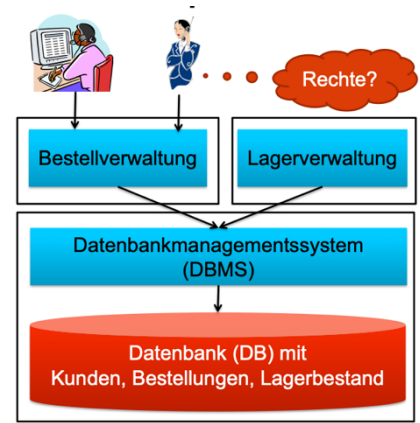


Kapitel 1: Einleitung

Daten dauerhaft speichern

- | | |
|--------------------------|---|
| 1. Operationen | - (lesen, schreiben, verändern, abfragen...) |
| 2. Konsistenzüberwachung | - (Prüft Bed. z.B. Kunde existiert) |
| 3. Synchronisation | - (mehrere Benutzer haben Zugriff) |
| 4. Datensicherung | - (ohne Datensalat) |
| 5. Integration | - (mehrere unt. Prg. haben Zugriff auf die DB) |
| 6. Benutzersichten | - (nur Teilansicht auf die DB) |
| 7. Transaktionen | - (Komplexe Prg Stücke) |
| 8. Katalog | - (Selbstbeschreibung der DB) |
| 9. Zugriffskontrolle | - (Benutzergruppen mit eingeschränkten Rechten) |



Begriffsklärungen

Datenbank (DB) / Datenbasis

- Strukturierter Datenbestand, der von einem Datenbankmanagementsystem verwaltet wird.
 - *Datenbankschema*
 - Legt die Struktur der Datenobjekte in der Datenbank fest (Metadaten)
 - *Datenbankausprägung (Instanz)*
 - Konkreter Inhalt / Zustand der Daten in einer Datenbank

Datenbankmanagementsystem (DBMS)

- Software zur Verwaltung von Datenbanken (Daten definieren, Daten speichern/ändern/löschen, Anfragen implementieren, Sicherheit)

Datenbanksystem (DBS)

- DBMS und Datenbank(en)

ANSI / SPARC - Modell

Externe Ebene: (Users sicht)

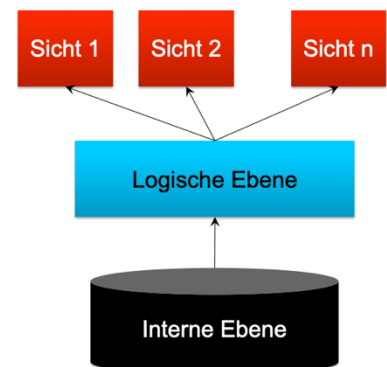
Teilmenge an Informationen, die für eine Anwendung erforderlich sind.

Logische Ebene:

In einem Schema wird festgelegt, welche Daten gespeichert werden.

Interne Ebene: (Speicher)

Legt fest, wie Daten auf den Speichermedien organisiert, codiert und abgelegt werden. (Auch: physische Ebene)



VL02_Konzeptionelles_Modell

Phasen des Datenbankentwurfs

1. Das **Fachproblem** liegt normalerweise vor.
2. **Anforderungsanalyse**: Welche Informationen werden in der Datenbank gespeichert, welche Operationen werden auf den Daten ausgeführt werden, usw.
3. **Konzeptioneller Entwurf**: Beschreibe das Schema der Daten unabhängig von der späteren Implementierung
4. **Logischer Entwurf**: Übersetzen des konzeptionellen Schemas in ein Implementierungsmodell, z.B. das relationale Modell. Verbesserung des Modells durch z.B. Normalisierung.
5. **Physischer Entwurf**: Schema-Entwicklung für ein spezielles DBMS, Deklaration der Daten, Festlegung der (Speicher-)Zugriffstrukturen
6. **Implementierung und Wartung**: Installation des Datenbanksystems, Anpassung an neue Anforderungen.

