Einführung in die Wirtschaftsinformatik I

WS 2024/25

Relationale Datenbanken & SQL

Dr. Rainer Wieching

Zusammenfassung der ERM und Datenorganisation

Das **ERM** modelliert **statische Datenstrukturen** in einer gegebenen Domäne.

Planung einer Datenbank gliedert sich unter Bezug des ER-Modells in die **5 Schritte**:

1. Entw. des semantischen Modells.
2. Erfassung der Entitätstypen.
3. Zuordnung der Attribute.
4. Analyse der Entitätsbeziehungen.
5. Erstellung des ER-Diagramms

Die Datenorganisation gliedert sich in die **logische** und **physische Datenorganisation**.

Datenbankmanagementsysteme (DBMS) sind eine **Sammlung von Programmen zum Erstellen und Verwalten einer Datenbank**, die gleichzeitigen effizienten Datenzugriff ermöglicht.

Gliederung

1. Entwurf des Relationalen Datenbankschema

    Mapping von Entitätstypen

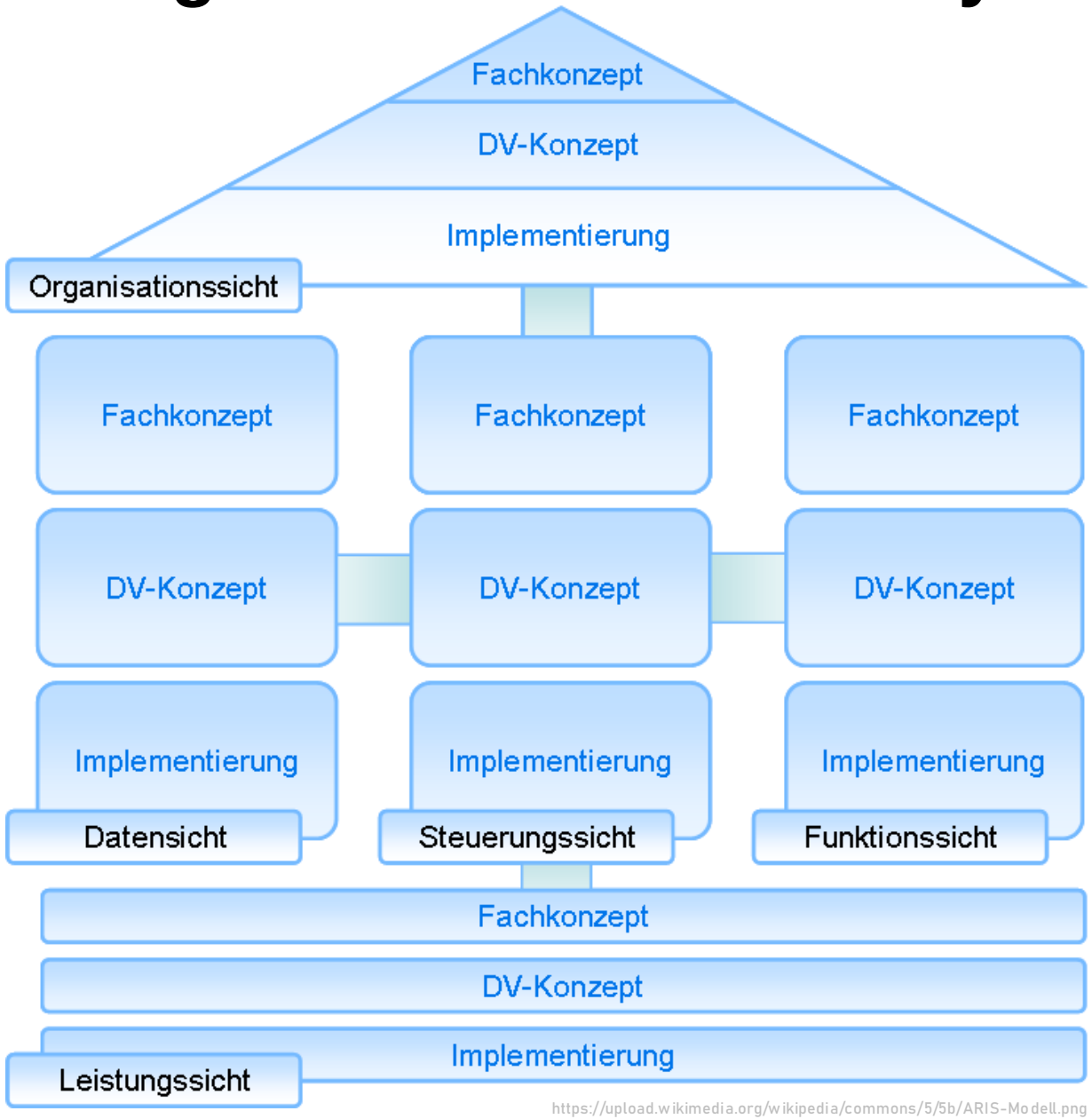
    Nicht-elementare Attribute

    Relationen

