Teil 5: Einführung in logisches Design

Literatur:

- Elmasri/Navathe:Fundamentals of Database Systems, 3. Auflage, 1999. Chapter 3, "Data Modeling Using the Entity-Relationship Model"
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3. Auflage, Ch. 2, "Entity-Relationship Model".
- Ramakrishnan: Database Management Systems, Mc-Graw Hill, 1998, Ch. 14, "Conceptual Design and the ER-Model"
- Kemper/Eickler: Datenbanksysteme, Ch. 2, Oldenbourg, 1997.
- Rauh/Stickel: Konzeptuelle Datenmodellierung, Teubner, 1997.
- Teorey: Database Modeling and Design, 3. Auflage, 1999.
- Barker: CASE*Method, Entity Relationship Modelling, Oracle/Addison-Wesley, 1990.
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.

Lernziele

Nach diesem Kapitel sollten Sie folgendes können:

- Ein gegebenes Entity-Relationship-Diagramm in das relationale Modell übersetzen.

  D.h. ein äquivalentes relationales Datenbank-Schema ermitteln (einschließlich Schlüssel und Fremdschlüssel).

- Erklären welche Konstrukte (Kardinalitäten) nicht direkt übersetzt werden können.
- Typische Strukturen (wie viele-zu-viele-Beziehungen) in relationalen Datenbank-Schemen wiedererkennen.

Inhalt

1. Ziele des logischen Design
2. Basis ER-Konstrukte
3. Schwache Entities
4. Eins-zu-Eins-Beziehungen
5. Letzte Schritte, Einschränkungen

Generelle Bemerkungen (1)

- Um ein relationales Schema zu entwickeln, entwirft man zunächst ein ER-Diagramm, und transformiert es dann in das relationale Modell, da das ER-Modell
  - ◇ eine bessere Dokumentation der Beziehung zwischen dem Schema und der realen Welt erlaubt.

    Z.B. Entity-Typen und Relationships kennzeichnet.
  - ◇ eine nützliche graphische Notation hat.
  - ◇ Konstrukte wie Vererbung beinhalt, für die es keinen Gegenspieler im relationalen Modell gibt.

    Das schwierige konzeptionelle Design kann etwas erleichtert werden, wenn man zunächst die erweiterten Möglichkeiten verwendet.

Generelle Bemerkungen (2)

- Für ein gegebenes ER-Schema S_E soll ein relationales Schema S_R entwickelt werden, sodass es eine eins-zu-eins-Übertragung τ zwischen den Zuständen für S_E und S_R gibt.

  D.h. für jeden möglichen DB-Zustand bezogen auf S_E gibt es genau einen Zustand bezogen auf S_R, und umgekehrt.

- Zustände, die im relationalen Schema möglich sind, aber im ER-Schema bedeutungslos, müssen durch Integritätsbedingungen ausgeschlossen werden.

  Z.B. können Relationships im ER-Modell nur zwischen bereits existierenden Entities bestehen. Im relationalen Modell müssen "dangling Zeiger" durch Fremdschlüsselbedingungen ausgeschlossen werden.

Generelle Bemerkungen (3)

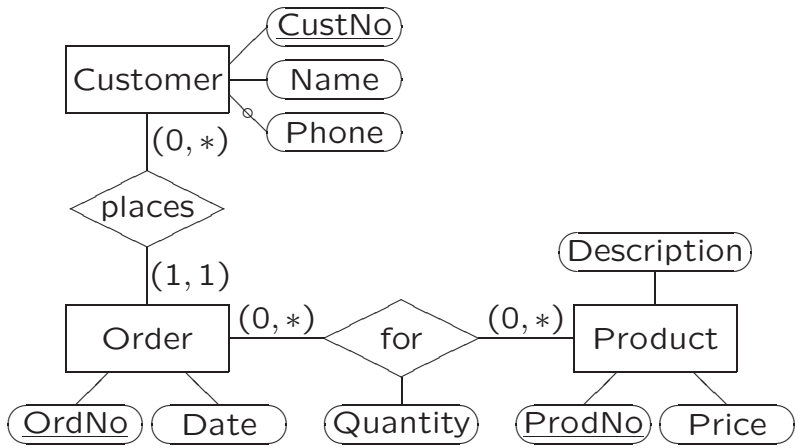
- Zusätzlich muss es möglich sein, Anfragen bezogen auf S_E in Anfragen bezogen auf S_R zu übersetzen, diese im relationalen System auszuwerten, und die Antwort zurückzuübersetzen.
- D.h. es muss möglich sein, die entworfene ER-Datenbank durch die eigentlich implementierte relationale Datenbank zu simulieren.