Entity Relationship Modell (ER Modell)

- Basiert auf den Grundkonzepten **Entity** (Informationseinheit), **Attribut** (Eigenschaft eines Entitys) und **Relation** (Beziehung zwischen Entities)

Miniwelt

- relevanter Ausschnitt der Realität besteht aus "Objekten" die bestimmten Eigenschaften haben.

Konzeptionelle Modelle

- Strukturieren die Miniwelt und beschreiben die relevanten Daten unabhängig von Implementierung oder einzelnen Anwendungen
- Integritätsbedingungen verbessern Übereinstimmung zwischen Realität und Modell

Entity: „etwas“ aus der realen Welt, physisch oder konzeptionell existierendes Objekt

Entity-Typ:

- definiert eine Menge von gleichartigen Entities mit gleichen Attributen, also gemeinsamen Eigenschaften (diese Menge wird auch als **Entitymenge** bezeichnet)

Attribute

- Eigenschaft, die alle Elemente desselben Entitytyps besitzen (gemeinsame Eigenschaften).
- Die zulässigen Werte eines Attributes nennt man Wertebereich oder Domäne (engl. Domain).

Beziehung

- Ein Beziehungstyp deklariert eine Beziehung zwischen Entity-Typen.
- **(Stelligkeit)** Es kann eine beliebige Anzahl von Entity-Typen an einem Beziehungstyp teilhaben
- **(Kardinalität)** 1:1, 1:n, n:m Notation modelliert die Anzahl der beteiligten Entities an einer Beziehung
- **(min, max)** Notation modelliert, wie oft ein Entity in der Beziehung vorkommen kann.
- **Rekursive Beziehungstypen**: Ein Entity-Typ kann auch mehrfach an dem gleichen Beziehungstyp teilnehmen.
- **Beziehungstypen** sind abstrahierte Beziehungen zwischen Entity-Typen

Schlüssel

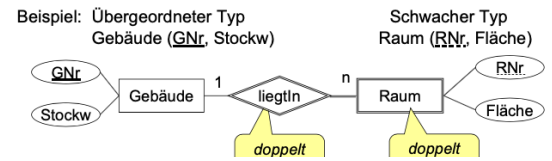
- **Schlüssel** sind **Teilmengen** von Attributen, die ein Objekt **eindeutig identifizieren**
- **Schlüssel müssen eindeutig und unveränderlich sein!**

Erweiterte Konzepte

- Mengenwertige und zusammengesetzte Attribute

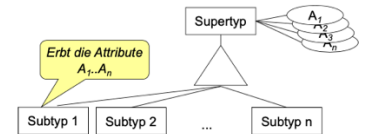
Schwache Entity-Typen

- sind von der **Existenz eines übergeordneten Entities abhängig**
- nur in Kombination mit dem **Schlüssel** des übergeordneten Entities eindeutig identifizierbar.
- **Doppelt umrandet und Schlüssel gestrichelt!**



Generalisierung/(Spezialisierung)

- Gemeinsame Attribute werden "**herausfaktoriert**" und dem Supertypen (Obertypen) **zugeordnet**
- **Subtypen** (Untertypen) erben die **Attribute** ihrer Supertypen (vom Speziellen zum Allgemeinen)



Klassenhierarchien Vorgehensweise beim Modellentwurf

Top-down – Strategie

- beginnen mit einem Schema, das hohe Abstraktionen enthält
- Sukzessive Verfeinerungen

Benennungskonflikte

- **Synonyme** - Mehrere Bezeichnungen für das gleiche Konzept: Chef / Vorgesetzter
- **Homonyme** - Gleiche Bezeichnung für unterschiedliche Konzepte Blatt (Papier, am Baum), -> Projektglossar anlegen!

