# 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 - (mehre 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 eigeschränkten 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 eins 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

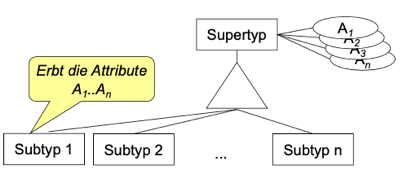
* Mengenwertige und zusammengesetzte Attribute

### 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 "**herausfaktorisiert**" 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 / Vorgesetzer
* **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**

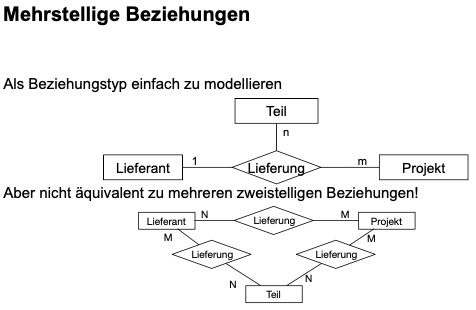
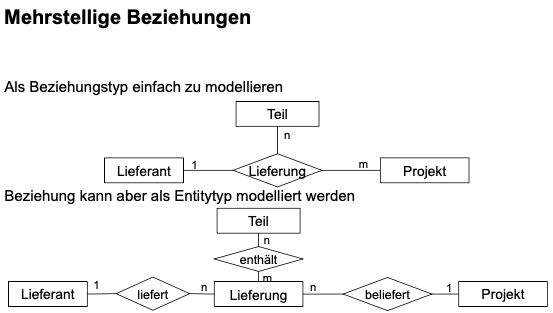
* verschieden Wertemengen eines Attributs

**Konflikte zwischen Einschränkungen**

* verschiedene Schlüssel, verschiedene Kardinalitäten

**Vererbungshierarchien**

****



# Kapitel 3 Implementierungsmodell

Relationenschema Personen(Name, Straße, TelNummer)

dom(Name) = dom(Strasse) = {Zeichenkette}

dom(TelNummer) = {8 - stellige Zahl}

## Relationenmodell:

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

## Relationale Algebra:

* Anfragen lassen sich in Relationenalgebra 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)*

σP(R) - πName(Kunde)   
Bsp. σGehalt > 50000(πName, Gehalt, Alter(Person))

* Kreuzprodukt -> **Verbund** *(Vergleichsoperator Θ ∈ {<, =, >, ≤,≠, ≥})*

σKunde.Kdnr = Telefon.Kdnr(Kunde × Telefon), entspricht Kunde ⨝Kunde.Kdnr = Telefon.Kdnr Telefon

* Left Outer Join: R ⟕Bedingung S *(Alle R-Tupel plus ggf. passende von S)*
* Right Outer Join: R ⟖Bedingung S *(Alle S-Tupel plus ggf. passende von R)*
* Full Outer Join: R ⟗Bedingung S *(Alle von R und S)*
* Umbenennung ρNeuerName(AlterName)

*σV2.Nachfolger="Java-Projekt" ∧ V1.Nachfolger=V2.Vorgänger(ρV1(Voraus) x ρV2(Voraus))*

πPNR, ALT, ANAME(σAORT='H'(ABT) ⨝ABT.ANR=PERS.ANR σALT > 30 ∧ ALT < 34(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)

);

INSTER 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:@localho", "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

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

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

# Kapitel 06\_Normalisierung

## Negative Eigenschaften eines Relationenschematas

**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 🡪 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 → 𝑌 ∈ 𝐹+ heißt 𝑌 **voll funktional abhängig** von 𝑋 genau dann, wenn es keine echte Teilmenge 𝑋′ ⊊ 𝑋 gibt, so dass 𝑋′ → 𝑌 ∈ 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 ⊇ 𝑌 folgt 𝑋 → 𝑌, insbes. 𝐴 → 𝐴

