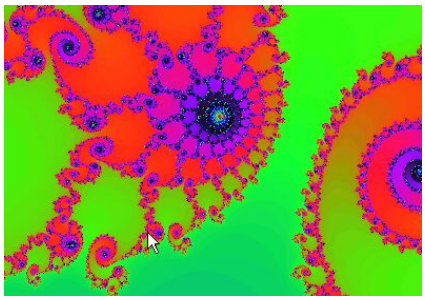
Datenbanken

7. ERM ➔ RM Transformation



Fachhochschule Darmstadt
Fachbereich Informatik

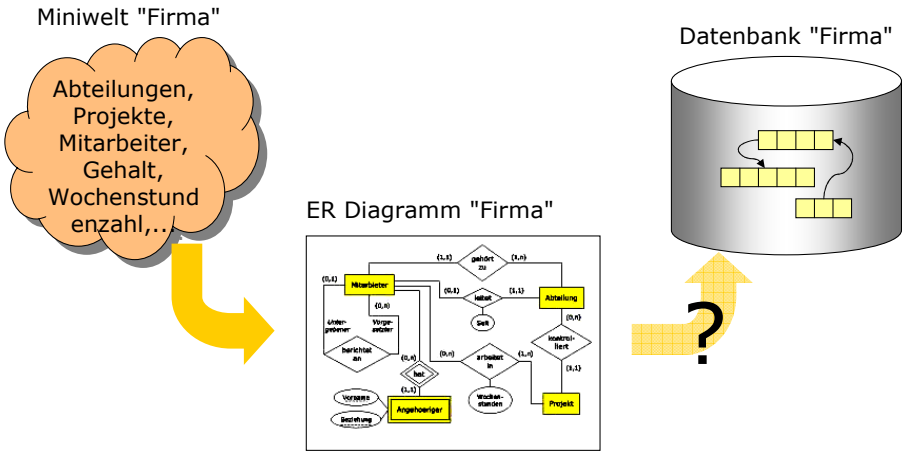
Dr. Peter Nevermann

Rückblick

- **SQL DDL**
  - Datenbanken erzeugen und löschen
  - Tabellendefinitionen innerhalb einer Datenbank erzeugen, verändern und löschen
  - Attributdefinitionen innerhalb einer Tabelle erzeugen, verändern und löschen
  - Integritätsregeln innerhalb einer Tabelle/Datenbank erzeugen und löschen

ERM ➔ RM Transformation

- **Finde ein geeignetes Datenbankschema für die Miniwelt "Firma"**



Nachdem die Miniwelt "Firma" mit Mitteln der ER Modellierung beschrieben wurde, kommt nun der nächste Schritt, nämlich die Überführung in ein konkretes Datenbankschema.

Für die Definition eines relationalen Datenbankschemas haben wir bereits SQL DDL kennengelernt ... bleibt also noch zu klären, wie die Elemente des ER Modells in SQL DDL Definitionen transformiert werden können.

CASE Tools

Es gibt eine Reihe von CASE Tools (CASE = *Computer Aided Software Engineering* - also Werkzeuge, die die Software Entwicklung unterstützen und automatisieren) mit denen ER Entwürfe erstellt und automatisch in SQL DDL Definitionen überführt werden können.

Beispiele (Reihenfolge zufällig):

- PowerDesigner (Sybase)
- Innovator (MID)
- ERWin (Computer Associates)
- Rational Rose (IBM)

Ich selbst habe vor einigen Jahren die CASE Tools "Predict Case" und "Natural Engineering Workbench" der Software AG mitentwickelt. In beiden Werkzeugen gibt es eine Komponente namens "Schemagenerator" zur automatischen ER -> RM Transformation ☺.

Entities

- **Das Entity "Abteilung" wird in die Relation "abteilung" überführt**


```
CREATE TABLE abteilung (
  abtname              VARCHAR(50)   NOT NULL,
  anr                  INT           NOT NULL,
  mgrstartdat          DATE,
  PRIMARY KEY (anr),
  UNIQUE (abtname));
```

Datenbanken I | 4 | SS 05, 27.05.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Schritt 1: Entity ➔ Tabelle (*Entity-Tabelle*)

•Jedes Entity (ausgenommen Weak-Entities) wird in eine Tabelle überführt.

•Jedes einfache Attribut (d.h. einwertig, atomar) des Entities wird zu einer Spalte in der Tabelle.