Typkonflikte

- das gleiche Konzept in zwei Schemas unterschiedlich modelliert (Abteilung in einem Schema als ET und in einem anderen Schema als Attribut)

Konflikte zwischen Wertemengen

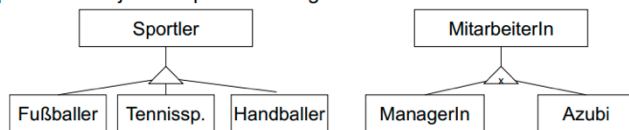
- verschieden Wertemengen eines Attributs

Konflikte zwischen Einschränkungen

- verschiedene Schlüssel, verschiedene Kardinalitäten

Vererbungshierarchien

Überlappende vs. disjunkte Spezialisierungen

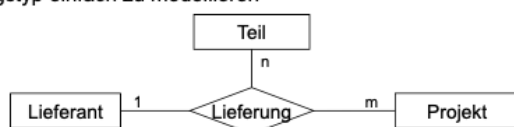


Totale vs. partielle Spezialisierungen

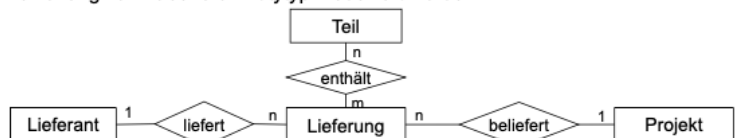


Mehrstellige Beziehungen

Als Beziehungstyp einfach zu modellieren



Beziehung kann aber als Entitytyp modelliert werden



Kapitel 3 Implementierungsmodell

Relationenschema Personen(Name, Straße, TelNummer)

$$\text{dom}(\text{Name}) = \text{dom}(\text{Strasse}) = \{\text{Zeichenkette}\}$$
$$\text{dom}(\text{TelNummer}) = \{8\text{-stellige Zahl}\}$$

Relationenmodell:

- Relationen mit Attributen (Tabellen)
- jedes Attribut hat einen Datentyp (atomare Werte - Zeichenketten (Strings), Zahlen)

Relationale Algebra:

- Anfragen lassen sich in Relationalenalgebra ausdrücken
 - Relationale Operatoren arbeiten auf Relationen (Tabellen)
- Operationen, um Daten aus den Tabellen abzufragen.
- Zentrale Operationen:
 - Selektion** (Einträge mit bestimmten Eigenschaften), **Projektion** (bestimmte Spalten einer Relation)

$\sigma_P(R)$
 Bsp. $\sigma_{\text{Gehalt} > 50000}(\pi_{\text{Name}, \text{Gehalt}, \text{Alter}}(\text{Person}))$

$-\pi_{\text{Name}}(\text{Kunde})$
 - Kreuzprodukt -> **Verbund** (Vergleichsoperator $\theta \in \{<, =, >, \leq, \neq, \geq\}$)

$\sigma_{\text{Kunde.Kdnr} = \text{Telefon.Kdnr}}(\text{Kunde} \times \text{Telefon})$, entspricht $\text{Kunde} \bowtie_{\text{Kunde.Kdnr} = \text{Telefon.Kdnr}} \text{Telefon}$
 - Left Outer Join: $R \Join_{\text{Bedingung}} S$ (Alle R-Tupel plus ggf. passende von S)
 - Right Outer Join: $R \Join_{\text{Bedingung}} S$ (Alle S-Tupel plus ggf. passende von R)
 - Full Outer Join: $R \Join_{\text{Bedingung}} S$ (Alle von R und S)
 - Umbenennung $\rho_{\text{NeuerName}}(\text{AlterName})$

$\sigma_{V2.\text{Nachfolger} = \text{"Java-Projekt"}} \wedge V1.\text{Nachfolger} = V2.\text{Vorgänger}(\rho_{V1}(\text{Voraus}) \times \rho_{V2}(\text{Voraus}))$

$$\pi_{\text{PNR, ALT, ANAME}}(\sigma_{\text{AORT}=\text{'H'}}(\text{ABT}) \bowtie_{\text{ABT.ANR}=\text{PERS.ANR}} \sigma_{\text{ALT} > 30 \wedge \text{ALT} < 34}(\text{PERS}))$$