  Jede Schema-Übersetzung sollte den Zusammenhang von Schemaelementen erklären, sodass, in unserem Fall, jede Anfrage an das ER-Schema auch an das relationale Schema gestellt werden kann.

Inhalt

1. Ziele des logischen Design
2. Basis ER-Konstrukte
3. Schwache Entities
4. Eins-zu-Eins-Beziehungen
5. Letzte Schritte, Einschränkungen

Beispiel



Schritt 1: Entities (1)

- Als erstes erstellt man für jedes Entity eine Tabelle. Der Name der Tabelle ist der Name des Entities.

  Alternativ kann die Pluralform verwendet werden.

- Die Spalten der Tabelle sind die Attribute des Entity-Typs.

  Optionale Attribute werden in Spalten übersetzt, die Nullwerte erlauben.

- Der Primärschlüssel der Tabelle ist auch der Primärschlüssel des Entity-Typs.

  Falls der Entity-Typ keinen Primärschlüssel hat, wird ein künstlicher Schlüssel hinzugefügt.

Schritt 1: Entities (2)

Customers		
CustNo	Name	Phone
10	Jones	624-9404
11	Smith	

Orders	
OrdNo	Date
200	2/15/00
201	2/16/00

Products		
ProdNo	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

Schritt 2: Eins-Zu-Viele-Bez. (1)

- Hat eine Beziehung die maximale Kardinalität 1 auf einer Seite, so ist es eine 1-zu-viele Beziehung. Z.B. ist "places" 1-zu-viele von "Customer" zu "Order".

  Hat es die maximale Kardinalität 1 auf beiden Seiten, so ist es eigentlich eine eins-zu-eins-Beziehung (siehe unten).

- In diesem Fall wird der Schlüssel der "eins"-Seite (Customer) als Spalte zur "viele"-Seite (Order) zugefügt.

- Diese Spalte wird ein Fremdschlüssel, die die Zeile des entsprechenden Entities referenziert.

Schritt 2: Eins-Zu-Viele-Bez. (2)

- Ergebnis für das Beispiel:

  `Orders(OrdNo, Date, CustNo→Customers)`

Orders		
OrdNo	Date	CustNo
200	2/15/00	11
201	2/16/00	11

Customers		
CustNo	Name	Phone
10	Jones	624-9404
11	Smith	

Schritt 2: Eins-Zu-Viele-Bez. (3)

- Wenn die Minimum-Kardinalität 1 ist (wie in diesem Fall), so sind Nullwerte für die Fremdschlüsselspalte nicht erlaubt.
- Sollte die Minimum-Kardinalität 0 sein, so müssen Nullwerte in der Fremdschlüsselspalte erlaubt sein.

  Der Wert in dieser Spalte ist Null für die Entities, die nicht an der Beziehung teilnehmen.

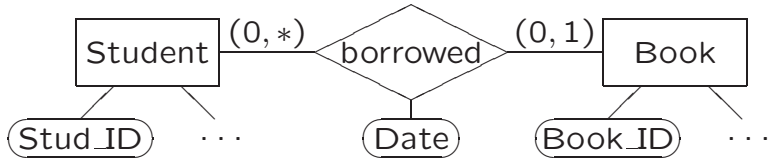
Schritt 2: Eins-Zu-Viele-Bez. (4)

- Der Beziehungsname kann in dem Spaltennamen verwendet werden, z.B.