•Jede einfache Komponente eines einwertigen, gegliederten Attributs wird ebenfalls zu eine Spalte (z.B. wird das Attribute Name (Nachname, Vorname) des Entities "Mitarbeiter" in die Spalten "nachname" und "vorname" der Tabelle "*mitarbeiter*" überführt.

•Der Primärschlüssel des Entities wird ebenfalls in der Tabelle als PRIMARY KEY deklariert.

Schritt 2: Weak-Entity ➔ Tabelle (*Weak-Entity-Tabelle*)

•Jedes Weak-Entity wird in eine eigene Tabelle überführt.

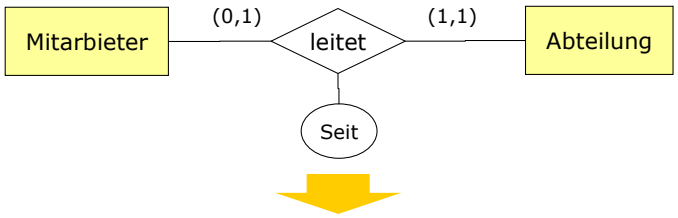
•Mit den Attributen (einfache Komponenten gegliederter Attribute) wird wie unter Schritt 1 beschrieben verfahren.

•Für den Primärschlüssel des Owner-Entities wird eine neue Spalte definiert und als FOREIGN KEY mit Referenz auf den Primärschlüssel der Owner-Entity-Tabelle deklariert. Dabei sollte ON UPDATE CASCADE und ON DELETE CASCADE eingestellt werden, weil ein Weak-Entity-Tupel keine Exitenzberechtigung mehr hat, wenn das Owner-Entity-Tupel gelöscht wird.

•Der PRIMARY KEY der Weak-Entity-Tabelle setzt sich aus dem FOREIGN KEY (Referenz auf den PK der Owner-Entity-Tabelle) und dem partiellen Schlüssel des Weak-Entities zusammen.

Beziehungen

- **Die Beziehung "leitet" (Mitarbeiter → Abteilung) wird als Fremdschlüssel "mgrpnr" in der Tabelle "abteilung" realisiert. Das Attribut "seit" wird in "mgrstartdat" überführt.**




```
ALTER TABLE abteilung
  ADD mgrstartdat DATE,
  ADD FOREIGN KEY (mgrpnr) REFERENCES mitarbeiter(pnr)
    ON UPDATE CASCADE ON DELETE SET NULL);
```

Datenbanken I | 5 | SS 05, 27.05.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Schritt 3: Beziehungen mit (1,1)- oder (0,1)-Seite ➔ Fremdschlüssel

•Für jede Beziehung mit einer (1,1)- oder (0,1)-Seite wird in der Tabelle - die zu dieser Seite gehört - eine neue Spalte definiert und als FOREIGN KEY mit Referenz auf den Primärschlüssel der gegenüberliegenden Tabelle deklariert. Dabei wird man i.d.R. ON UPDATE CASCADE und ON DELETE SET NULL einstellen.

•Jedes einfache Attribut (bzw. einfache Komponente eines gegliederten Attributs) der Beziehung wird als Spalte in die selbe Tabelle übernommen, in der schon der Fremdschlüssel angelegt wurde.

•Hat eine Beziehung sowohl eine (0,1)- als auch eine (1,1)-Seite, so ist i.d.R. die (1,1)-Seite vorzuziehen, da man auf der (0,1)-Seite mit potentiell vielen NULL Werten im Fremdschlüssel rechnen muß.

•Hat eine Beziehung zwei (1,1)-Seiten spielen evtl. andere Kriterien eine Rolle, z.B. die Richtung in der die Beziehung durch die Anwendungen "navigiert" werden soll. Selten sinnvoll aber denkbar wäre auch noch beide Entities in eine einzige Tabelle zu überführen.

Schritt 4: Beziehungen ohne (1,1)- oder (0,1)-Seite ➔ Tabelle (*Beziehungstabelle*)

•Für jede Beziehung ohne (1,1)- oder (0,1)-Seite muß eine eigene Tabelle erzeugt werden, da solche Beziehungen sich nicht mittels Fremdschlüssel realisieren lassen.