Super-Schlüssel

- Es sind auch mehrere Elemente erlaubt: (Menge aller Attribute)
- (Eine Attributmenge, über die alle Tupel einer Relation eindeutig identifiziert werden)

Schlüssel zur Identifikation von Tupeln

Primärschlüssel

- **Dauerhaft eindeutig:** Es gibt jede Ausprägung nur einmal (Schlüsseleigenschaft), auch in Zukunft!
- **Unveränderlich:** Die Attribute des PK einer Zeile ändern sich nicht
- Ein Primärschlüssel entsteht indem man einen Superschlüssel verkleinert!

Integritätsbedingungen (Fremdschlüssel)

- Integritätsbedingungen sind Bestimmungen, die eingehalten werden müssen, um die Korrektheit und die logische Richtigkeit der Daten zu sichern.

Wahrheitsanforderungen:

- Können nur durch Vergleich mit der Realität überprüft werden

Logische Integritätsanforderungen:

- Betreffen die Gestalt der einzelnen Tabellen bzw. Relationen und die Beziehungen zwischen den verschiedenen Relationen

VL06/07_SQL

Wichtige Befehle:

```
CREATE TABLE ANG_PRO(  
    PNR INTEGER, ANGNR INTEGER, [...]  
    PRIMARY KEY (ANGNR, PNR),  
    FOREIGN KEY (ANGNR) REFERENCES ANGEST (ANGNR)  
);  
  
INSERT INTO ANGEST values (112, m, k, p, 4500, 3);  
  
ALTER TABLE PROJEKT ADD CONSTRAINT check_name CHECK(P_LEITER is NOT NULL);  
  
ALTER TABLE ANGEST ADD EINSTELLUNGSDATUM DATE DEFAULT '01.01.1000';  
  
ALTER TABLE ANG_PRO DROP COLUMN PROZ_ARB;  
  
DELETE FROM PROJEKT WHERE P_NAME = 'Datawarehouse';  
  
SELECT to_number(to_char(e1.hire_date, 'YYYY')) as year  
  
SELECT SUBSTR(z, INSTR(z, '.')+1, INSTR(z, '.',2) -1) AS d  
  
SELECT lpad(last_name,25) || ' ' || TO_CHAR(salary,'99G999')  
  
CASE WHEN e1.manager_id IS NULL then NULL ELSE e2.first_name END as Chefin  
  
WHERE e1.employee_id = ANY/ALL(SELECT manager_id  
                                FROM HR.employees) and salary > 12000;  
  
WHERE not EXISTS (SELECT * FROM HR.employees e3 );
```

DBS_VL10_Zugriff auf Daten in RDBS aus Programmiersprachen

Probleme: Konzeptionelle Unterschiede zwischen Programmiersprache

- SQL: **deklarativ, mengenorientiert**; Ergebnis einer Anfrage ist eine Relation
- Programmiersprache: **imperativ und/oder objekt-orientiert; satz-orientiert**
- Unterschiede zwischen den **Datentypen**
- JDBC-Nutzung nicht nur für Anfragen (**Data Query Language**) möglich
- Es können auch Daten verändert (**Data Manipulation Language**) werden
- Es kann auch das DB-Schema bearbeitet werden (**Data Definition Language**)

JDBC (Java Database Connectivity)

- Dynamisches SQL: SQL wird als String in Java "zusammengebaut"

```
private static Connection conn;

public static void main(String[] args) throws SQLException {
    conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost", "name", "pw");
    conn.close();
}
```

DDL-Befehl

```
public static void tabelleerstellen() throws SQLException {
    String createOrderItems =
        "CREATE TABLE order_items(" +
        "    order_id NUMBER(8), " +
        "    name VARCHAR2(100)," +
        "    PRIMARY KEY (order_id, name));"
    try (Statement stmt = conn.createStatement()) {
        stmt.executeUpdate(createOrderItems);
    }
}
```