  `Orders(OrdNo, Date, placed_by→Customers)`
- Das schließt natürliche Joins aus (verbindet Tabellen durch gleichnamige Spalten, vgl. Kap. 6), diese Operation ist in SQL aber nicht wichtig (Stilfrage).
- Natürlich müssen alle Spalten einer Tabelle eindeutige Namen haben. Hat ein zugefügter Fremdschlüssel den gleichen Namen wie eine vorhandene Spalte, muss mindestens eine umbenannt werden.

Schritt 2: Beziehungsattribute

- Beziehungen können Attribute haben, z.B.:



- Solche Attribute werden zusammen mit dem Zeiger auf das bezogene Entity gespeichert.

  `Books(Book_ID, ..., Stud_ID`^{O}`→Students, Date`^{O}`)`

  "Stud_ID" und "Date" können Null sein, da nicht jedes Buch ausgeliehen wird, aber sie können nur zusammen Null oder nicht Null sein (→ `CHECK`-Constraint).

Schritt 2: Eine Variante

- Eins-zu-viele-Beziehungen mit Kardinalität $(0, 1)$ können in eine eigene Tabelle übersetzt werden.

  `borrowed_by(Book_ID→Books, Stud_ID→Students, Date)`
- Die Schlüssel beider zugehöriger Entities und die Beziehungsattribute werden in der Tabelle gespeichert. Das Schlüsselattribut der Seite mit der $(0, 1)$-Kardinalität wird Schlüssel der Relation.

  Jedes Buch kann zu einer Zeit nur ein Mal ausgeliehen werden.
- Dies funktioniert mit der (1,1)-Kardinalität nicht.

Schritt 3: Viele-Zu-Viele-Bez. (1)

- Eine Beziehung ist viele-zu-viele, wenn sie die maximale Kardinalität $*$ auf beiden Seiten hat (z.B. "for").
- Viele-zu-viele-Beziehungen werden in eigene Tabellen übersetzt.
- Die Spalten der Tabelle sind die Schlüssel der teilnehmenden Entity-Typen, die zusammen den Schlüssel dieser Tabelle bilden.
- Diese Spalten sind zugleich auch Fremdschlüssel, die die Tabellen der Entity-Typen referenzieren.

Schritt 3: Viele-Zu-Viele-Bez. (2)

- Beziehungsattribute werden als Spalten hinzugefügt. Sie sind nicht Teil des Schlüssels, z.B.

  `for(`<u>OrdNo</u>`→Orders, `<u>ProdNo</u>`→Products, Quantity)`
- Man beachte, dass der Schlüssel von "for" wirklich aus "OrdNo" und "ProdNo" bestehen muss.

  Da eine Bestellung mehrere Produkte beinhalten kann, kann "OrdNo" allein nicht Schlüssel sein.
  Da das gleiche Produkt in verschiedenen Bestellungen auftreten kann, reicht auch "ProdNo" allein als Schlüssel nicht aus.
- Tabellen können umbenannt werden. Z.B. ist "`Order_Details`" ein besserer Name als "`for`".

Schritt 3: Viele-Zu-Viele-Bez. (3)

for		
<u>OrdNo</u>	<u>ProdNo</u>	Quantity
200	1	1
200	2	1
201	1	5

Orders		
<u>OrdNo</u>	Date	CustNo
200	2/15/00	11
201	2/16/00	11

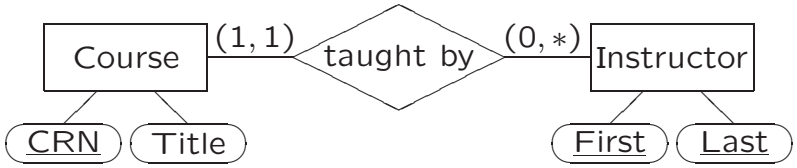
Products		
<u>ProdNo</u>	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

