseph R. Hudicka: Oracle8 Design Using UML Object Modeling. ORACLE Press, 1998, ISBN 0-07-882474-5, ca. $40.
- R. J. Muller: Database Design for Smarties — Using UML for Data Modeling. Morgan Kaufmann, 1999, ISBN 1-55860-515-0, ca. $40.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.

Objectives

After completing this chapter, you should be able to:

- write a short paragraph on design strategies (which describe a sequence of refinement or extension steps to develop an ER-schema), enumerate top-down schema transformations.
- enumerate possible sources for information that is the input to database design (requirements).
- start doing business interviews for collecting requirements (you still need a lot more experience).
- evaluate the view integration method.
- use look-up tables to make DB application systems easier to change, evaluate the use of generic/data-driven solutions.

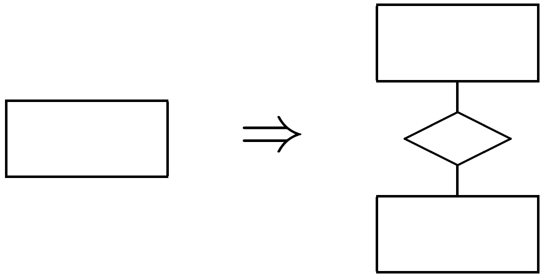
Overview

Design Strategies: Introduction

- "Building an ER schema is an incremental process: our perception of reality is progressively refined and enriched, and the conceptual schema is gradually developed."
  [Batini/Ceri/Navathe, 1992]
- The authors suggest to start with an initial schema, then change it by applying schema transformations, until the final schema is reached.
- They propose a set of primitive transformations, which they classify as top-down or bottom-up.
- A pure top-down approach would start with a single top-level entity and then successively split it by applying the top-down primitives T1–T8 listed on the following pages.

Top-Down Primitives (1)

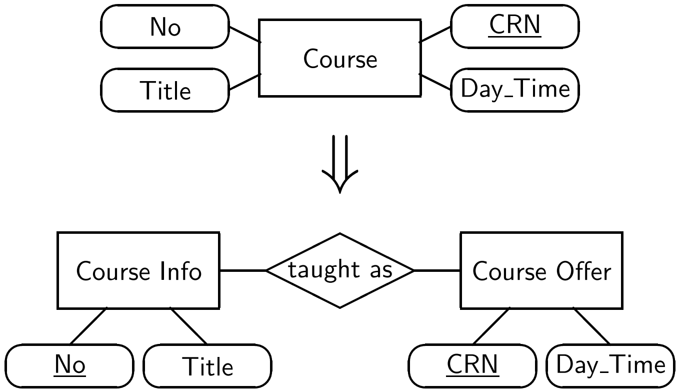
- T1: An entity can be split into two related entities:



- An example is given on the next page. However, attributes are usually developed later in the top-down strategy.

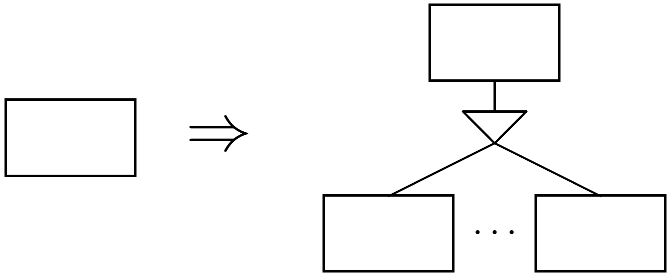
Top-Down Primitives (2)

- E.g. distinction between instances and general information:



Top-Down Primitives (3)

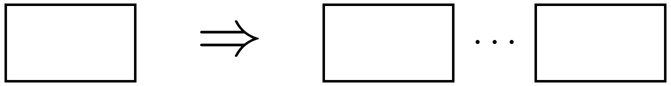
- T2: Introduction of subtypes (specialization):



- Relationships involving the original entity type can be linked to the supertype or to one or more of the subtypes.
  More than one subtype: Relationship is split (see T4).