2. Abfragen von Daten aus relationalen Datenbanken

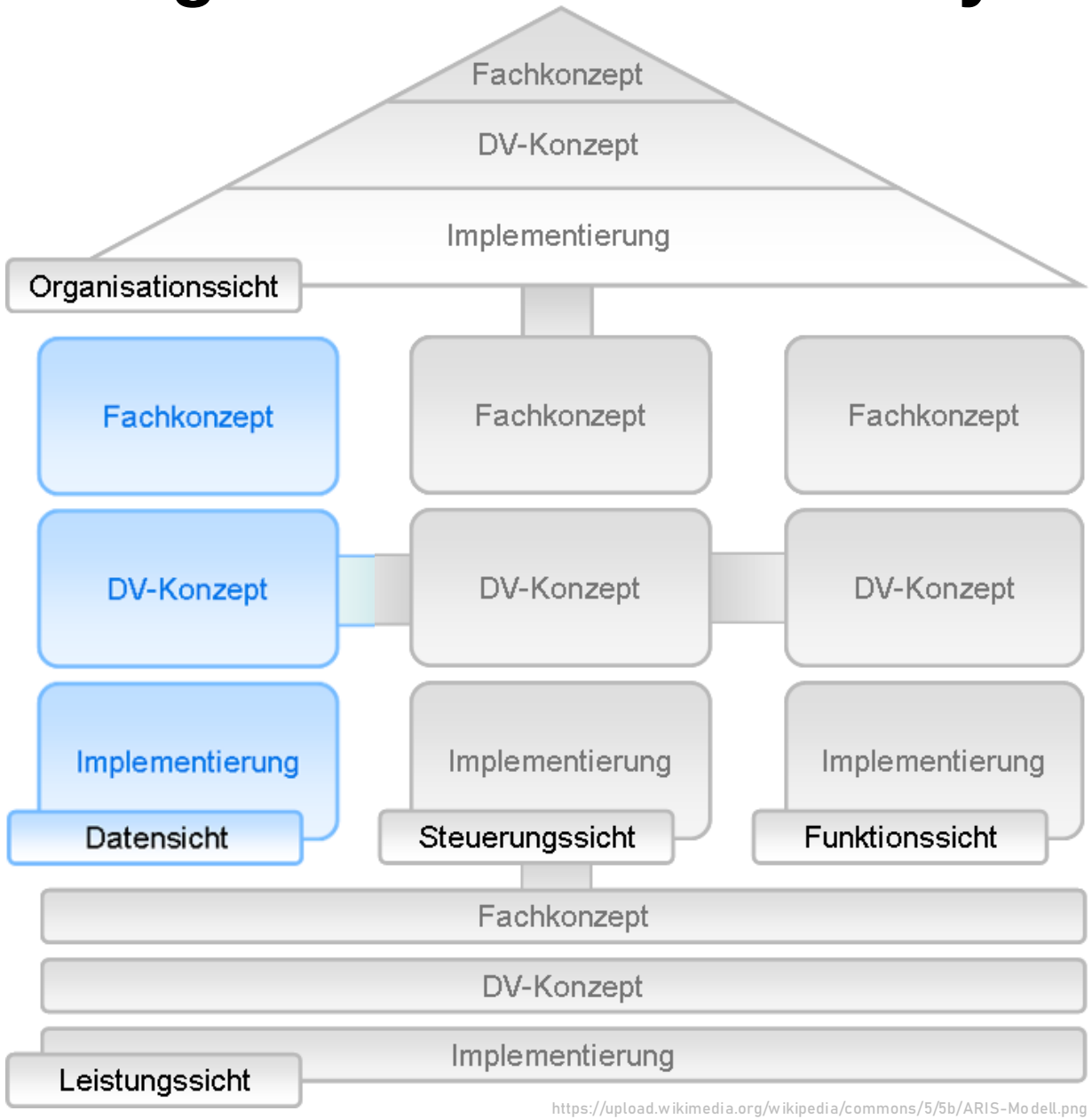
    SQL

Architektur integrierter Informationssysteme (ARIS)



https://upload.wikimedia.org/wikipedia/commons/5/5b/ARIS-Modell.png

Architektur integrierter Informationssysteme (ARIS)



https://upload.wikimedia.org/wikipedia/commons/5/5b/ARIS-Modell.png

Gliederung

1. Entwurf des Relationalen Datenbankschema

- Mapping von Entitätstypen
- Nicht-elementare Attribute
- Relationen

2. Abfragen von Daten aus relationalen Datenbanken

- SQL

Mapping ER-Modell in Relationales Datenbankschema

ER-Modell als Ausgangspunkt für Entwicklung/ Anpassung datenbankbasierter Anwendungen.

Jedoch:

ER-Modell ist nicht unmittelbar für Aufbau eines Datenbankschemas geeignet.

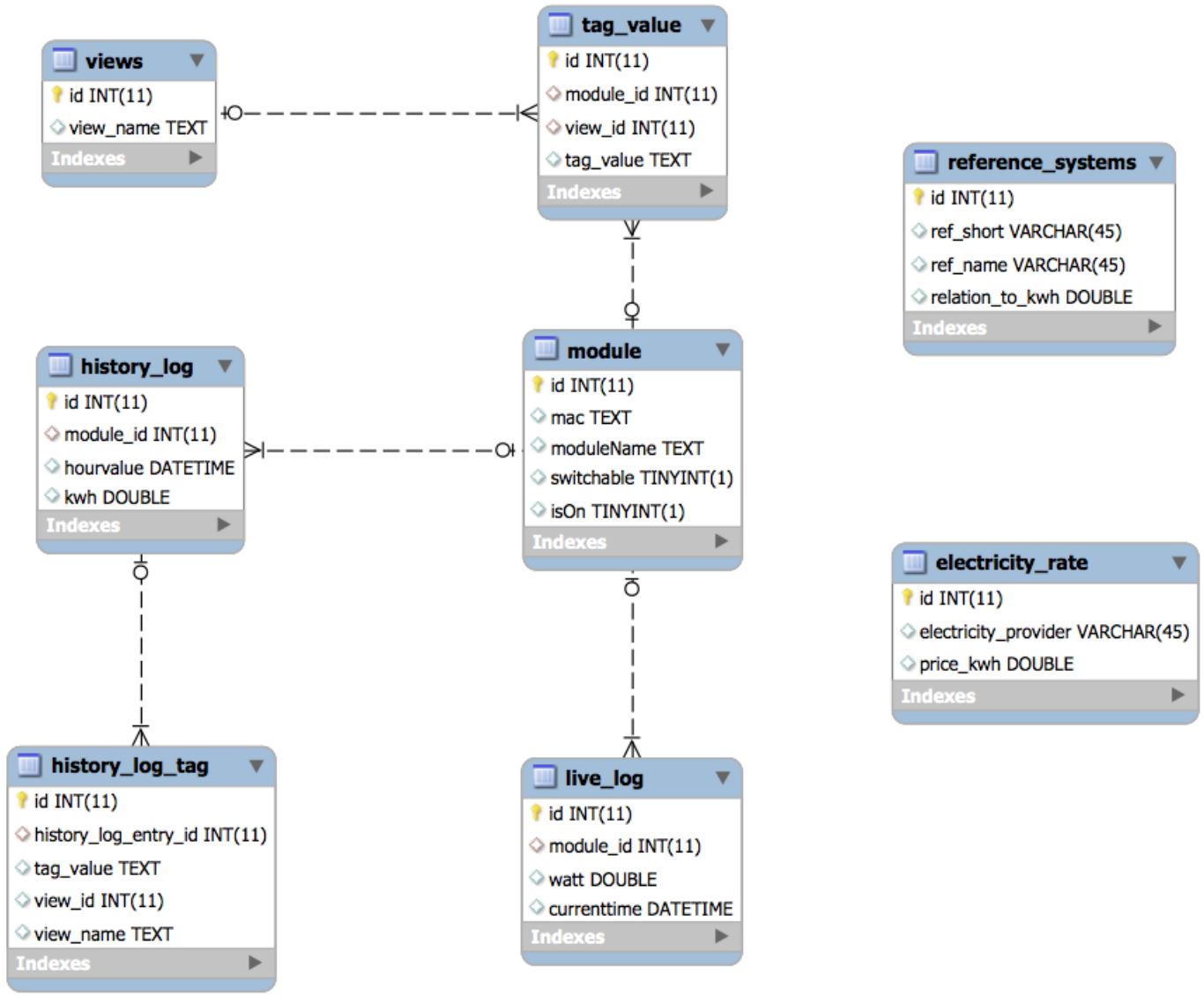
Mittels **Transformation-Regeln** lässt sich ein ER-Modell (Entitytypen, Attribute, Beziehungen etc.) in ein relationales Schema (Tabellen, Schlüssel, etc.) eindeutig überführen.

