Datenbanken II A: DB-Entwurf

Chapter 2: ER-Diagrams II: Weak Entities, Subtypes

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

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

Objectives

After completing this chapter, you should be able to:

- enumerate the ER-constructs supported by Oracle SQL Developer Data Modeler.
- draw ER-diagrams in the graphical syntax of Oracle SQL Developer Data Modeler (“Barker Notation”).
- You should also be able to read such diagrams.
- explain the difference between the global DB schema and the views contained in single diagrams.

Contents

Domains (1)

- Often different attributes should have the same data type, i.e. especially the same length. E.g.:
    - Years: Year an instructor got tenure, Year a course is offered, Year a student was admitted, etc.
    - URLs: Links to homepages of courses, instructors, departments.
    - Last Names: Of students, instructors, staff.
- It would be strange if
    - some years are stored with two digits, others with four, or
    - student names can be longer than instructor names.

Domains (2)

- Characteristics such as the maximal length of all kinds of URLs should be defined only once.
- This ensures greater consistency in the schema, especially when later changes are done (e.g. attribute length increases).
- In Oracle Designer, one can define data types of columns indirectly via domains:

$$\begin{tikzcd}
\begin{array}{c}\text{Column}\\ \text{“Homepage”}\end{array} \arrow[r] & \begin{array}{c}\text{Domain}\\ \text{“URL”}\end{array} \arrow[r] & \begin{array}{c}\text{Data Type}\\ \text{“}\texttt{VARCHAR(80)}\text{”}\end{array}
\end{tikzcd}$$

Domains (3)

- One first defines a domain and then assigns this domain to one or more attributes.

  Instead of directly defining the data type details for the attributes. That would have to be done for each attribute separately, while with the domain the details are defined only once and used in possibly many attributes. In Oracle Designer, domains are defined under "`Edit` $\rightarrow$ `Domains`".

- If a domain definition changes, one can propagate this change to all attributes having this domain.

  In Oracle Designer, this is done only semi-automatically. One must call "`Utilities` $\rightarrow$ `Update Attributes in Domain`".

Domains (4)

- Different domains may have same data type.
- E.g. last names of customers and names of cities may both be `VARCHAR(20)`, but it makes no sense to compare them. Different domains should be used.

  One should consider attributes of different domains as uncomparable (unless declared as subtype).
- A domain can be seen as a "shorthand" for a standard data type, but with a specific meaning, different from other domains.

Domains (5)

- Domains can be used to capture the information which attributes should be comparable.

  This requires logical domain names, e.g. `CITY`, not `VC20`.

- The SQL-92 standard has a similar notion of domains (without the restriction that columns of different domains cannot be compared).

  This is not implemented in Oracle 8. But when domains are defined in the Designer, they might be partially mapped to SQL domains in other DBMS. Oracle 8 has PL/SQL types which could also be used. But for consistent schema changes, it is already helpful that they are supported in the Designer.

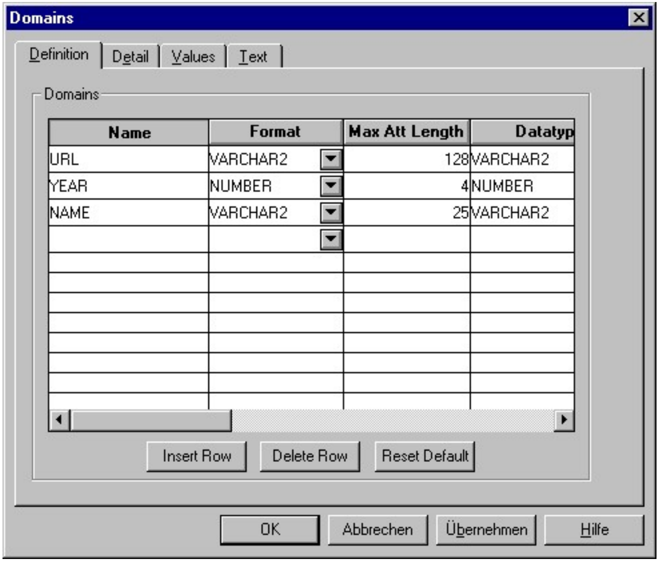
Domains (6)

- Domain names can often be used as attribute names. This makes joinable attributes very clear.

