Part 6: More ER-Constructs

References:

- Teorey: Database Modeling & Design, 3rd Edition. Morgan Kaufmann, 1999, ISBN 1-55860-500-2, ca. $32.
- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Ed.
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German), Teubner, 1997.
- Kemper/Eickler: Datenbanksysteme (in German), Oldenbourg, 1997.
- Graeme C. Simsion, Graham C. Witt: Data Modeling Essentials, 2nd Edition. Coriolis, 2001, ISBN 1-57610-872-4, 459 pages.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.

Objectives

After completing this chapter, you should be able to:

- explain multivalued and structured attributes and their relation to weak entities.
- use n-ary relationships correctly
- explain the problems with cardinality specifications for ternary and higher relationships

Overview

1. Multivalued and Structured Attributes
2. Ternary Relationships

General Remarks

- Many additional constructs have been proposed for the Entity-Relationship Model.
- As mentioned above, one goal is to avoid general constraints (e.g. with weak entities or subclasses).
- Another goal is to have a more compact notation for common situations.

  If one wants to develop large database schemas with the ER-Model, it is of course better to use abbreviations. The translation of the classical ER-model to the relational model is more or less one-to-one (except that for many-to-many relationships an extra table is needed). Now some constructs are presented where the ER-Schema is smaller (counting the number of entity types) than the relating relational model.

Multivalued Attributes (1)

- A multi-valued attribute contains a set of attribute values, e.g.

- In the example, more than one phone number can be stored for a customer.
- However, the set can also be empty.

Multivalued Attributes (2)

- In the relational model, attribute values must be atomic.

  Actually, the relational model can be seen as more general, but 1NF is usually assumed.

- Then an additional table is needed for the phone numbers:

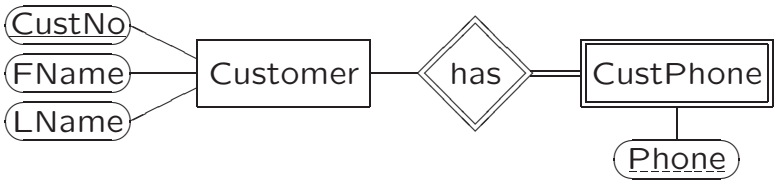
  `Customer(`<u>`CustNo`</u>`, FName, LName)`
  `Customer_Phone(`<u>`CustNo`</u> $\rightarrow$ `Customer,` <u>`Phone`</u>`)`

- This is also the result of translating the weak entity type on the next slide.

  I.e. the two constructs are equivalent.

Multivalued Attributes (3)

- If multivalued attributes are not available, a weak entity must be chosen in this case:



- Obviously, this looks much more complicated.

  The auxillary weak entity distracts the view from the really important part. Furthermore, if the ER-diagram is displayed without attributes, the multivalued attribute is suppressed, whereas the weak entity is shown.

Multivalued Attributes (4)

- Even without special ER-constructs, one can permit arbitrary data types for the attributes in the ER-model.
- Then a data type "set of integer" can be used.
- The translation into the relational model remains the same.

  Uncommon data types need more work during the logical design.

- But now data structures are not only described on the level of the data model, but also on the level of data types (problem for query language).

Multivalued Attributes (5)

- If the implementation DBMS does not only support classical 1NF relations, but has also objectrelational features, there might be a data type constructor for sets or lists of values.
- When the multivalued attribute or a special set datatype is used in the ER-design, it is obvious that during logical design, such a construct can be used.
- With the weak entity, this is less obvious.

  Of course, it would be legal. But a more detailed analysis of the schema is needed to discover this situation.

List-valued Attributes (1)

- If the sequence of the phone numbers is important, the attribute is list-valued.

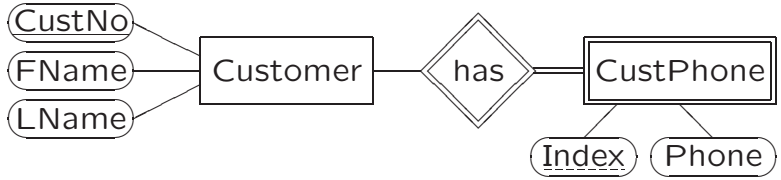
- Translation into the relational model:

  `Customer(`<u>`CustNo`</u>`, FName, LName)`
  `Customer_Phone(`<u>`CustNo`</u> $\rightarrow$ `Customer,` <u>`Index`</u>`, Phone)`

  Note that `Index` is most probably a reserved word in SQL.