DML-Befehl

- Wichtig: Rückgabewert von `executeUpdate()` : Anzahl der geänderten Datensätze

```
public static void prepareStatement (int id, String name) throws SQLException {
    String insertItem = "INSERT INTO rezept VALUES (?, ?)";
    try (PreparedStatement stmt = conn.prepareStatement(insertItem)) {
        stmt.setInt(1, id);
        stmt.setString(2, name);
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static void createStatement () throws SQLException {
    String insertItem1 = "INSERT INTO order_items VALUES (123,12,'SampleItem1',48.32,12)";
    try (Statement stmt = conn.createStatement()) {
        int num = stmt.executeUpdate(insertItem1);
        System.out.println("Tabelle orderItems "+num+" Zeilen eingefügt!");
    }
}
```

DQL im Detail

```
public static void rezeptAusgeben(int id) throws SQLException {
    try (Statement stmt = conn.createStatement()) {
        String query = "SELECT first_name, last_name, salary "
            + " FROM hr.employees WHERE salary > 5000";
        try (ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()){
                String last_name = rs.getString("last_name");
                double sal = rs.getDouble(3);
                System.out.println(last_name + "\t" + sal);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Metadaten aus der DB in JDBC

- **Metadaten** sind Daten, die Datenbankstrukturen und deren Eigenschaften beschreiben
- **Metadaten** ermöglichen es allgemeinen Zugriffsschichten oder Werkzeugen, mit beliebigen **Datenbankstrukturen zu arbeiten**

JDBC verfügt über 2 Klassen für Metadaten

- **ResultSetMetaData** liefert Informationen zu **ResultSet**
- **DatabaseMetaData** liefert Informationen zu **Datenbanksystem und DB-Schema**

Metadaten des Result-Set

Methoden von ResultSetMetaData

Typinformationen über Spalten im ResultSet

- Name einer Spalte: `String getColumnName(int column)`