- Some designers have a set of standard domains, which they always use.

  E.g. names of length 10, 20, 40, descriptions of size 2000, email/URL of size 250, ZIP codes, SSN, boolean values, etc. Selecting from a set of predefined standard domains can be done faster than considering every attribute in isolation. In some projects, only a "domain administrator" is allowed to create a new domain.

- However, this at least partially contradicts the idea of logical domains.


Domains

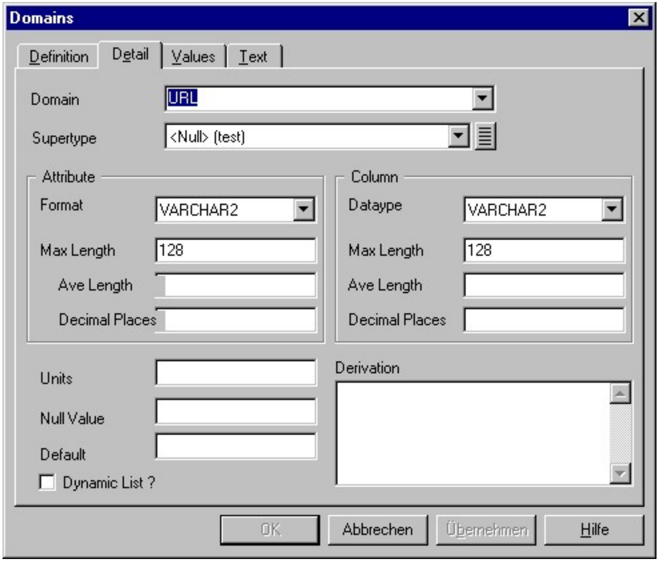
Definition | Detail | Values | Text

Domains

Name	Format	Max Att Length	Datatyp
URL	VARCHAR2	128	VARCHAR2
YEAR	NUMBER	4	NUMBER
NAME	VARCHAR2	25	VARCHAR2

Domains (8)

- The dialog box for defining domains has four tabs:
    - “Definition”: list of all defined domains.
    - “Detail”: one page per domain.
    - “Values”: for defining enumerated types etc.
    - “Text”: contains descriptions, notes, etc.
- In principle, the same parameters can be set as in the attribute definitions.
- Whether an attribute is optional and whether it is part of the key can only be defined in the entity definition dialog.



Domains

Definition | Detail | Values | Text

Domain	URL
Supertype	<Null> (test)

Attribute

Format	VARCHAR2
Max Length	128
Ave Length	
Decimal Places	

Column

Dataype	VARCHAR2
Max Length	128
Ave Length	
Decimal Places	

Units	
Null Value	
Default	

☐ Dynamic List ?

Derivation

Domains (10)

Format/Attribute vs. Datatype/Column:

- A domain definition contains information at the ER-level (Format) as well as about the implementation (Datatype).

  The documentation also mentions that the datatype can also be used for application program variables, but then it depends on the programming language. The type system of languages like C are quite different from the SQL type system.

- E.g. “`IMAGE`” can be selected on the conceptual level, but it is implemented as a “`BLOB`”.

  `BLOB`: “Binary Large Object”. The Datatype selector contains all datatypes of Oracle as well as some types from other systems.

Domains (11)

Dynamic List:

- If this is checked, the possible values will be retrieved at runtime from a lookup table.

  This makes it easy to change the possible values of the enumeration type later: One can simply insert a new value into the lookup table.

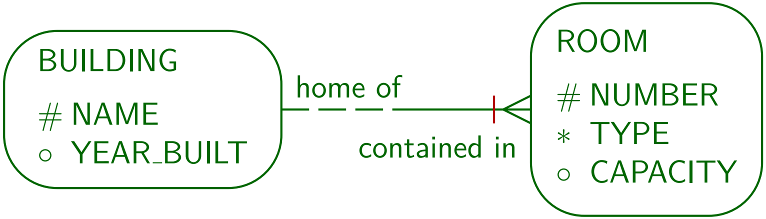
- Otherwise, they will be hardcoded (e.g. as `CHECK`-constraint in the `CREATE TABLE` statement).

  While an `ALTER TABLE` statement to change the constraint is not too difficult (but not that several attributes in different tables might have to be changed), the possible values might also be hardcoded in application programs.