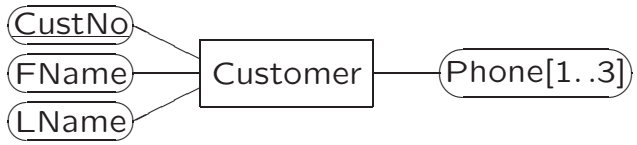
List-valued Attributes (2)

- This situation (sequence of phone numbers for each customer) can be expressed with a weak entity as follows:



List-valued Attributes (3)

- It is also possible to restrict the size of the list, i.e. to use an array.
- For example, each customer has between 1 and 3 phone numbers:



List-valued Attributes (4)

- If one uses the same translation into the relational model as before, one can restrict the index to the range 1..3 with a check constraint:

  `Customer(`<u>`CustNo`</u>`, FName, LName)`
  `Customer_Phone(`<u>`CustNo`</u> $\rightarrow$ `Customer,` <u>`Index`</u>`, Phone)`

- Now general constraints are needed to ensure that
  - ◇ Every customer really has at least one phone number (i.e. every `CustNo` appears in `Cust_Phone`).
  - ◇ The values for `Index` start at 1 and are sequential without holes (i.e. if $i > 1$ appears, $i - 1$ also appears).

List-valued Attributes (5)

- Since the maximal number of values is not large in this example, one can also use multiple columns directly in the `Customer` table:

  `Customer(`<u>`CustNo`</u>`, FName, LName,`
  `Phone1, Phone2`^{o}`, Phone3`^{o}`)`

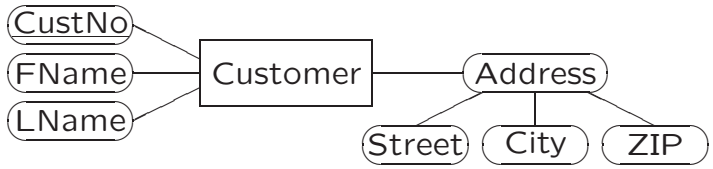
- Now one only needs a constraint that if `Phone3` is defined, `Phone2` must be defined, too:

  `CHECK(Phone3 is null or Phone2 is not null)`

- Queries for a specific phone number become more difficult (one could define a view).

Structured Attributes (1)

- Structured attributes consist of several components (like records in Pascal or structs in C):



- The components can themselves be structured.

  In general, a tree is possible.

- In the diagram, one can collapse the structure and show only the attribute "Address".

Structured Attributes (2)

- If an entity type has many attributes, this construct helps to improve the readability of the diagram by structuring the set of attributes into meaningful larger units.
- The alternative without structured attributes is:

Structured Attributes (3)

- The translation into the relational model is the same, i.e. one creates a column for each leaf in the component tree of the structured attribute:

```
Customer(CustNo, FName, LName,
         Address_Street, Address_City,
         Address_ZIP)
```

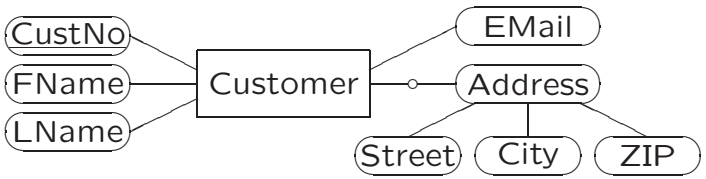
- Only the column names document the structure.

  Of course, one could choose an abbreviation, e.g. "`Addr_`".

- With more general data types in object-relational DBMS, a single column with a row type is possible.

Structured Attributes (4)

- An additional advantage is that one can make sure that certain attributes (the components of a structured attribute) can be null only together:



- Now the address is either completely undefined (all three components are null) or completely defined.

Structured Attributes (5)

- The relational translation now needs the constraint:

```
CHECK(Address_Street IS NULL AND
      Address_City IS NULL AND
      Address_ZIP IS NULL
      OR
      Address_Street IS NOT NULL AND
      Address_City IS NOT NULL AND
      Address_ZIP IS NOT NULL)
```

- Of course, it is also possible to mark only single components as optional.

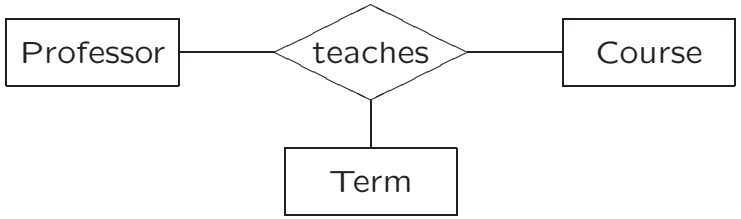
  By using the circle on the line connecting the component to the structured attribute.