```
for (int i = 1; i <= numberOfColumns; i++) {  
    System.out.print(rsmd.getColumnName(i) + "\t");  
}
```

Metadaten zum DB-Schema

Beispielmethode der Klasse DatabaseMetaData (Details und Weiteres siehe JDBC-API):

Signatur	Beschreibung
<code>ResultSet getCatalogs()</code>	Gibt verfügbare Katalognamen zurück
<code>ResultSet getSchemas()</code>	Gibt verfügbare Schemanamen zurück
<code>ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)</code>	Gibt eine Beschreibung der passenden Tabellen zurück (Ergebnis hat 10 Spalten mit Details zu den Tabellen)
<code>boolean supportsX()</code> Beispiel: <code>supportsFullOuterJoins()</code>	Gibt zurück, ob die DB eine spezifische Funktionalität X besitzt
<code>ResultSet getPrimaryKeys(String catalog, String schema, String table)</code>	Gibt eine Beschreibung der Primärschlüssel passender Tabellen zurück

Kapitel 06_Normalisierung

Negative Eigenschaften eines Relationenschemas

Einfügeanomalie

- entsteht, wenn man Informationen zweier Entitätstypen miteinander verknüpft
- *Kunden eintragen, der noch nichts abonniert hat (Eintragen von Nullwerten)*

Änderungsanomalie

- entsteht, wenn man zwei verschiedene Entitätstypen miteinander verknüpft
- Aktualisieren eines Entities erfordert viel mehr Aktionen / Operationen als man sinnvoll benötigt.
- *Preisänderungen oder Adressänderungen in allen betroffenen Datensätzen notwendig*

Löschanomalie

- entsteht, wenn man zwei verschiedene Entitätstypen miteinander verknüpft
- Löschen eines Entities löscht möglicherweise auch Informationen eines anderen Entities
- *Keine Information mehr über Zeitschrift, wenn kein Kunde sie abonniert hat*

Redundanz:

- Änderungsoperationen müssen auf allen Kopien ausgeführt werden, d.h. sie sind ineffizient
- Speicherplatz wird verschwendet
- Wenn die Änderungen fehlerhaft durchgeführt werden, sind die Kopien ggf. inkonsistent

Es gibt gute und schlechte Datenbankentwürfe, gute Datenbankentwürfe sind **frei von Anomalien**

Funktionale Abhängigkeiten

- sind **Integritätsbedingungen**.
- Funktionale Abhängigkeiten: $X \rightarrow Y$
- **Fd-Menge** = (Menge der funktionalen Abhängigkeiten) und wird mit ***F*** bezeichnet
- Die Menge ***F***⁺ aller funktionalen Abhängigkeiten, die von ***F*** abgeleitet werden können, heißt **Hülle** von ***F***
- Für eine ***Fd***-Menge ***F*** und eine funktionale Abhängigkeit $X \rightarrow Y \in F^+$ heißt ***Y*** **voll funktional abhängig** von ***X*** genau dann, wenn es keine echte Teilmenge $X' \subsetneq X$ gibt, so dass $X' \rightarrow Y \in F^+$.

Beispiele:

- ***ANBAUGEBIET*** ist voll funktional abhängig von ***LIEFERANT, ARTIKEL***
- ***QUALITÄT*** ist voll funktional abhängig von ***ANBAUGEBIET***
- ***QUALITÄT*** ist nicht voll funktional abhängig von ***ANBAUGEBIET, LIEFERANT***

Regeln für die Ableitung von neuen funktionalen Abhängigkeiten

Reflexivität (triviale Abhängigkeiten)

aus $X \supseteq Y$ folgt $X \rightarrow Y$, insbes. $A \rightarrow A$

Links können beliebige Attribute hinzugefügt werden:

aus $X \rightarrow Y$ folgt $X \cup Z \rightarrow Y$

beliebige Attribute können gleichzeitig links und rechts hinzugefügt werden:

aus $X \rightarrow Y$ folgt $X \cup Z \rightarrow Y \cup Z$