Top-Down Primitives (4)

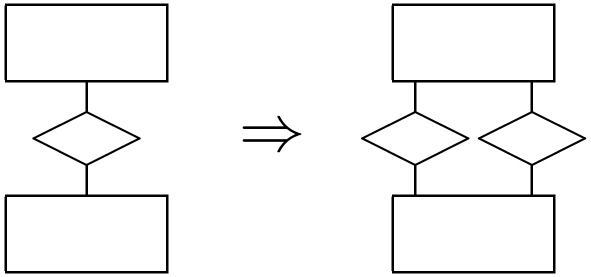
- T3: An entity type can be split into unrelated entity types:



- Normally, already the original entity type had relationships to other entity types.
- These relationships are then distributed among the new entity types.
- So normally one does not get an unconnected schema:
  No top-down primitive would allow to later connect two unrelated entities.

Top-Down Primitives (5)

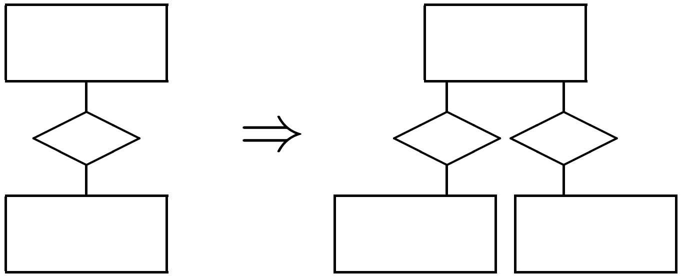
- T4: A relationship can be split into two parallel ones:



- E.g. one originally has "Person is related to City".
  Then one discovers that there are actually two different relationships: "born in" and "lives in".

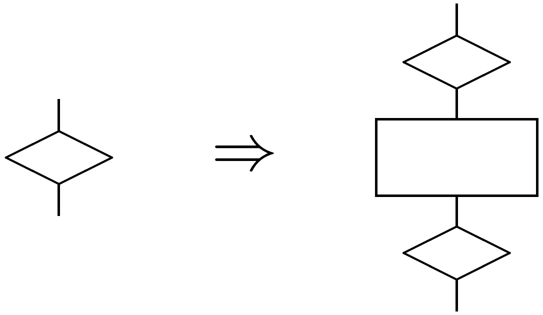
Top-Down Primitives (6)

- If T4 is applied followed by T3, this transformation results:



Top-Down Primitives (7)

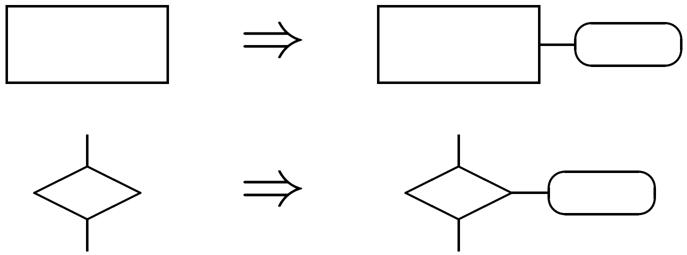
- T5: A relationship can be split by introducing an entity.



- E.g. one has the relationship "Customer orders product".
  Then one introduces the "order" entity type.

Top-Down Primitives (8)

- T6: Introduction of attributes for entities and relationships.



- T7: Introduction of structured attributes.
- T8: Refinement (splitting) of attributes.

Top-Down Schema Design

- So the general idea of top-down schema design is refinement:
  - One starts with a small schema containing very general concepts on a high abstraction level,
  - then details are introduced by adding attributes, introducing subtypes and splitting schema constructs.
- A pure top-down approach (which allows only the above primitives) is probably very difficult.
- However, it can be a useful documentation to show the schema afterwards at different abstraction levels.

  There are also formal techniques and tools for schema simplification/compression/condensation (to generate a birds eye view of an otherwise too complicated schema).

Bottom-Up Primitives

- B1: Generate a new entity.
- B2: Connect two entities by a new relationship.
- B3: Generalization (create a new supertype for existing subtypes).
- B4: Attribute aggregation (create an entity type which has existing attributes).
- A pure bottom-up development of a schema would
  - start with the set of all attributes,
  - then aggregate them to entities, use generalization, and connect the entities by relationships.
- So bottom-up means to stepwise extend, abstract, aggregate.

Inside-Out Strategy

- The inside-out strategy uses only the bottom-up primitives.
- It first produces a core schema for the most important or most evident concepts,
- and then moves outward by attaching more and more entities to the current schema.

  So one first adds the entity types that are conceptually close to the entities one already has.