**Links können beliebige Attribute hinzugefügt werden:**

aus 𝑋 → 𝑌 folgt 𝑋 ∪ 𝑍 → 𝑌

**beliebige Attribute können gleichzeitig links und rechts hinzugefügt werden**:

aus 𝑋 → 𝑌 folgt 𝑋 ∪ 𝑍 → 𝑌 ∪ 𝑍

**Transitivität und Pseudotransitivität**:

aus 𝑋 → 𝑌 und 𝑌 → 𝑍 folgt 𝑋 → 𝑍

aus 𝑋 → 𝑌 und 𝑊 ∪ 𝑌 → 𝑍 folgt 𝑊 ∪ 𝑋 → 𝑍

**Rechte Seite Aufteilen und Zerlegen**:

aus 𝑋 → 𝑌 ∪ 𝑍 folgt 𝑋 → 𝑌 und 𝑋 → 𝑍

aus 𝑋 → 𝑌 und 𝑋 → 𝑍 folgt 𝑋 → 𝑌 ∪ 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**

Ein Bild, das Tisch enthält.

Automatisch generierte BeschreibungEin Bild, das Karte enthält.

Automatisch generierte Beschreibung

1. **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**

1. **dritte Normalform (3NF),** wenn für jede Funktionale Abhängigkeit 𝑋 → 𝐴 ∈ 𝐹+, 𝐴 ⊄ 𝑋 gilt:

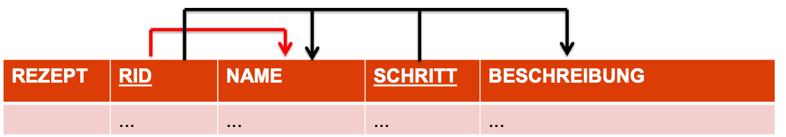
* 𝑋 enthält einen Schlüssel

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:

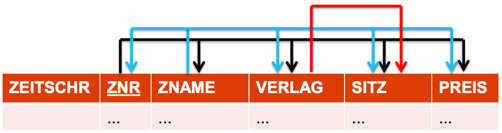
**𝑅𝐼𝐷 → 𝑁𝐴𝑀𝐸 ∈ 𝐹+:**

* 𝑁𝐴𝑀𝐸 ist kein Schlüsselattribut und
* 𝑅𝐼𝐷 enthält keinen Schlüssel

**𝑅𝐼𝐷, 𝑆𝐶𝐻𝑅𝐼𝑇𝑇 → 𝐵𝐸𝑆𝐶𝐻𝑅𝐸𝐼𝐵𝑈𝑁𝐺 ∈ 𝐹+:**

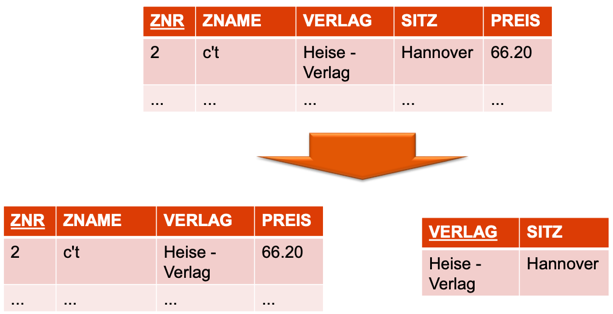
* 𝐵𝐸𝑆𝐶𝐻𝑅𝐸𝐼𝐵𝑈𝑁𝐺 ist kein Schlüsselattribut
* 𝑅𝐼𝐷, 𝑆𝐶𝐻𝑅𝐼𝑇𝑇 ist (und damit enthält) aber einen Schlüssel

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

****

### Zweites Beispiel

**𝑉𝐸𝑅𝐿𝐴𝐺 → 𝑆𝐼𝑇𝑍 ∈ 𝐹+:**

* ****𝑆𝐼𝑇𝑍 ist kein Schlüsselattribut und
* {𝑉𝐸𝑅𝐿𝐴𝐺} enthält keinen Schlüssel

### Test auf 3NF

𝐹 = {𝑍𝑁𝑅 → 𝑍𝑁𝐴𝑀𝐸, 𝑍𝑁𝑅 → 𝑉𝐸𝑅𝐿𝐴𝐺, 𝑍𝑁𝑅 → 𝑃𝑅𝐸𝐼𝑆, 𝑍𝑁𝐴𝑀𝐸 → 𝑍𝑁𝑅}:

Man kann folgendes beweisen:

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

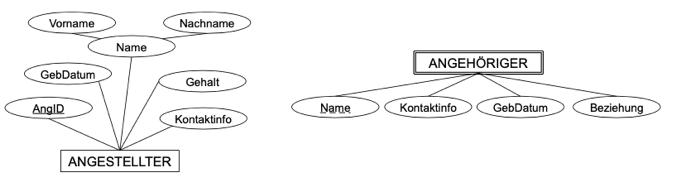
**🡪 Daraus folgt: es reicht, 𝐹 zu betrachten (statt 𝐹+)!**

# Kapitel 7: Transformation von ER- Modellen ins Relationen Modell

**Schritt 1: Aus allen Entity-Typen wird eine Relation**

**Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung**

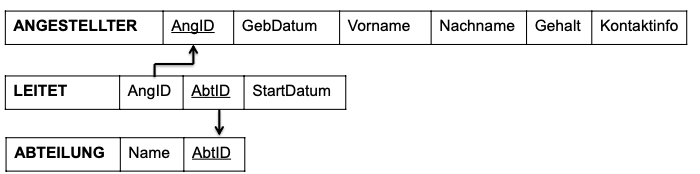
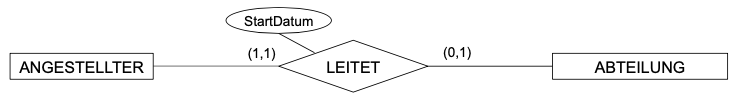
**Schritt 2: Relationen für schwache Entity-Typen**

**Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung**

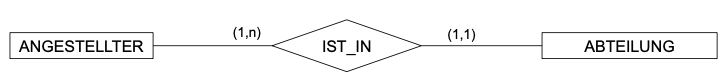
*Primärschlüssel des besitzenden Entity-Typs wird mitgenommen (AngID) und als Fremdschlüssel deklariert*

**Schritt 3: Binäre 1:1 Beziehungen**

****

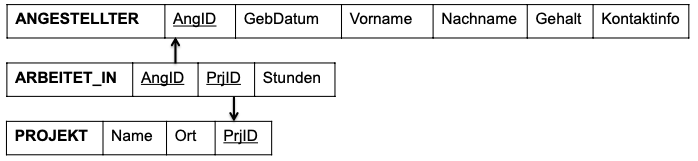
*2 Fremdschlüssel, einer davon wird Primärschlüssel (einer mit totaler Beteiligung, sofern vorhanden)*

**Schritt 4: Binäre 1:n Beziehungen**

**Ein Bild, das Tisch enthält.

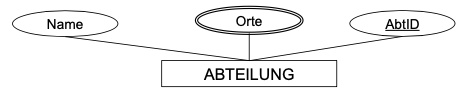
Automatisch generierte Beschreibung**

**Schritt 5: Binäre n:m Beziehungen**

****

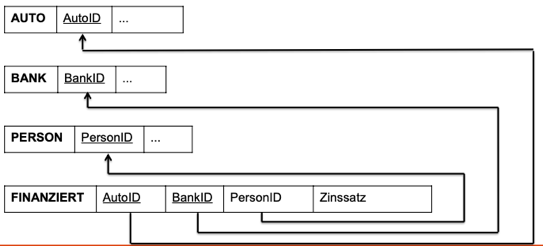
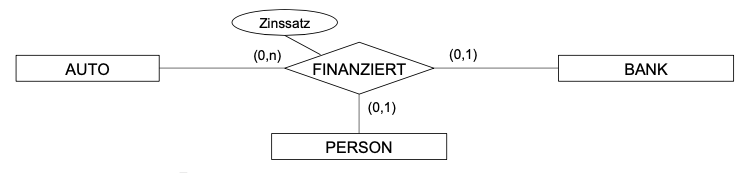
*Die Primärschlüssel der beiden beteiligten Relationen werden eingefügt und gemeinsam der Primärschlüssel der neuen Relation*

**Schritt 6: Mehrwertige Attribute Weitere Aspekte:**

**Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung**

**Mehrstellige Relationen**

****