Beispiel: Datenbank eines Energiemonitor

User Interface



Graphisches DB-Schema



- *reference_system*: Speicherung der Einheiten, in denen die Energiewerte angezeigt werden (EUR, kWh, ...).
- *electricity_rate:* Aktuelle Energieversorger sowie der Arbeitspreis für eine kWh.
- *module:* Liste der Mess-Module (SmartPlugs) und deren Eigenschaften
- *live_log*: Speicherung aktuell gemessener Energieverbrauch der einzelnen Mess-Module in Watt.
- *tag_value:* Gruppierung von Mess-Modulen zu semantischen Einheiten (z.B. Raum-Gruppierung, Gerätetyp-Gruppierung, etc.)
- *views:* Eine View besteht aus Menge von Modulen, deren Verbrauch angezeigt wird.
- *history_log:* Speicherung vergangener Energieverbräuche auf Stundenbasis.
- *history_log_tag:* Speichert die vergangene Zuordnung von Modulen zu einen Tag.

Relationenschema

energy13.electricity_rate (id, ref_short, ref_name, relation_to_kwh)

energy13.reference_system (id, electricity_probider, price_kwh)

energy13.module (id, mac, moduleName, switchable, isOn)

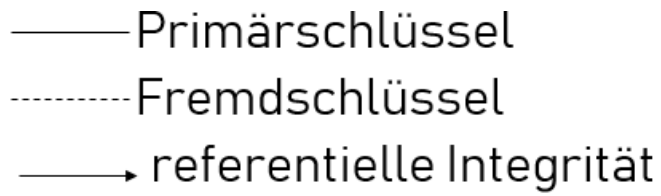
energy13.tag_value (id, module_id, view_id, tag_value)

energy13.views (id, view_name)

energy13.live_log (id, module_id, watt, currenttime)

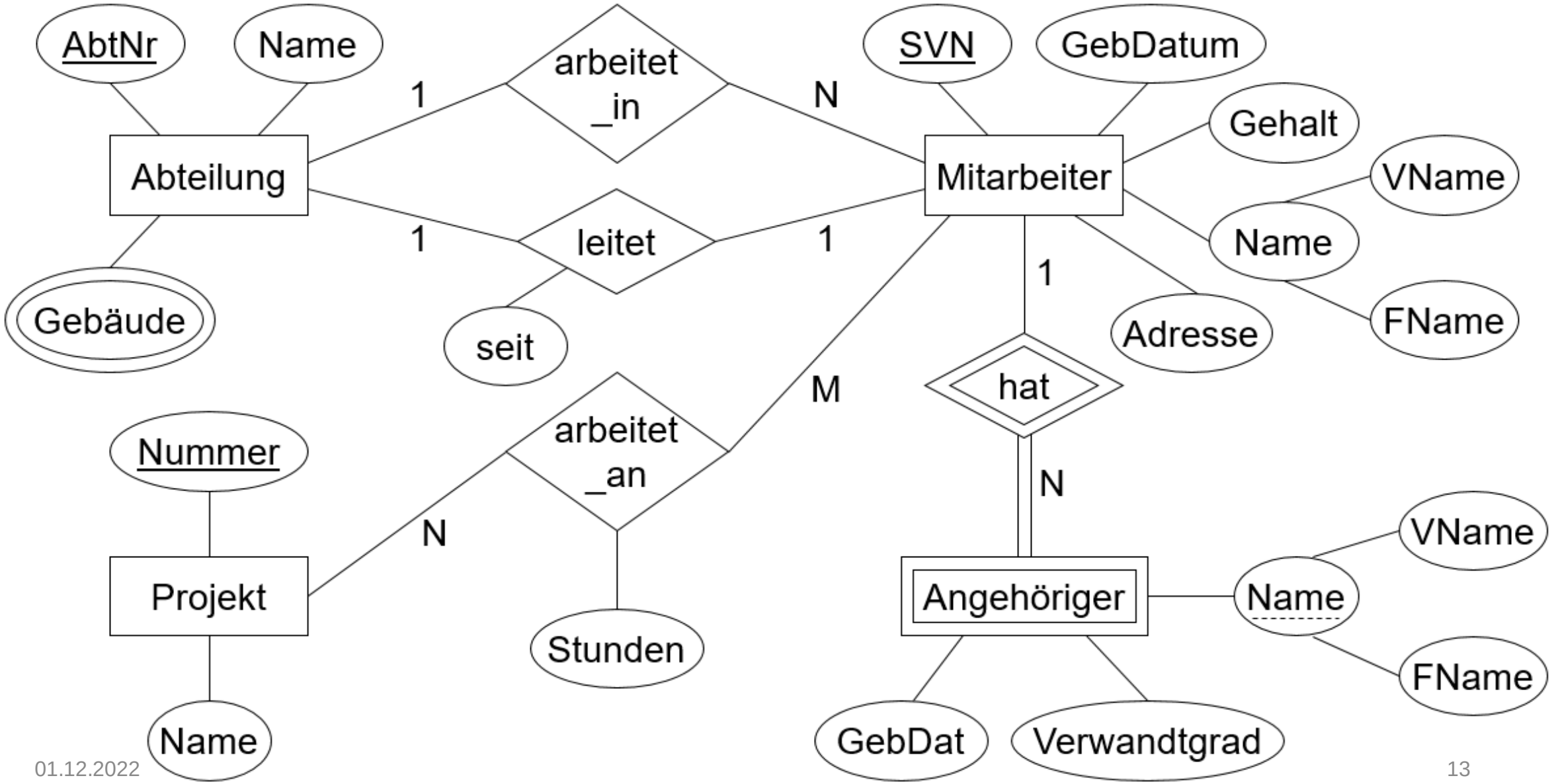
energy13.history_log (id, module_id, hourvalue, kwh)

energy13.history_log_tag (id, history_log_entry_id, tag_value, view_id, view_name)