•Die Primärschlüssel der beteiligten Tabellen werden jeweils als Fremdschlüssel in der Beziehungstabelle eingerichtet. Dabei wird für beide Fremdschlüssel ON UPDATE CASCADE und ON DELETE CASCADE eingestellt, da ein Beziehungstupel nur existieren kann, wenn die beteiligten Entity-Tupel existieren.

•Die Kombination der beiden Fremdschlüssel wird als Primärschlüssel der Beziehungstabelle deklariert.

•Jedes einfache Attribut (bzw. einfache Komponente eines gegliederten Attributs) der Beziehung wird als Spalte in die Beziehungstabelle übernommen.

Beispiel:

Für die Beziehung [Mitarbeiter –(0,n)-- <arbeitet in> --(1,n)-- Projekt] mit dem Attribut 'Wochenstunden' wird folgende Beziehungstabelle definiert:

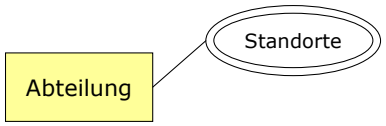
```
CREATE TABLE arbeitet_in (
  mapnr          int        NOT NULL,
  projectnr      int        NOT NULL,
  stunden        int        NOT NULL,
  PRIMARY KEY (mapnr, projectnr),
  FOREIGN KEY (mapnr)     REFERENCES mitarbeiter (pnr)  ON UPDATE CASCADE ON DELETE CASCADE,
  FOREIGN KEY (projectnr) REFERENCES projekt (pnummer)  ON UPDATE CASCADE ON DELETE CASCADE,
);
```

Bemerkung:

Beziehungen ohne (1,1)-Seite aber mit (0,1)-Seite können statt mit einem Fremdschlüssel auch in eine eigene Beziehungstabelle überführt werden, wie in Schritt 4 beschrieben. Das macht besonders dann Sinn, wenn relativ wenig Beziehungs-Instanzen existieren und man viele NULL-Werte im Fremdschlüssel vermeiden will.

Mehrwertige Attribute

- **Das mehrwertige Attribut "Standorte" des Entities "Abteilung" wird in eine eigene Tabelle "standort" überführt [Alternative A].**




```
CREATE TABLE standort (
  abtnummer           int          NOT NULL,
  stadt               varchar(50)  NOT NULL,
  PRIMARY KEY (abtnummer, stadt),
  FOREIGN KEY (abtnummer) REFERENCES abteilung (anr)
    ON UPDATE CASCADE ON DELETE CASCADE);
```


Schritt 5: Mehrwertige Attribute ➔ Tabelle (*Wertetabelle*)

•Für jedes mehrwertige Attribut muß eine eigene Tabelle erzeugt werden, da solche Attribute sich nicht in der Entity-Tabelle realisieren lassen.

•Die Primärschlüssel der zugehörigen Entity-Tabelle wird als Fremdschlüssel in der Wertetabelle eingerichtet. Dabei wird ON UPDATE CASCADE und ON DELETE CASCADE eingestellt, da ein Wert-Tupel nur existieren kann, wenn das beteiligte Entity-Tupel existiert.

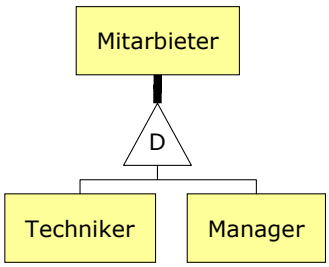
•Das Attribut selbst wird als Spalte in der Wertetabelle realisiert (*Attributspalte*). Falls das Attribut gegliedert ist, wird jede einfache Komponente als Spalte in der Wertetabelle realisiert. Gibt es eine Komponente, die selbst wieder multipel ist, muß dieses in eine weitere Wertetabelle gestuft ausgelagert werden.

•Die Kombination des Fremdschlüssels mit der Attributspalte wird als Primärschlüssel der Wertetabelle deklariert.

In dem Beispiel der Folie ist "stadt" die Attibutspalte.

Spezialisierung/Generalisierung

- **Für die Spezialisierung des Entities "Mitarbeiter" werden Super- und Subtypen jeweils in eine eigene Tabelle überführt.**




```
CREATE TABLE mitarbeiter (
CREATE TABLE techniker (
CREATE TABLE manager (
  ..
  budget DECIMAL,
  ..
  PRIMARY KEY (pnr),
  FOREIGN KEY (pnr)
    REFERENCES mitarbeiter(pnr)
    ON UPDATE CASCADE
    ON DELETE CASCADE);
```

Datenbanken I | 8 | SS 05, 27.05.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Schritt 6: Spezialisierungen

Für die Realisierung von Spezialisierungen im Relationenmodell gibt es 3 Alternativen.