Contents

Weak Entity Types (1)

- If a relationship contributes to the identification, there is a bar across the connecting line:



- A room is identified by building and room number, e.g. “Crawford Hall 169”.

Weak Entity Types (2)

- Different buildings of the university can have rooms with the same number.
- When translated into tables, the key of the `ROOMS` table will be composed from the building name and the room number.

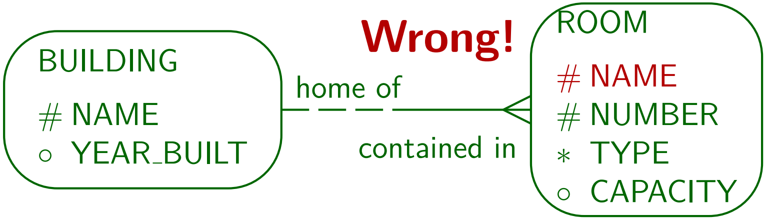
  The building name in addition will be a foreign key that references the `BUILDINGS` table.

- Entity types that must borrow key attributes from other entity types are called “weak entity types”.

  In the Oracle Designer documentation, this name is not used. One simply declares a relationship as part of a UID for an entity type.

Weak Entity Types (3)

- It would be a bad design to explicitly replicate the key attribute of the referenced table:



- Now the constraint is needed that a room with name X is always related to a building with name X, so that the relationship is actually redundant.

Weak Entity Types (4)

- In general, advanced constructs in the ER-model are often introduced in order to avoid certain common kinds of constraints.

  Or at least to specify these constraints graphically instead of as text and permit a special implementation. If one would translate the above schema where name and number are explicitly defined as key attributes into the relational model, one would get two copies of "Name": A second copy is introduced as foreign key in order to implement the relationship (see below). Now with the constraint it becomes clear that the two copies can be merged.

- Weak entity types are often used in master-detail relationships, e.g. for an invoice and its line items.

Weak Entity Types (5)

- A weak entity type is existence-dependent on its parent entity type (BUILDING in this case): If a building is sold and removed from the database, all rooms in it should be automatically removed.

  For weak entity types, it is quite typical that in the resulting relationional schema "`DELETE CASCADES`" is defined for the foreign key that implements the relationship.

- It is often a design decision how one selects a key: If rooms have a number that it unique over all buildings, the "Room" entity type is no longer weak.

Weak Entity Types (6)

- One can use a relationship for identification only if there is at most one related entity (cardinality $(1, 1)$ or $(0, 1)$):
  - On the many side of a one-to-many relationship.
  - On both sides of a one-to-one relationship.

  If there were several related entities, one would need set-valued attributes (not supported in the standard relational model).

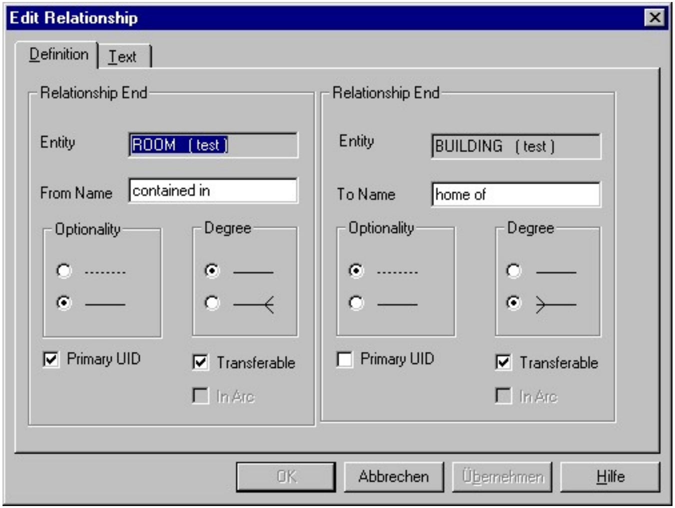
- At least for primary keys, the participation in the relationship must be mandatory.

  Primary key attributes cannot be null.

Weak Entity Types (7)

- There are two places to specify that a relationship contributes to the identification of the entity:
  - In the entity definition, tab UIDs.
  - In the relationship definition.

    In the "Edit Relationship" dialog box, one can also change the optionality (minimum cardinality) and degree (maximum cardinality) for each end, change the role name, and store a description or notes for each relationship end.