Überführung eines ER-Modell “Musterunternehmen” in ein DB-Schema

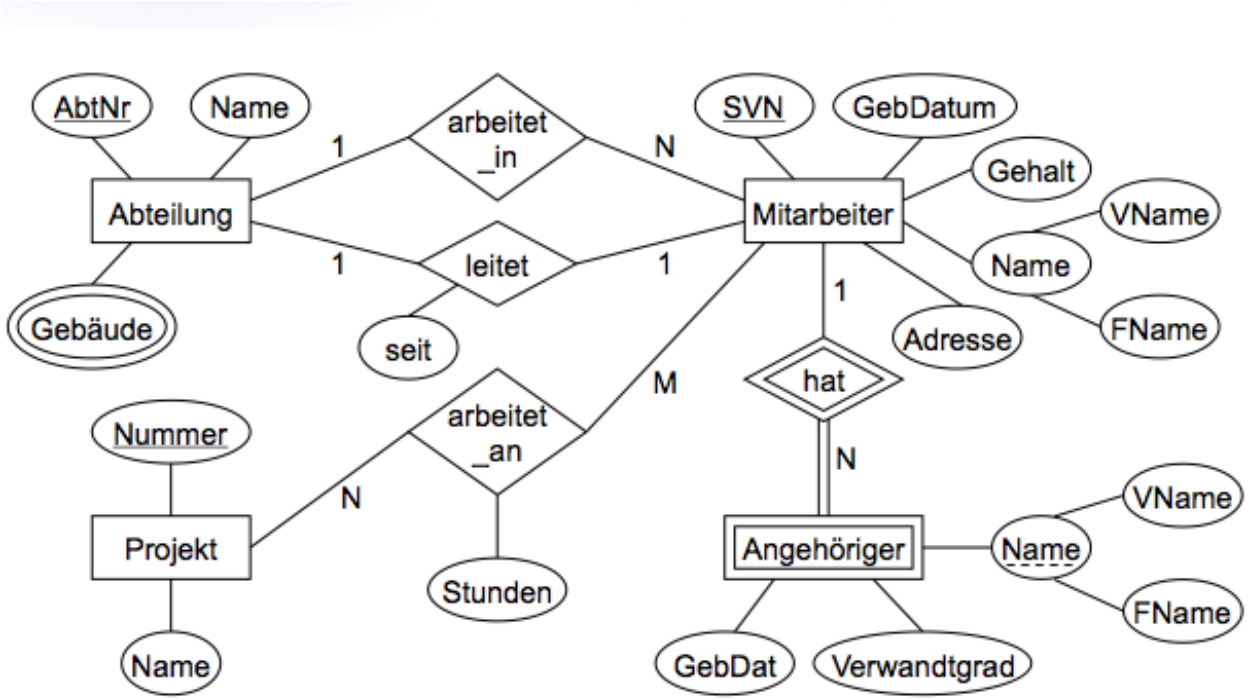
ERM des Musterunternehmens



Umsetzung von Entitätstypen und Attributen I

Transformationsregel:

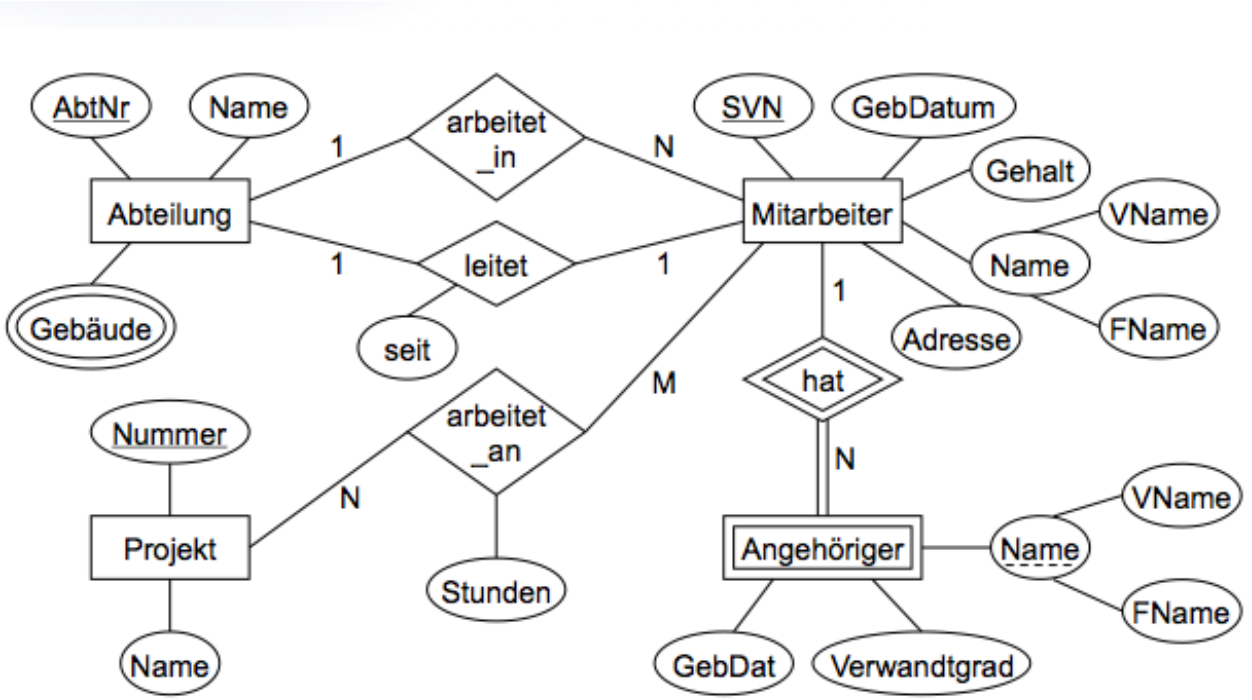
- Für jeden Entitytyp E wird eine Relation (Tabelle) S erstellt
- Alle einfachen Attribute von E werden zu Spalten der Tabelle
- Als Primärschlüssel der Relation S fungiert der identifizierende Schlüssel des Entitytypen E
- Beispiel:
  Abteilung (<u>AbtNr</u>, Name, …)
  Mitarbeiter (<u>SVN</u>, GebDat, Gehalt, Adresse, …)
  Projekt (<u>Nummer</u>, Name)



Umsetzung von Entitätstypen und Attributen II

Ein schwacher Entitytyp E wird in eine Relation R umgesetzt

- A) die alle Attribute des Entitytyps E als Spalten enthält,
- B) die erweitert wird um den Primärschlüssel des dominierenden Entitytyps, und
- C) deren Primärschlüssel sich aus B) und dem partiellen Schlüssel ergibt.