Mixed Strategy

- The mixed strategy partitions the requirements into subsets, which are separately considered.
- It produces first a skeleton schema (bottom-up).
- Each entity in the skeleton schema can then be separately refined.
- The resulting schemas can then be integrated based on the skeleton schema.

  This may require a refinement of the relationships, too.
- In this way, we get an overview schema (the skeleton schema) and detailed schemas for different "modules".

Overview

Sources for Information

- Interviews
- Analysis of existing forms
  Filled out forms are more useful than empty ones.
- Analysis of reports which should be generated out of the DB.
- Analysis of existing files
- Analysis of existing manuals, descriptions, documentations (e.g. information for new employees, business reports).
- Partial reuse of published solutions ("reference models").
- Utilization of experiences of other companies.
- Standard-Software like SAP/R3 also comes with a DB schema.

Interviews (1)

Problems:

- You need to know the application domain (or learn about it).
- Your interview partners (experts of the application domain) might use technical terms which you do not know (yet).
- Some things might be so trivial or obvious for the application experts that they do not tell you about them.
  But they might not at all be obvious for you.
- The interview partners will tell you about the normal way things are done, but you need to know all exceptions.
  Ask specifically for exceptional/extremal situations.
- Sometimes it might be possible to do things differently than they are currently done.

Interviews (2)

Questions:

- Fishing: "Tell us something about instructors."
- Number: "How many instructors can teach a course?"
- Connections: "Do books have something to do with courses?"
- Without Questions: "Can somebody write an exam without being registered for the course?"
- Wandering: "What else is connected with a course?"
- Points of View: "What do your students think about this?"
- Active listening: Repeat what you understood with your own words and ask whether you understood it correctly.
  [List modelled after Table 6.2-2 in Rauh/Stickel, 1997.]

Interviews (3)

More Advice:

- Taping the interview would be useful, but not all people feel comfortable with this.

    You may need an assistant for note taking. Look at the notes soon when everything is still fresh in your mind.

- Ask about any documents, forms, copies of example data etc. you may collect. Go over example scenarios.
- There should be a short presentation (briefing) before the interview to increase the interest and remove fears.
- Barker [1990] distinguishes in-depth interviews (2–4 hours) and directional interviews (0.5–1 h, e.g. for management).

Interviews (4)

- Interviews take a lot of time. You should carefully select the interview partners (minimum number of people who together know all what is done now and needs to be done in future).

    Make sure that the important people feel involved.

- Develop rough ER-diagrams and function hierarchies etc. immediately after the interview.
- Many questions come up when you try to sort out your ideas and read the collected material after the interview is done.

    It might be necessary to schedule two interviews.

- There should be a feedback session where you present your integrated understanding to the group of interviewed people.

Overview

View Integration (1)

- Often, it is not possible to create the complete ER-Schema in one step, because this would be too large.
- Then one first creates small ER-schemas for each application or user. So one does not start with the conceptual schema level, but instead with the external views.
- After that, these schemas must be integrated to get the complete enterprise data model.
- The integration is a labor-intensive and mainly manual task, but the single views are at least a very useful documentation.

    It is also possible to really implement the relational version of the views later, based on the relational version of the integrated schema.

View Integration (2)

- In order to integrate schemas, one must find correspondences between schema elements. There may be conflicts:
  - Naming conflicts (synonyms, homonyms),

    Synonyms: Two entity types on different schemas denote the same real-world concept, but have different names. Homonyms: Two entity types have the same name, but denote different concepts.
  - type conflicts,

    The same concept is modelled as an entity on one schema and as an attribute on the other schema.
  - domain conflicts (different data types or units of measure),
  - conflicts among constraints (e.g. different keys).

View Integration (3)

- In part, the discovered conflicts will be removed from the view schemas before the integration itself is done.

  E.g. homonyms will later be confusing.
- However, not all conflicts need to be removed: It is only necessary that one understands which schema elements are needed in the integrated schema, so that the views can be defined based on that schema.
- There are different strategies for integration: E.g. one first takes two schemas that are similar (i.e. describe highly related / overlapping domains), then one integrates the result with another highly related schema and so on, until all schemas are integrated ("binary ladder integration").

Overview

Changes (1)

- Often the requirements will change. Then it will be necessary to update the database schema and/or the application programs.
- "The only programs which are never changed are those which are not used."
- "80% of the project cost goes into the maintenance, only 20% in the initial development."
- "Software ages and becomes unmaintainable after many programmers have applied patches to it without fully understanding the architecture/concept of the software."