Alternative A: ➔ Super- und Subtypen werden je in eine eigene Tabelle überführt

•Der Supertyp wird wie unter Schritt 1 beschrieben in eine Tabelle (*Supertyp-Tabelle*) überführt.

•Für jeden Subtypen wird eine eigene Tabelle (*Subtype-Tabelle*) erzeugt, welche die subtype-spezifischen Attribute als Spalten enthält.

•Der Primärschlüssel der Supertyp-Tabelle wird als Primärschlüssel in jede Subtyp-Tabelle hinzugefügt. Dieser wird in jeder Subtyp-Tabelle zusätzlich als Fremdschlüssel mit Referenz auf den Primärschlüssel der Supertyp-Tabelle deklarierte. Dabei wird ON UPDATE CASCADE und ON DELETE CASCADE eingestellt, da ein Subtyp-Tupel nur existieren kann, wenn das zugehörige Supertyp-Tupel existiert.

Das Beispiel auf der Folie zeigt Alternative "A" für die Mitarbeiter-Spezialisierung.

Alternative B: ➔ Nur die Subtypen werden je in eine eigene Tabelle überführt

•Für jeden Subtypen wird eine eigene Tabelle (*Subtype-Tabelle*) erzeugt, welche alle Supertyp-Attribute plus die subtype-spezifischen Attribute als Spalten enthält.

•Der Primärschlüssel des Supertyp-Entities wird als Primärschlüssel in jede Subtyp-Tabelle hinzugefügt.

Alternative C: ➔ Super- und Subtypen werden in eine einzige, gemeinsame Tabelle (*Spezialisierungstabelle*) überführt

•Es wird eine einzige Tabelle für alle an der Spezialisierung beteilgten Entities erzeugt

•Alle Supertyp-Attribute und alle subtyp-spezifischen Attribute werden zu Spalten der Spezialisierungstabelle

•Der Primärschlüssel des Supertyps wird als PRIMARY KEY in der Spezialisierungstabelle deklariert

•Falls die Spezialisierung überlappungsfrei (*disjoint*) ist, wird noch eine Spalte "typ" hinzugefügt (*Typ-Attribut*), mit dessen Hilfe der Subtyp des Tupels festgelegt werden kann. In unserem Beispiel könnte die Spalte vom Aufzählungstyp enum('Techniker', 'Manager') sein.

•Ist die Spezialisierung überlappend (*overlapping*), kann statt einer einzigen Typ-Spalte auch pro Subtyp eine boolesche Typ-Spalte hinzugefügt werden, z.B. "typ-techniker" und "type-manager" jeweils mit den Werten {TRUE, FALSE}.

Die Alternativen im Vergleich

Alternative A eignet sich für jede Art von Spezialisierung (*disjoint/oberlapping, total/partiell*) und hat den Vorteil, daß sie arm an Redundanzen und NULL-Werten ist. Nachteil ist, daß teure Joins erforderlich sind, um Informationen zu gewinnen.

Alternative B geht nur für *disjoint + total* Spezialisierungen. Nachteil gegebüber A ist, daß die Attribute des Supertyps redundant gespeichert werden. Vorteil gegenüber A ist, daß die Joins der Super- mit den Subtypen sozusagen in den Subtyp-Tabellen materialisiert sind und somit Alternative B effizienter als A sein kann.

Alternative C eignet sich, wie A, für alle Spezialisierungsarten. Allerdings ist C nur dann zu empfehlen, wenn die Anzahl der subtyp-spezifischen Attribute gering ist, da sonst mit einer großen Anzahl von NULL-Werten zu rechnen ist. Alternative C hat außerdem den Vorteil des materialisierten Joins.

Spezialisierungshierarchien

Bei Spezialisierungshierarchien können unterschiedliche Strategien für jede einzelne Ebene der Hierarchie verwendet werden, d.h. Alternativen A, B und C können vermischt werden.

Aufgabe:

•Was tun bei Mehrfach-Spezialisierungen des selben Entities?

•Was tun bei Mehrfach-Vererbung?

Ausblick

- **Relationale Algebra**
  - Eine mathematische Formalisierung des Relationenmodells