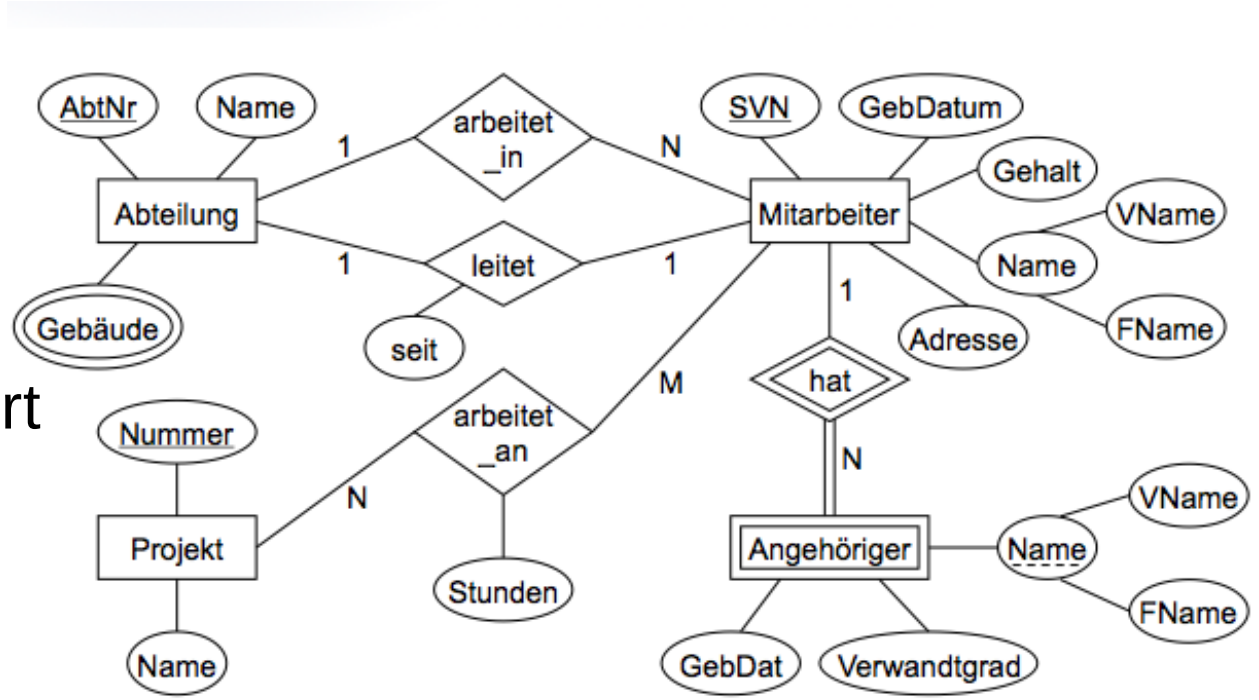


Beispiel: **Angehöriger(SVN, Name, GebDat, Verwandtgrad)**

Umsetzung von nicht elementaren Attributen I

Zusammengesetzte Attribute
in Entitytyp E

Relation S wird um die einzelnen Bestandteile des zusammengesetzten Attributes als einzelne Spalten erweitert

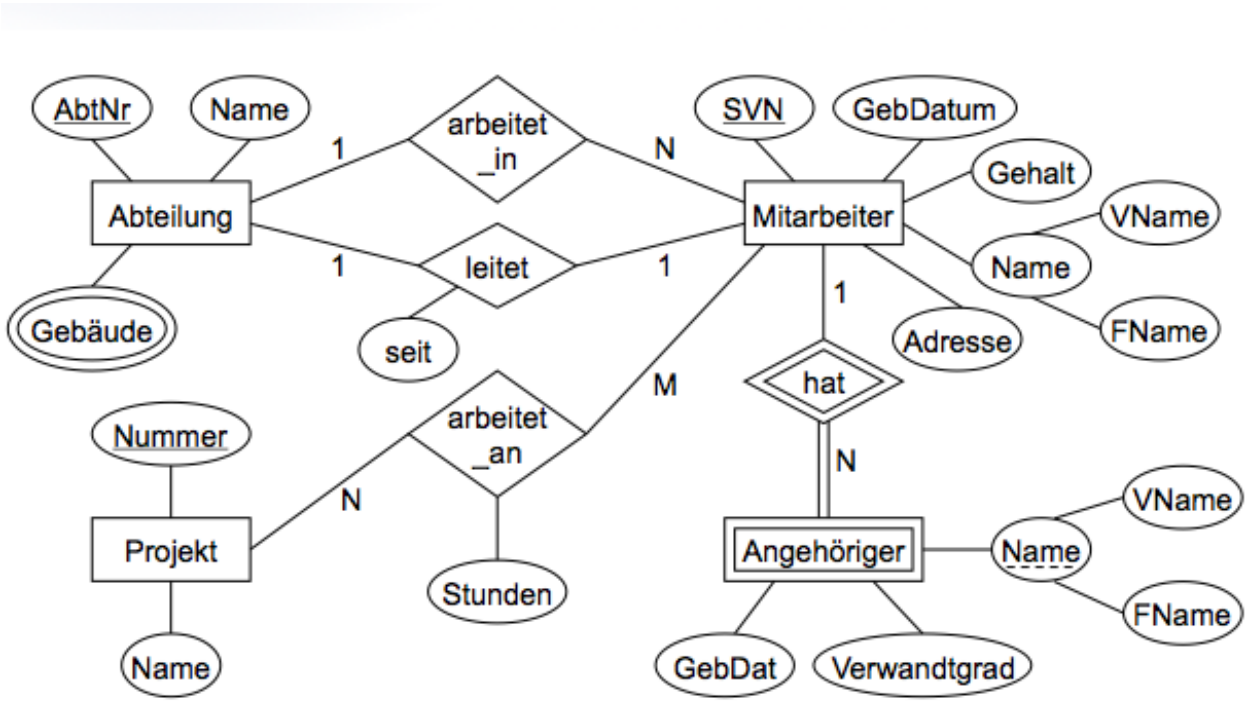


Beispiel: **Mitarbeiter (SVN, ~~Name~~, Vorname, Familienname, …**

Umsetzung von nicht elementaren Attributen II

Mehrwertige Attribute in Entitytyp E

- Für jedes mehrwertige Attribut wird eine eigene Relation R gebildet
- Deren Primärschlüssel setzt sich aus dem Attribut selbst und dem Primärschlüssel der zugehörigen Relation S zusammen.
- Beispiel:
  Abteilung (<u>AbtNr</u>, Name, ~~Gebäude~~)
  Gebäude (<u>AbtNr</u>. <u>Gebäude</u>)