  Maybe a change would have required to change the basic architecture of the software which was not done.

Changes (2)

- Personal experiences with the grades DB:
  - I forgot the possibility of half points.
  - I allowed as grades only "A+, A, A-, B+, B, B-".

    So I forgot "G". Later another instructor wanted to use the software for an ungraduate course, where "C", "D", and "F" grades are quite possible.
  - I allowed as exercise categories only homeworks, midterm exam, and final.

    This broke when I had two sessions of one course with two different midterm exams. The statistics needed to be computed over each exam version.
  - I could collect data from only a single course in the DB.

Changes (3)

- First, this is a sign of bad planning/design.
- Already during conceptual design, one should think about exceptional cases and extensions that may be required in the near future.
- Anticipated changes can be prepared and made simpler.
- Simple changes require to modify only a single, easy-to-identify place in the database or the program.

  E.g. inserting/deleting/updating a single tuple in the database, modifying a single constraint, modifying the value of a single constant in a parameter file for the application programs. Such anticipated changes such be documented in the manuals.

Changes (4)

- Even if only an attribute is added to a table, or a data type of an attribute is modified, this will require to change also application programs.
- The later a change is done, the more expensive it is.
- E.g. instead of hard-coding possible values in application programs and constraints, one can use lookup tables which contain all allowed values (see below).
- Or: Instead of modelling a a few questionaires each as their own table, you might want to model general questionaires and put the field names etc. in a table.
- Sometimes the more general solution is actually simpler.

Changes (5)

- However, there are limits:
  - If the requirements say that there are no half points, it is an error if the system allows them.

    One could have a parameter that can easily be switched.
  - If the schema becomes more complicated by anticipating possible changes, a cost/benefit analysis must be done.

    When the change does not occur, time and money was wasted. If does occur, money was saved, because choosing a more general design from the beginning is much cheaper than doing the change later when data and application programs already exist.
  - Most projects are anyway behind schedule/above budget.

Look-Up Tables (1)

- Values hard-coded as a constraint:

```
CREATE TABLE STUDENTS(...
             GRADE VARCHAR(2)
                   CHECK(GRADE IN ('A+','A',...))
```

- Look-up table (referenced with foreign key constraint):

```
CREATE TABLE STUDENTS(...
             GRADE VARCHAR(2)
                   REFERENCES VALID_GRADES));
CREATE TABLE VALID_GRADES(GRADE VARCHAR(2)
                                PRIMARY KEY);
INSERT INTO  VALID_GRADES VALUES('A+');
INSERT INTO  VALID_GRADES VALUES('A'); ...
```


Look-Up Tables (2)

- It is important that the application programs do not contain their own checks for valid grades.

  E.g. the program for entering grades. It is especially bad if the check is programmed multiple times in different programs.

- This would be another case of redundant information that requires changes in several places and can easily lead to inconsistencies.
- However, depending on the DBMS, the error message produced by an `INSERT` command with an invalid grade might not be helpful to end users.

  One should give the constraint a meaningful name. The DBMS will show the name of the violated constraint.

Look-Up Tables (3)

- A compromise is to do error checking in the program, but to use the values in `VALID_GRADES`.

  Then one can give a nice error message and still has defined the valid grades only at a single point (namely in the DB).

- Another solution to the duplication problem is to check for valid values only in the program, and not in the database.
- This is not a good solution:
  - After some time, other programs might be developed that also allow to enter grades. Then the test will be duplicated.
  - Some users might be allowed to access the database directly via SQL. Then they can enter invalid grades.

Look-Up Tables (4)

- Look-up tables can contain more information besides a list of valid values. E.g. lookup-table for exercise types:

EX_TYPES		
CODE	HEADLINE	OUTPUT_ORDER
H	Homeworks	1
M	Midterm Exam	2
F	Final Exam	3

- The `OUTPUT_ORDER` and the `HEADLINE` are used to print a report for each student that lists first his/her homework results, then the results for the midterm exam, etc.
- The `CODE` is referenced in the `EXERCISES` table.

Look-Up Tables (5)

- “Look-up tables” are in general tables that do not change during normal database operations.