Relationship End

Entity: ROOM (test)

From Name: contained in

Optionality: ○ ········ [✓ ● ———]

Degree: [✓ ● ———] ○ ——<

[✓] Primary UID

[✓] Transferable

[ ] In Arc

Relationship End

Entity: BUILDING (test)

To Name: home of

Optionality: [✓ ● ········] ○ ———

Degree: ○ ——— [✓ ● >——]

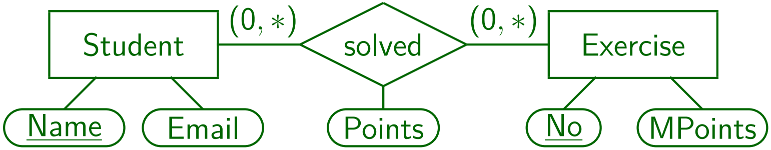
[ ] Primary UID

[✓] Transferable

[ ] In Arc

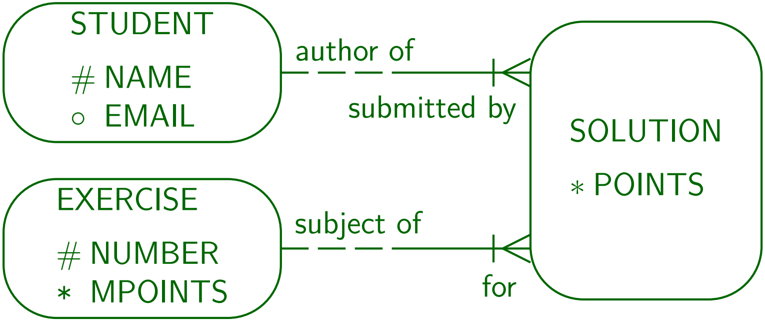
Association Entity Types (1)

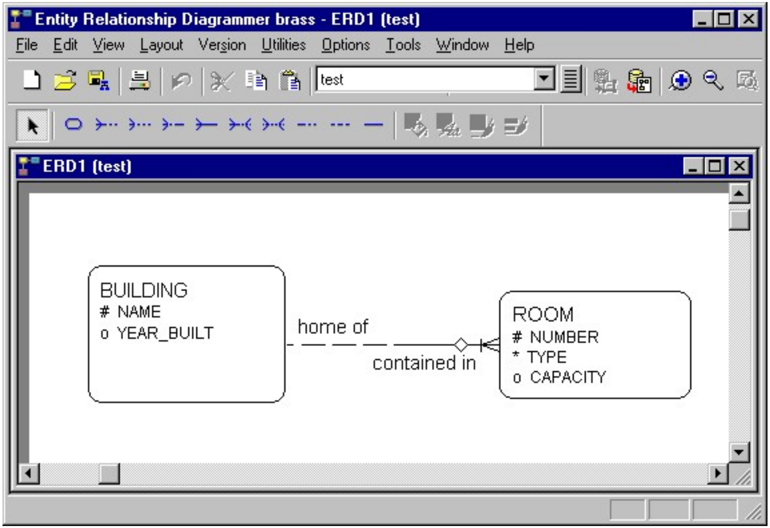
- Weak entity types can have more than one parent.
- Weak entity types with two (or more) parent types are sometimes called "association entity types", because they are similar to a kind of relationship between the parent entity types.
- E.g. suppose that we need a relationship attribute:



Association Entity Types (2)

- Oracle Designer does not support relationship attributes. However, one can turn the relationship into an association entity type:

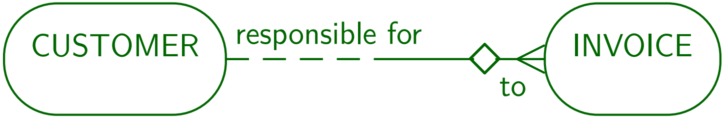



Entity Relationship Diagrammer brass - ERD1 (test)
File Edit View Layout Version Utilities Options Tools Window Help
test
ERD1 (test)
BUILDING
# NAME
o YEAR_BUILT
home of
contained in
ROOM
# NUMBER
* TYPE
o CAPACITY

Contents

Fixed Relationships (1)

- A relationship can be marked as non-transferable:



- In this way, an invoice cannot be disconnected from a customer and connected to another customer.
- I.e. the foreign key attribute (customer number in the invoice) is non-updatable.