Transitivität und Pseudotransitivität:

aus $X \rightarrow Y$ und $Y \rightarrow Z$ folgt $X \rightarrow Z$

aus $X \rightarrow Y$ und $W \cup Y \rightarrow Z$ folgt $W \cup X \rightarrow Z$

Rechte Seite Aufteilen und Zerlegen:

aus $X \rightarrow Y \cup Z$ folgt $X \rightarrow Y$ und $X \rightarrow Z$

aus $X \rightarrow Y$ und $X \rightarrow Z$ folgt $X \rightarrow Y \cup Z$

Normalformen

1. **erste Normalform (1NF)**, wenn alle Attribute von R einen atomaren (unteilbaren) Wertebereich haben.

Herstellung der ersten Normalform:

- **Zusammengesetzte Attribute: in mehrere Attribute aufteilen**

ABO	KDNR	KNAME	ADRESSE	ZEITSCHR	PREIS
333	Meier	30459 Hannover, Ricklinger Stadtweg 120	Focus, Focus-Verlag, c't, Heise-Verlag	140.40, 66.20	
333	Meier	31275 Lehrte, Ahornallee 2c	Informatik-Spektrum, Springer	54.30	
666	Niemeier	30444 Hannover, Gosseriede 23	Heise-Verlag, Spiegel-Verlag, Springer	66.20, 145.60	

Zusammengesetztes Attribut (ADRESSE)

Mengenwertiges Attribut (PREIS)

Was ist das? (ADRESSE)

ABO	KDNR	KNAME	PLZ	ORT	STR.	HNR	ZEITSCHR	VERL.
333	Meier	30459	Hannover	Ricklinger Stadtweg	120	Focus	Focus-Verlag	
333	Meier	30459	Hannover	Ricklinger Stadtweg	120	c't	Heise-Verlag	66.20
545	Meier	31275	Lehrte	Ahornallee	2c	Informatik-Spektrum	Springer	54.30
464	Müller	30444	Hannover	Gosseriede	23	c't	Heise-Verlag	66.20
464	Müller	30444	Hannover	Gosseriede	23	IX	Heise-Verlag	66.00
666	Niemeier	30459	Hannover	Stammesstr.	55	c't	Heise-Verlag	66.20

2. **zweite Normalform (2NF)**, wenn jedes Nichtschlüsselattribut voll funktional abhängig ist von jedem Schlüssel. Ist immer dann verletzt, wenn der Wert eines Nichtschlüsselattributs funktional von einer Teilmenge eines Schlüssels abhängt.

- **Redundanzen kommen auf**

3. **dritte Normalform (3NF)**, wenn für jede Funktionale Abhängigkeit $X \rightarrow A \in F^+$, $A \notin X$ gilt:

- X enthält einen Schlüssel
- oder
- Entweder A ist Schlüsselattribut

(ist **nicht** in (3NF), wenn X kein Schlüssel enthält und A Nichtschlüsselattribut ist)

(keine Abhängigkeit eines Nichtschlüsselattributs von einer Attributmenge, die keinen Schlüssel enthält)

Erstes Beispiel:

$RID \rightarrow NAME \in F^+$:

- $NAME$ ist kein Schlüsselattribut und
- RID enthält keinen Schlüssel

3NF ist hierdurch verletzt!

REZEPT	<u>RID</u>	NAME	<u>SCHRITT</u>	BESCHREIBUNG
...

$RID, SCHRITT \rightarrow BESCHREIBUNG \in F^+$:

- $BESCHREIBUNG$ ist kein Schlüsselattribut
- $RID, SCHRITT$ ist (und damit enthält) aber einen Schlüssel

3NF ist hierdurch nicht verletzt!

<u>RID</u>	NAME	<u>SCHRITT</u>	BESCHREIBUNG
1	Pfannkuchen	1	Hefe in warmen Wasser auflösen
...

Wenn durch $X \rightarrow A$ die 3NF verletzt wird, wird A aus der Relation entfernt und in eine neue Relation mit passendem Schlüssel verschoben.

<u>RID</u>	<u>SCHRITT</u>	BESCHREIBUNG
1	1	Hefe in warmen Wasser auflösen
...

GERICHT	<u>RID</u>	NAME
1	1	Pfannkuchen
...

Zweites Beispiel

$VERLAG \rightarrow SITZ \in F^+$:

- $SITZ$ ist kein Schlüsselattribut und
- $\{VERLAG\}$ enthält keinen Schlüssel

3NF ist hierdurch verletzt!

ZEITSCHR	<u>ZNR</u>	ZNAME	VERLAG	SITZ	PREIS
...

<u>ZNR</u>	ZNAME	VERLAG	SITZ	PREIS
2	c't	Heise-Verlag	Hannover	66.20