Schritt 3: Viele-Zu-Viele-Bez. (4)

- Eine andere Minimumkardinalität als 0 bei einer viele-zu-viele-Bez. kann nicht durch Standard-Constraints des relationalen Modells spezifiziert werden.
- Z.B. macht es Sinn zu verlangen, dass jede Bestellung mindestens ein Produkt enthält. Dies wird aber ein allgemeiner Constraint im relationalen Modell.
- Da dies für die Gültigkeit des DB-Zustandes wichtig ist, kann man die Kardinalitäten spezifizieren und später durch Anwendungsprogramme überprüfen.

  Es kann nur in der `CREATE TABLE` -Anweisung nicht festgelegt werden.

Zusammengesetzte Fremdschl.



- Ein zusammengesetzter Fremdschlüssel wird verwendet, um eine Tabelle mit einem zusammengesetzten Schlüssel zu referenzieren.

  Course(CRN, Title, (First, Last) → Instructor)

- Wäre die Minimum-Kardinalität 0, könnten "First" und "Last" Null sein, aber nur zusammen.

Inhalt

1. Ziele des logischen Design

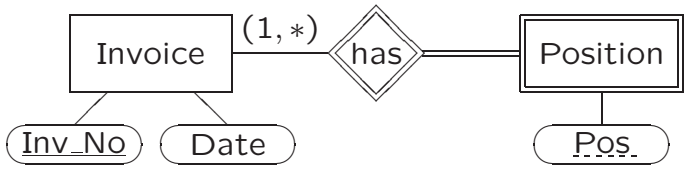
2. Basis ER-Konstrukte

3. Schwache Entities

4. Eins-zu-Eins-Beziehungen

5. Letzte Schritte, Einschränkungen

Schritt 1B:Schwache Entities (1)



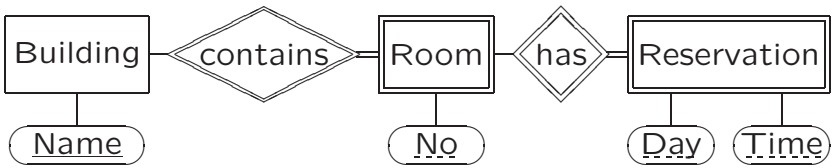
- Wird ein schwaches Entity übersetzt, müssen die Schlüssel des Master-Entity als Fremdschlüssel zugefügt werden.

  Position(Inv_No → Invoice, Pos, . . .)

- Das implementiert automatisch die Beziehung.

  Eine solche Beziehung muss in Schritt 2 ignoriert werden. Es ist sinnvoll, hier "`DELETE CASCADES`" für den Fremdschlüssel zu spezifizieren. Man beachte, dass es im relationalen Modell keine "gestrichelte Unterstreichung" gibt.

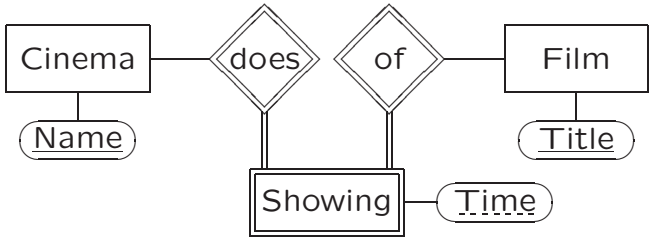
Schritt 1B:Schwache Entities (2)



- Ist ein schwaches Entity selbst Master eines anderen schwachen Entities, so wird das vererbte Schlüsselattribut weitergereicht:
  Buildings(Name)
  Rooms(Name → Buildings, No)
  Reservations((Name, No) → Rooms, Day, Time)

- Übung: Muss Name in Reservations als Fremdschlüssel deklariert werden, der Buildings referenziert?