Oracle Designer allows the "non-transferable" sign also on the other side of the relationship. Semantics unclear (?).

Fixed Relationships (2)

- It might be a good idea to collect non-updatability information for arbitrary attributes, but Oracle Designer does not allow that.

  However, one can extend it in this way. It also has cross-referencing tools which show CRUD (create, retrieve, update, delete) information for all entities based on the business functions.

- Non-Updatability is a simple kind of dynamic constraint, which refers not to single database states as a normal static constraint, but to pairs of DB states.

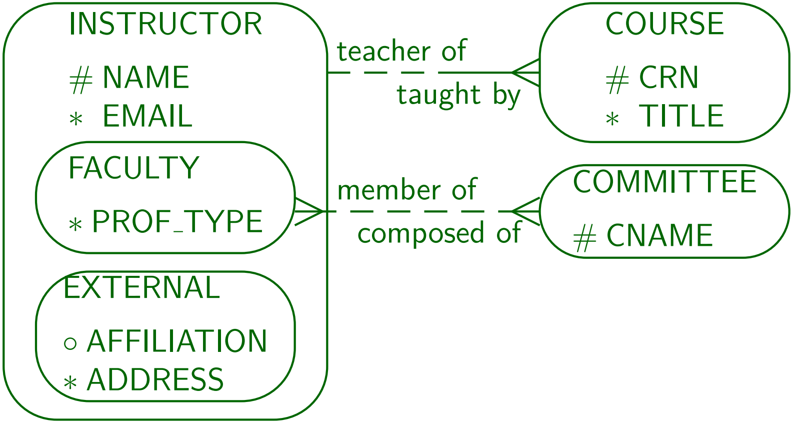
  Another example is "Salaries cannot decrease."

Contents

Specialization (1)

- Two or more entity types may have attributes or relationships in common.
- Then it might be useful to create a generalized entity type, which contains only the common characteristics, and abstracts from the differences.
- Or, some attributes or relationships may apply to only a subset of the entities. Then creating a specialized entity type for this set should be considered.
- Inheritance ("is-a" relationships) and subclasses are also a useful feature of object-oriented languages.

Specialization (2)



Specialization (3)

- In the above example:
  - Instructors are faculty members (i.e. long-term employees of the university) or external teachers which are paid for the course only.
  - For all instructors, name and email address is stored.
  - For faculty members, in addition the professor type (Assistant, Associate, Full) is stored.
  - For external instructors, their affiliation and address is stored.

Specialization (4)

- Example, continued:
  - Both types of instructors can teach courses.
  - Only faculty members can serve on committees.
- In general, specialization can be distinguished in:
  - Disjoint: It is not possible that an instructor is at the same time a faculty member and an external teacher. Oracle Designer only supports this case.
  - Overlapping: Objects of the superclass can be in more than one subclass at the same time (then they do not have a unique type: uncommon).

Specialization (5)

- Specialization can also be:
  - Total: Every instructor must be a faculty member or an external teacher.
  - Partial: Elements of the superclass are not necessarily contained in one of the subclasses.
- Oracle Designer normally uses total specialization (but one can always create an “Other” subclass).
- However, when one generates tables, one can also select that the supertype is instantiable (meaning partial specialization).

Specialization (6)

- Total and disjoint specialization means that the set of entities of the superclass is partioned into the instances of the subclasses.

  It is very difficult to find information like "Oracle Designer supports only non-overlapping and total specialization" in the documentation. E.g. it is not mentioned in the online help, the manuals are anyway either too short or only interface lists, and books like the Oracle Designer Handbook assume that you know ER-modelling. Only the book by Barker clearly states this. Looking at the translation into tables also shows that a non-overlapping and total specialization is assumed. I later learnt about the option for the Database Design Transformer which gives partial specialization, but this is anyway the wrong place: If such an option is really to be used, it must be offered in the ER-Diagrammer.

Specialization (7)

- Subclasses can themselves have subclasses.

  In the ER-Diagrammer, one creates a subclass by creating an entity type within the boundaries of another entity type (the superclass).

- In general, one can create a tree of entity types.

  When specialization is total, real instances only exist at the leaves of the tree. All other classes in the tree simply have the union of the members of their subclasses.

- Multiple inheritance is not supported in Oracle Designer.