  Changing them is a bit like changing the DB schema: The real data changes frequently, the lookup tables very seldom, and the schema hardly ever.
- Usually, look-up tables are not large.
- Changing look-up tables is easier than changing the database schema or changing application programs.

  Changes in the look-up tables are anticipated.
  They are a way to parameterize the program.
- Look-up tables can also contain texts that the program has to print (in order to make a translation simpler).

Expandable Attribute Sets (1)

- Hard-coded attributes of the students table:

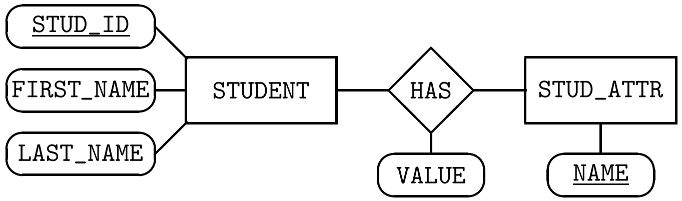
```
CREATE TABLE STUDENTS(
             STUD_ID NUMERIC(4) PRIMARY KEY,
             FIRST_NAME VARCHAR(20) NOT NULL,
             LAST_NAME VARCHAR(20) NOT NULL,
             ...)
```

- There is a lot of information that might be required about students besides these basic attributes.

  E.g. whether this is the last term (so G grades are impossible), the Oracle account, etc. In my German university, I needed the date and place of birth for printing them in certificates.

Expandable Attribute Sets (2)

- A solution is to have a table of possible student attributes and to store the attribute value in a relationship:



The relationship could also be called “STUD_EXTENSION”, “ADDITIONAL_STUDENT_DATA”, etc.

Expandable Attribute Sets (3)

- The additional attributes are listed in (a look-up table):

```
CREATE TABLE STUD_ATTR(
             NAME VARCHAR(10) PRIMARY KEY)
```

- The relationship is translated to:

```
CREATE TABLE STUD_EXTENSION(
             STUD_ID NUMERIC(4)
                     REFERENCES STUDENTS,
             ATTR_NAME VARCHAR(10)
                     REFERENCES STUD_ATTR,
             VALUE VARCHAR(20) NOT NULL,
             PRIMARY KEY(STUD_ID, ATTR_NAME))
```

Expandable Attribute Sets (4)

- In this way, all of the added attributes have the data type `VARCHAR(20)` (and all are optional).
- Most data can be stored as strings, but then again checks in programs are needed to enforce any constraints on the possible values.

  And the goal is to make the program independent of the attributes. Otherwise this effort would not be needed.
- Of course, one can also also define a schema for additional attributes that are of enumeration types, with the possible values for each attribute defined in a table.

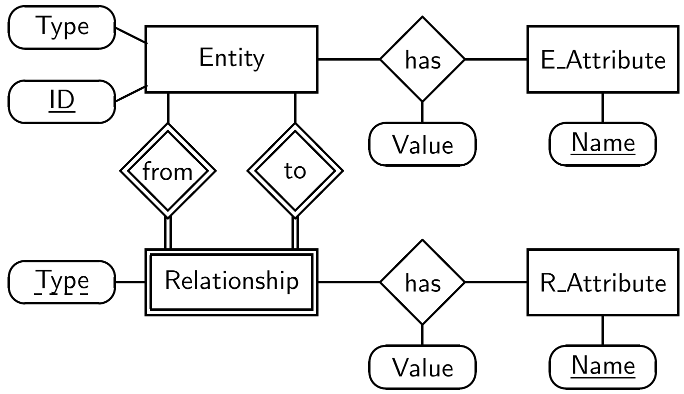
  One could have three tables, defining string-valued attributes, numeric attributes, and attributes of enumeration types.

Expandable Attribute Sets (5)

- Note that data-entry forms etc. must be dynamically created out of the attribute specifications in the database.

  This is quite a lot of work. For a normal table with a fixed set of attributes, tools like Oracle Developer Forms can automatically create an input form from the table declaration.
- The look-up table defining the attributes will normally also contain such information as the input field width and a number for printing the fields in the correct sequence, etc.
- Positive: Changes of the attributes are easy and can even be done by somebody without access to the source code.
- Positive: If the set of attributes is large, the generic program might actually be simpler.

Extreme Case



The entity type and the relationship type could also be made entity types in this meta schema.