Schritt 1B:Schwache Entities (3)



- Assoziationsentities vererben Schlüsselattribute von mehr als einer Quelle:

  Cinemas(Name)
  Films(Title)
  Showings(Name → Cinemas, Title → Films, Time)

Inhalt

1. Ziele des logischen Design

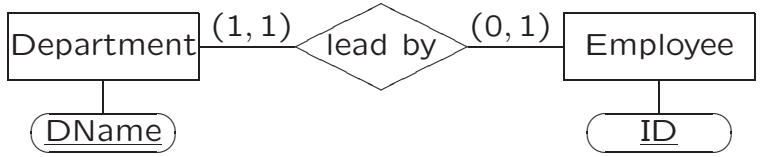
2. Basis ER-Konstrukte

3. Schwache Entities

4. Eins-zu-Eins-Beziehungen

5. Letzte Schritte, Einschränkungen

Schritt 4: Eins-zu-Eins-Bez. (1)



- Eine Beziehung ist eins-zu-eins, wenn sie die maximale Kardinalität 1 auf beiden Seiten hat.
- Die Übersetzung ist eigentlich die gleiche wie für eins-zu-viele-Beziehungen.

  Aber es wird ein zusätzlicher Schlüssel konstruiert, siehe unten.

Schritt 4: Eins-zu-Eins-Bez. (2)

- In diesem Beispiel ist es besser, den Schlüssel von Employee in die Tabelle Department zu übernehmen, als umgekehrt, da das Department die Kardinalität (1,1) hat:

  Department(DName, . . . , Head → Employee)

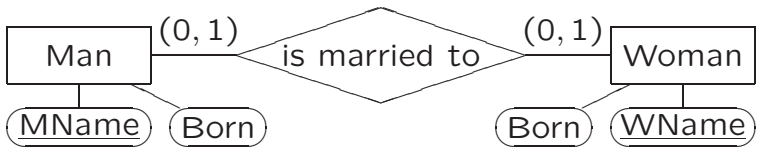
- So werden Nullwerte vermieden, und die Minimumkardinalität 1 gewährleistet.

  Würde man den Namen des Departments in die Employee-Tabelle aufnehmen, so könnte er Null sein. Zusätzlich wäre ein allgemeiner Constraint erforderlich, um zu sichern, dass jedes Department einen Leiter (Head) hat.

Schritt 4: Eins-zu-Eins-Bez. (3)

- "Head" ist nun auch Schlüssel für die Tabelle "Department" (!), da ein Angestellter (Employee) maximal Leiter (Head) eines Departments sein kann.
- "Head" ist nur ein Alternativschlüssel, nicht Teil des Primärschlüssels.
- Das sichert die maximale Kardinalität 1 auf der Employee-Seite.

Schritt 4: Eins-zu-Eins-Bez. (4)

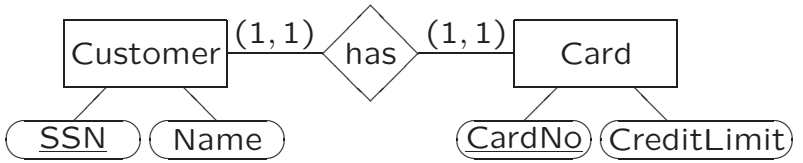


- Der Schlüssel einer der beiden Tabellen wird als (optionaler) Fremdschlüssel in die andere Tabelle übernommen.

  Es wäre aber falsch, beides zu tun (Redundanz).
- Oder man bildet eine eigene Tabelle (für die Bez.).

  Marriage(MName → Man, WName → Woman)
- Übung: Was ist der/die Schlüssel?

Schritt 4: Eins-zu-Eins-Bez. (5)



- Um die minimale Kardinalität 1 auf beiden Seiten zu gewährleisten, müssen die Tabellen zu einer Tabelle zusammengefasst werden.