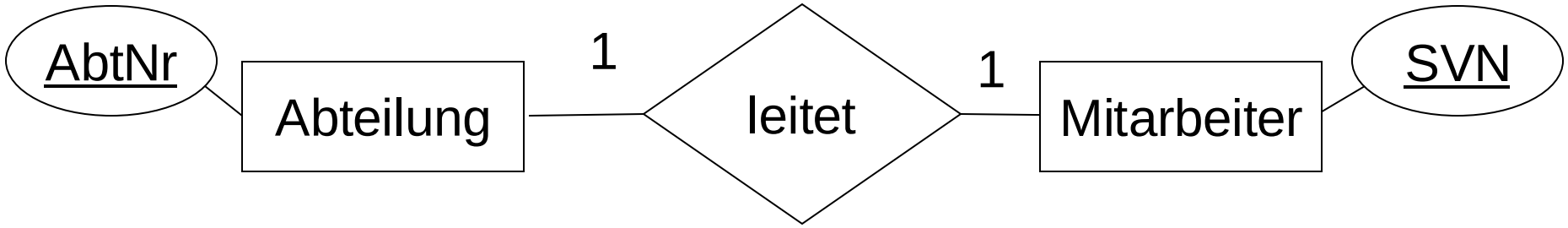


Umsetzung von Beziehungstypen I

1:1-Beziehungstyp zwischen Entity-Typen S und T

In diesem Fall kann **eine** der zu S und T gehörenden Tabellen um ein **Fremdschlüssel-Attribut** (Primärschüssel S bzw. T) und die Attribute des Beziehungstyps erweitert werden

Beispiel:



Abteilung(**AbtNr**, …)
Mitarbeiter(AbtNr, **SVN**, …)

oder

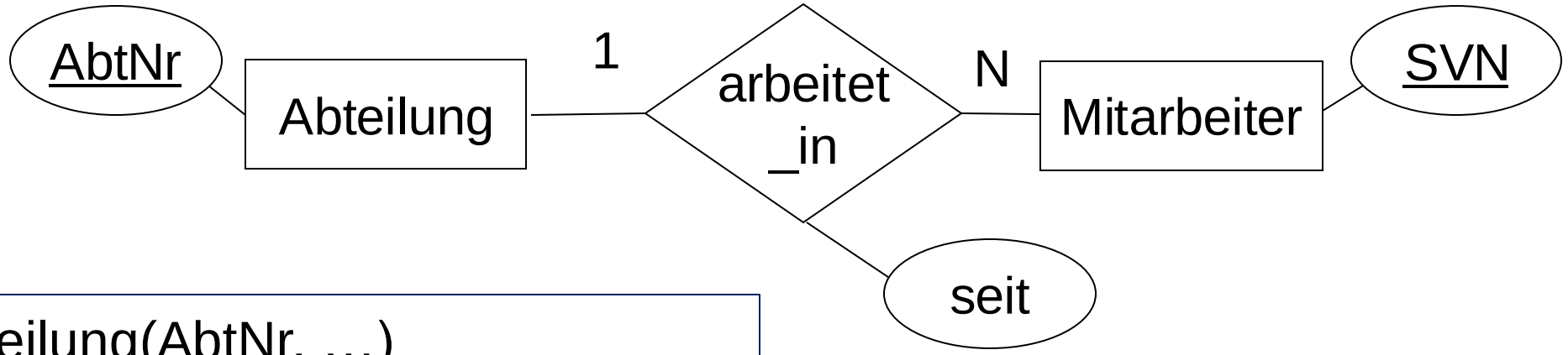
Abteilung(**SVN**, AbtNr, …)
Mitarbeiter(**SVN**, …)

Umsetzung von Beziehungstypen II

1:N-Beziehungstyp zwischen Entity-Typ S und T

Erweiterung der zu T gehörenden Relation um ein Fremdschlüsselattribut (Primärschüssel von S) und die Attribute des Beziehungstyps.

Beispiel:



Abteilung(<u>AbtNr</u>, …)

Mitarbeiter(**AbtNr**, **seit**, <u>SVN</u>, …)

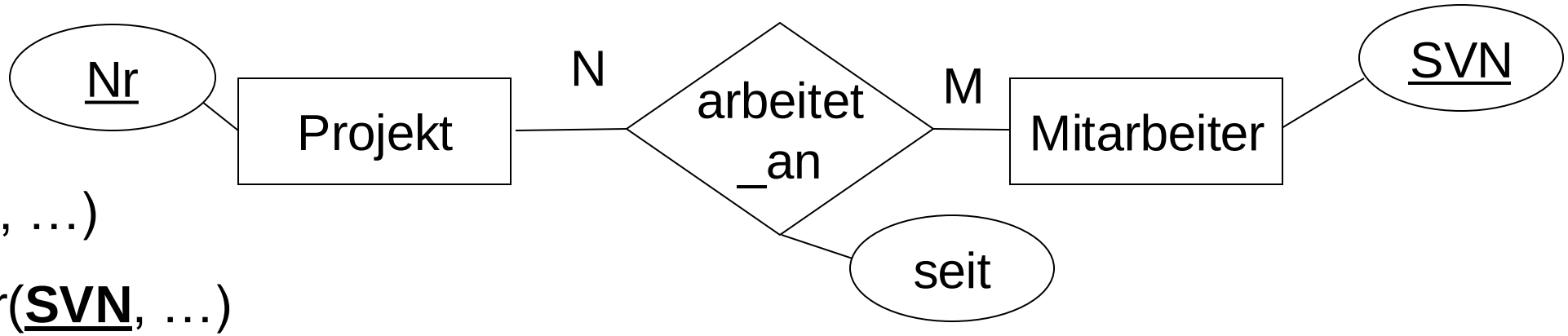
Umsetzung von Beziehungstypen III

N:M Beziehungstyp zwischen zwei Entitytypen R und S

Neue Relation B mit Primärschlüssel zusammengesetzt aus den Primärschlüsseln von R und S

Alle Attribute des Beziehungstyps werden in die Relation B übernommen

Beispiel:



Projekt(**Nr**, …)

Mitarbeiter(**SVN**, …)

arbeitet_an(**Nr, SVN**, seit)

Diese Vorgehensweise kann auch zur Abbildung von 1:1 und 1:N Beziehungen genutzt werden. Sie bietet sich zur Abbildung von 1:1 und 1:N Beziehungen immer dann an, wenn bei konkreten Ausprägungen nur wenige Beziehungen existieren.