Overview

1. Multivalued and Structured Attributes
2. Ternary Relationships

Ternary Relationships (1)

- Oracle Designer (and many other CASE tools) permit only binary relationships, where each instance relates only two entities.
- However, in general, any number of entity types can participate in a relationship:



Ternary Relationships (2)

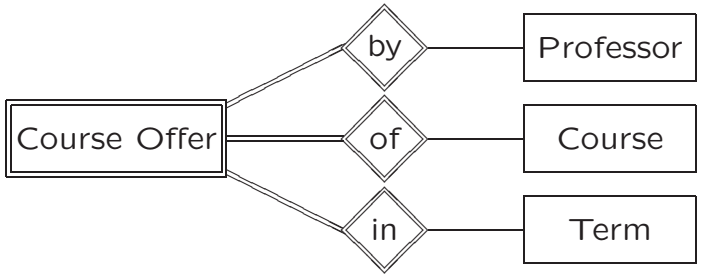
- Relationships can be described as sentences with holes/parameters, e.g.

  Professor P teaches course C in term T.

- This shows again that relationships correspond to predicates in logic.
- Thus, a database state interprets a ternary relationship as a set of triples. In the example, each triple consist of a professor object, a course object, and a term object.

Ternary Relationships (3)

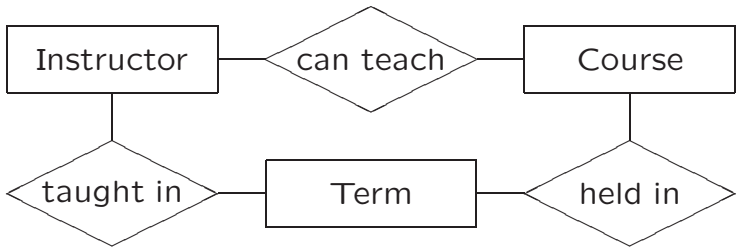
- Without ternary relationships, one must use an association entity:



- Clearly, this looks more complicated than the solution with the ternary relationship.

Ternary Relationships (4)

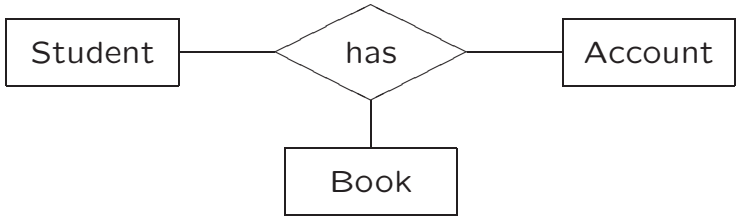
- Three binary relationships do not contain the same information as one ternary relationship.



E.g. instructors A and B both can teach courses 1 and 2. In the fall term, A taught 1 and B taught 2. In the spring term, they switched: A taught 2 and B taught 1. Then the exact course assignment cannot be uniquely reconstructed from the information in the above database.

Ternary Relationships (5)

- On the other hand, beginners often use ternary relationships when rather two binary relationships would be correct:



A student can have a computer account without having lended a book and vice versa. Moreover, when there are multiple books and accounts, which should be related? The relationships are independent.

Cardinalities (1)

- There are basically two methods for specifying cardinalities: Near entity type E, they specify an interval for the number of
  - ◇ relationship instances, in which a single entity of type E participates (Method I).

    This is the method we used in the course "Databases I".
  - ◇ entities of type E that can have a relationship to a fixed selection of entities for the other entity types (Method II).

    This is the method used in UML.

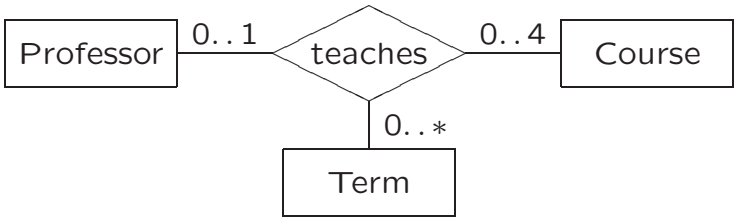
Cardinalities (2)

- For binary relationships, the two methods are equally powerful, only the intervals on the two sides are exchanged.
- However, for ternary (and higher) relationships, none of the two methods can express all restrictions that the other method can express.

  I.e. the two methods are not equivalent and none is more powerful than the other.

Cardinalities (3)

- Example: Method II (like UML)



  - ◇ Different professors cannot teach the same course in the same term.
  - ◇ A professor can teach a course 0 or more times.
  - ◇ A professor teaches at most 4 courses per term.

Cardinalities (4)

- In the above situation, professors, courses, terms each participate arbirarily often in the relationship.
- Therefore, with Method I, the cardinality specification is simply $(0, *)$ on every edge.
- This method cannot express the given restrictions.
- Vice versa, Method II cannot express the restriction that
  - ◇ Every course is offered at least once.
  - ◇ There are at most 20 course offers per term.