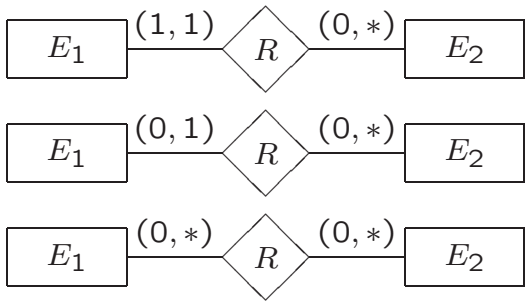
  CustomerCard(SSN, Name, CardNo, CreditLimit)
- SSN und CardNo sind beides Schlüssel. Einer wird als Primärschlüssel ausgewählt, der andere ist Alternativschlüssel.

Inhalt

1. Ziele des logischen Design
2. Basis ER-Konstrukte
3. Schwache Entities
4. Eins-zu-Eins-Beziehungen
5. Letzte Schritte, Einschränkungen

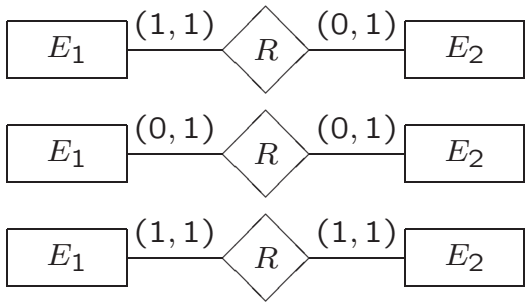
Einschränkungen (1)

- Folgende Kardinalitäten können mit den oben genannten Methoden übersetzt werden (wobei nur die Standard-Constraints des relationalen Modells verwendet werden):



Einschränkungen (2)

- Zusätzlich können alle Arten von eins-zu-eins-Beziehungen behandelt werden.

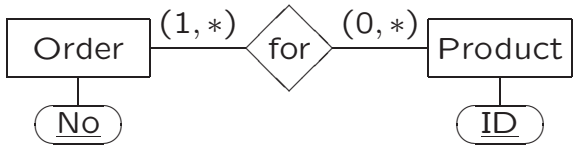


Einschränkungen (3)

- Trifft auf eine Beziehung keiner dieser sechs Fälle zu, müssen allgemeine Constraints verwendet werden, die implementiert werden, z.B. via
  - ◇ Überprüfungen in Anwendungsprogrammen, die zur Einfügung von Daten verwendet werden.
  - ◇ Trigger, d.h. in der DB gespeicherte Prozeduren, die automatisch, z.B. für jedes eingefügte oder modifizierte Tupel, ausgeführt werden.
  - ◇ SQL-Anfragen, die von Zeit zu Zeit ausgeführt werden, um Verletzungen von Constraints herauszufiltern.

Einschränkungen (4)

- Speziell ist die Kardianlität $(1,*)$ manchmal sinnvoll, z.B. sollte jede Bestellung mindestens ein Produkt umfassen:



- Die Minimumkardinalität 1 kann in den aktuellen DBMS nicht deklarativ gesichert werden.

  Man verwendet stattdessen die gleiche Übersetzung wie für $(0,*)$ und spezifiziert zusätzlich einen allgemeinen Constraint.

Schritt 5: Überprüfung (1)

- Zum Schluss werden die erstellten Tabellen überprüft, um festzustellen, ob sie Sinn machen.
- Z.B. kann man die Tabelle mit einigen Beispielzeilen füllen.
- Ist ein korrektes ER-Schema korrekt in das relationale Modell übersetzt, so erhält man ein korrektes relationales Schema.
- Trotzdem kann die von-Hand-Übersetzung Fehler mit sich bringen, und das ER-Schema kann versteckte Fehler enthalten.

Schritt 5: Überprüfung (2)

- Manchmal sind Tabellen redundant und können gelöscht werden.
- Man sollte ein letztes Mal über die Umbenennung von Tabellen und Attributen nachdenken.
- Wenn zwei Tabellen den gleichen Schlüssel haben, sollte man überlegen sie zu verschmelzen (das bedeutet aber nicht, dass man es immer tun muss!).
- Außerdem überprüft man die erstellten Tabellen auf relationale Normalform (z.B. 3NF, BCNF, 4NF) (vgl. Kapitel 11).