...

Test auf 3NF

$F = \{ZNR \rightarrow ZNAME, ZNR \rightarrow VERLAG, ZNR \rightarrow PREIS, ZNAME \rightarrow ZNR\}$:

<u>ZNR</u>	ZNAME	VERLAG	PREIS
2	c't	Heise-Verlag	66.20
...

VERLAG	SITZ
Heise-Verlag	Hannover
...	...

Man kann folgendes beweisen:

- Wenn keine der Fds in F die 3NF verletzt, dann verletzt auch keine der Fds in F^+ die 3NF.

→ Daraus folgt: es reicht, F zu betrachten (statt F^+)!

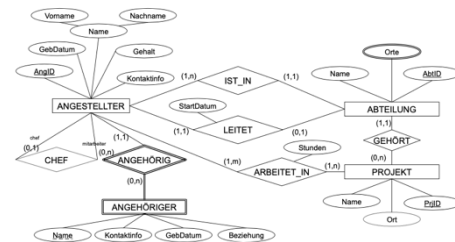
Kapitel 7: Transformation von ER- Modellen ins Relationen Modell

Schritt 1: Aus allen Entity-Typen wird eine Relation

ANGESTELLTER	<u>AnglID</u>	GebDatum	Vorname	Nachname	Gehalt	Kontaktinfo
--------------	---------------	----------	---------	----------	--------	-------------

ABTEILUNG	Name	<u>AbtID</u>
-----------	------	--------------

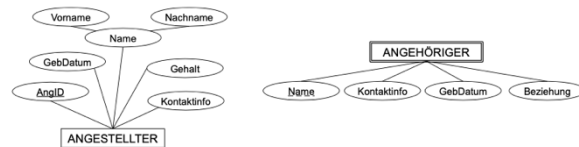
PROJEKT	Name	Ort	<u>PrjID</u>
---------	------	-----	--------------



Schritt 2: Relationen für schwache Entity-Typen

ANGEHÖRIGER	<u>AnglID</u>	Name	Kontaktinfo	GebDatum	Beziehung
-------------	---------------	------	-------------	----------	-----------

ANGESTELLTER	<u>AnglID</u>	GebDatum	Vorname	Nachname	Gehalt	Kontaktinfo
--------------	---------------	----------	---------	----------	--------	-------------



Primärschlüssel des besitzenden Entity-Typs wird mitgenommen (AnglID) und als Fremdschlüssel deklariert

Schritt 3: Binäre 1:1 Beziehungen

ANGESTELLTER	<u>AnglID</u>	GebDatum	Vorname	Nachname	Gehalt	Kontaktinfo
--------------	---------------	----------	---------	----------	--------	-------------

LEITET	AnglID	<u>AbtID</u>	StartDatum
--------	--------	--------------	------------

ABTEILUNG	Name	<u>AbtID</u>
-----------	------	--------------



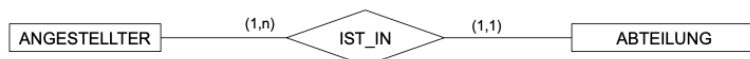
2 Fremdschlüssel, einer davon wird Primärschlüssel (einer mit totaler Beteiligung, sofern vorhanden)

Schritt 4: Binäre 1:n Beziehungen

ANGESTELLTER	<u>AnglID</u>	GebDatum	Vorname	Nachname	Gehalt	Kontaktinfo
--------------	---------------	----------	---------	----------	--------	-------------

IST_IN	AnglID	<u>AbtID</u>
--------	--------	--------------

ABTEILUNG	Name	<u>AbtID</u>
-----------	------	--------------



Schritt 5: Binäre n:m Beziehungen

ANGESTELLTER	<u>AnglID</u>	GebDatum	Vorname	Nachname	Gehalt	Kontaktinfo
--------------	---------------	----------	---------	----------	--------	-------------

ARBEITET_IN	AnglID	<u>PrjID</u>	Stunden
-------------	--------	--------------	---------

PROJEKT	Name	Ort	<u>PrjID</u>
---------	------	-----	--------------

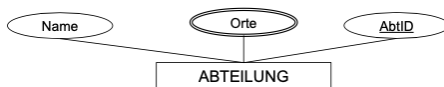


Die Primärschlüssel der beiden beteiligten Relationen werden eingefügt und gemeinsam der Primärschlüssel der neuen Relation

Schritt 6: Mehrwertige Attribute Weitere Aspekte:

ABTEILUNG	Name	<u>AbtID</u>
-----------	------	--------------

Orte	<u>AbtID</u>	Ort
------	--------------	-----



Mehrstellige Relationen

AUTO	<u>AutoID</u>	...
------	---------------	-----

BANK	<u>BankID</u>	...
------	---------------	-----

PERSON	<u>PersonID</u>	...
--------	-----------------	-----

FINANZIERT	<u>AutoID</u>	<u>BankID</u>	<u>PersonID</u>	Zinssatz
------------	---------------	---------------	-----------------	----------