Umsetzung von Beziehungstypen IV

Beziehungstypen vom Grad n>2

- Neue Relation B mit Primärschlüssel aller beteiligten Entitytypen als Fremdschlüssel-Attribute
- B erweitert um alle einfachen Attribute und die einzelnen Komponenten der zusammengesetzten Attribute des n-ären Beziehungstypen
- Der Primärschlüssel von B setzt sich je nach Art des Beziehungstypen (1:1:1, 1:1:N, 1:N:M, N:M:P) aus Kombinationen der Fremdschlüssel-Attribute der beteiligten Entitytypen zusammen

Spezialisierung / Generalisierung

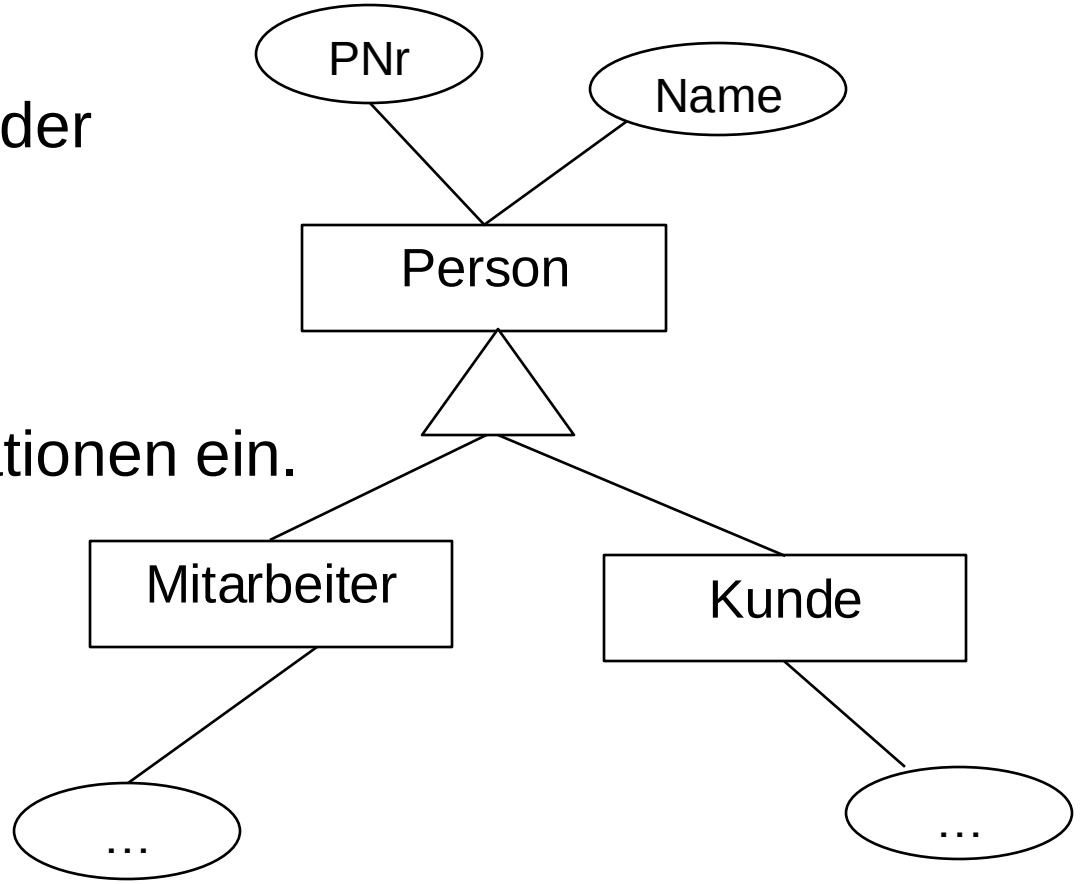
Transformationsregel:

- Für jeden Entitytypen der Oberklasse und der Unterklasse wird eine Relation gebildet.
- Der Primärschlüssel der Oberklasse geht hierbei in den Primärschlüssel der zu den Unterklassen gehörenden Relationen ein.
- Beispiel:

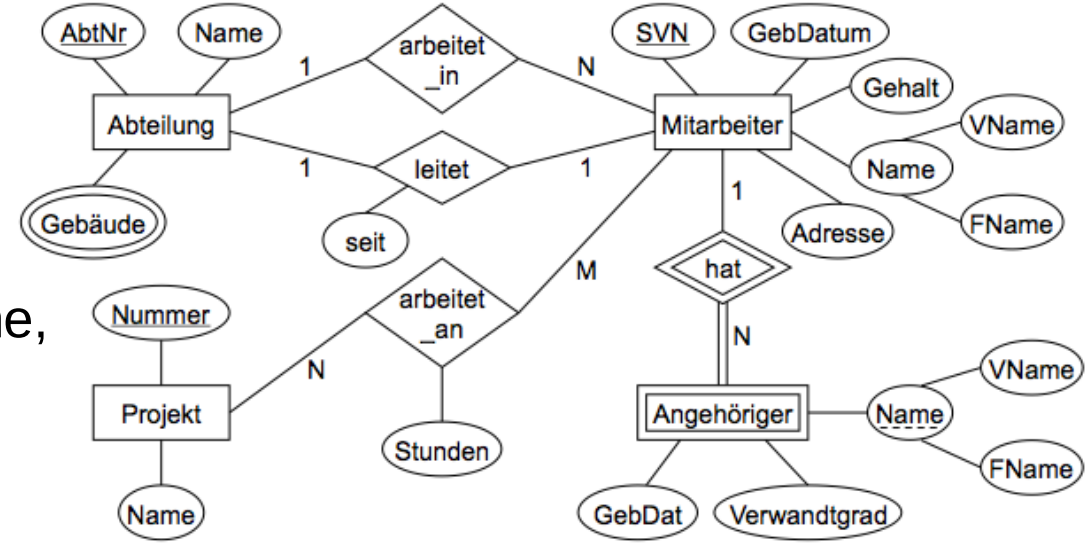
  Person (<u>PNr</u>, Name)

  Mitarbeiter (<u>PNr</u>, …)

  Kunde (<u>PNr</u>, …)



DB-Schema „Musterunternehmen“



R.Abteilung (AbtNr, Name, SVN)

R.Mitarbeiter (SVN, GebDat, Gehalt, Adresse, VName, FName, AbtNr, seit)

R.Projekt (Nummer, Name)

R.Angehöriger (Name, GebDat, Verwandtgrad, SVN)