  Multiple inheritance means that an entity type has more than one superclass, and inherits attributes and relationships from all of them.

Specialization (8)

- It makes no sense to define primary key attributes for a subclass: All attributes and the key constraint are inherited from the superclass.

  If the key uniquely identifies all members of the superclass, it especially uniquely identifies the members of the subclass.

- Of course, it is possible to declare additional (secondary) keys for the subclasses.

Specialization and Null Values

- In principle, one could avoid optional attributes with specialization: E.g. "Instructor" has a subclass "Instructor with Home Phone Number".
- When there are n attributes that can independently be null, one would need 2^n subclasses. Then this method is clearly not useful.
- However, when there is a group of attributes which can only be together null, or together not null, one should consider using a subclass.

Constructs like specialization reduce the need for constraints.

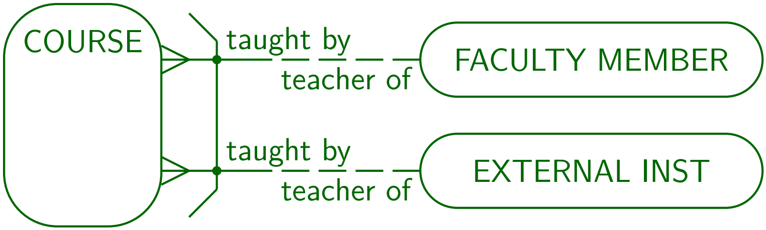
Generalization

- The specialization process starts with the superclass, discovers that some attributes apply only to a subset of the entities, and constructs subclasses.
- Vice versa, in generalization the subclasses are identified first, and then the discovery of common attributes leads to a superclass. The result is identical.
- Some authors use the term generalization or categorization if the subclasses have keys of their own, and their union should be considered, e.g. for defining a relationship.

Contents

Arcs (1)

- By linking two or more relationships with an arc, one can specify that they are mutually exclusive:



- I.e. a course is either taught by a faculty member or by an external instructor, but not by both.

Arcs (2)

- This is similar to defining two subclasses of courses:
  - Courses that are taught by a faculty member.
  - Courses taught by an external instructor.
- Alternatively, this corresponds to a generalization of faculty members and external instructors.

  One would use this model e.g. if external instructors and faculty members already have different keys of their own, and there is no natural key for their generalization. This is not a good example: The name or SSN would do. The classical example in the literature are invoices which can be sent to persons or companies.

Arcs (3)

- In general, arcs might help when for various reasons specialization is too restricted.
- Using arcs in the ER-Diagrammer is a bit tricky.

  Arcs are created by selecting at least two relationship ends and then clicking on the “create arc” symbol in the toolbar (or the Utilities menue). You must select the relationship ends, not the relationships (click on the role names). Use `Ctrl`-click to select the second end.
  In order to remove a relationship from an arc, select the arc by clicking on the line between the two relationships (this is a bit difficult). Then select the relationship end(s) you want to remove and select “Remove from Arc” on the toolbar or the Utilities menu. If an arc remains only for one relationship, I do not know how to select it. In this case, use the Repository Object Navigator, drill down to the relationship, and delete the “1” in the field “In Arc”.

References

- Barker: CASE*Method, Entity Relationship Modelling.
  Addison-Wesley, 1990, ISBN 0-201-41696-4, ca. $61.
- Koletzke/Dorsey: Oracle Designer Handbook, 2nd Edition.
  ORACLE Press, 1998, ISBN 0-07-882417-6, ca. $40.
- A. Lulushi: Inside Oracle Designer/2000.
  Prentice Hall, 1998, ISBN 0-13-849753-2, ca. $50.
- Oracle/Martin Wykes: Designer/2000, Release 2.1.1, Tutorial.
  Part No. Z23274-02, Oracle, 1998.
- Heli Helskyaho: Oracle SQL Developer Data Modeler for Database Design Mastery.
  McGraw Hill Education / Oracle Press, 2015, ISBN 0071850090, 336 pages.
- Teorey: Database Modeling & Design, 3rd Edition.
  Morgan Kaufmann, 1999, ISBN 1-55860-500-2, ca. $32.
- Elmasri/Navathe: Fundamentals of Database Systems, 2nd Ed.,
  Appendix A, "Alternative Diagrammatic Notations".
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German), Teubner, 1997.