R.Gebäude (Gebäude, AbtNr)

R.arbeitet_an (SVN, Nummer, Stunden)

——— Primärschlüssel

- - - - - - - - - - Fremdschlüssel

———→ referentielle Integritäten

DB-Schema „Musterunternehmen“

R.Abteilung (AbtNr, Name, SVN)

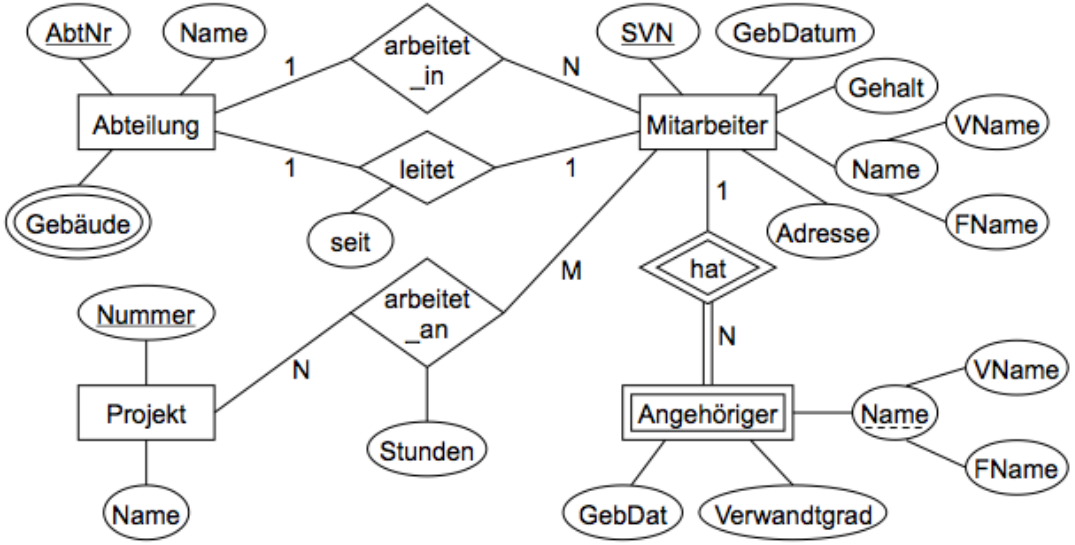
R.Mitarbeiter (SVN, GebDat, Gehalt, Adresse, VName, FName, AbtNr, seit)

R.Projekt (Nummer, Name)

R.Angehöriger (Name, GebDat, Verwandtgrad, SVN)

R.Gebäude (Gebäude, AbtNr)

R.arbeitet_an (SVN, Nummer, Stunden)

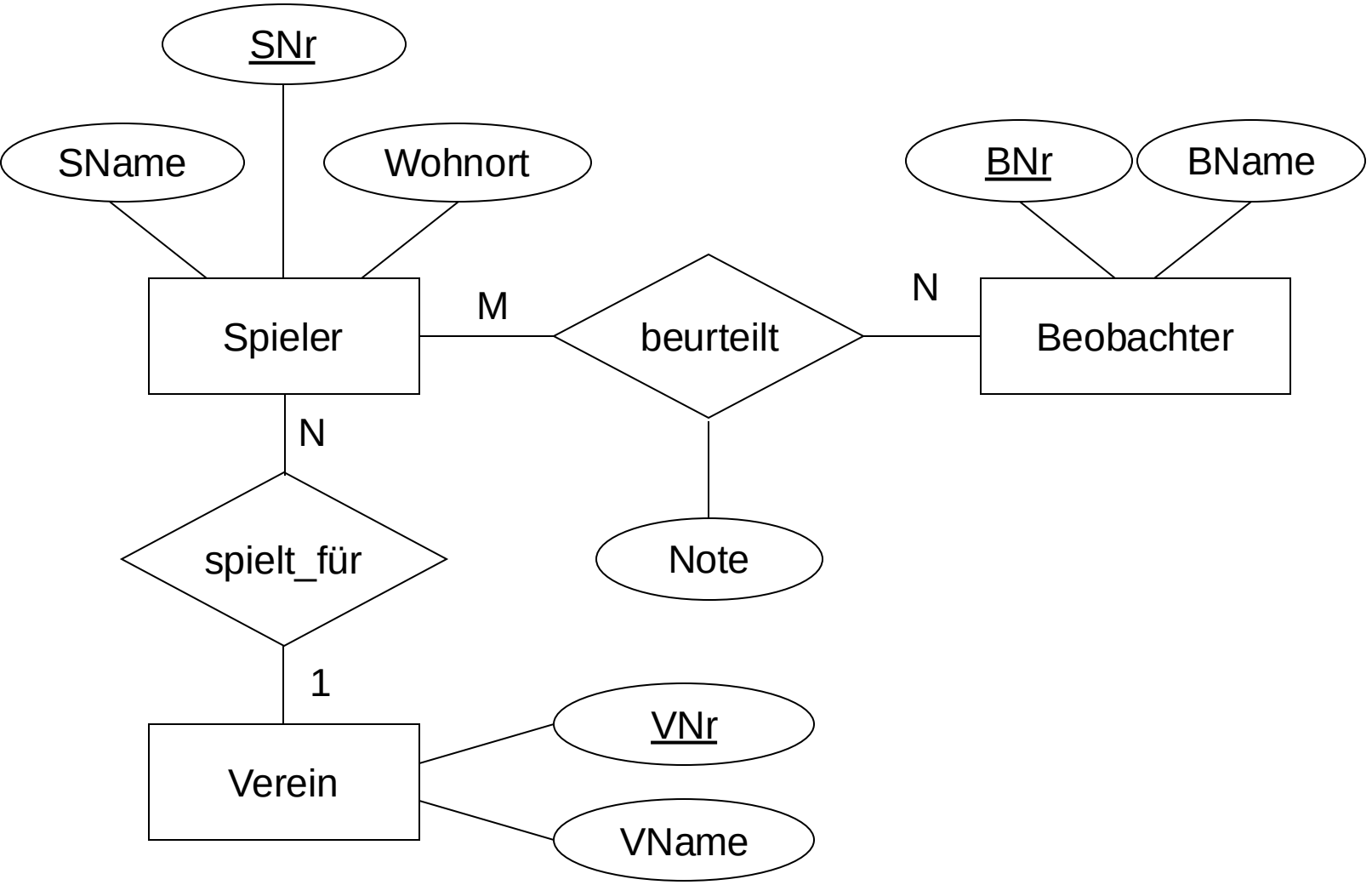


Primärschlüssel

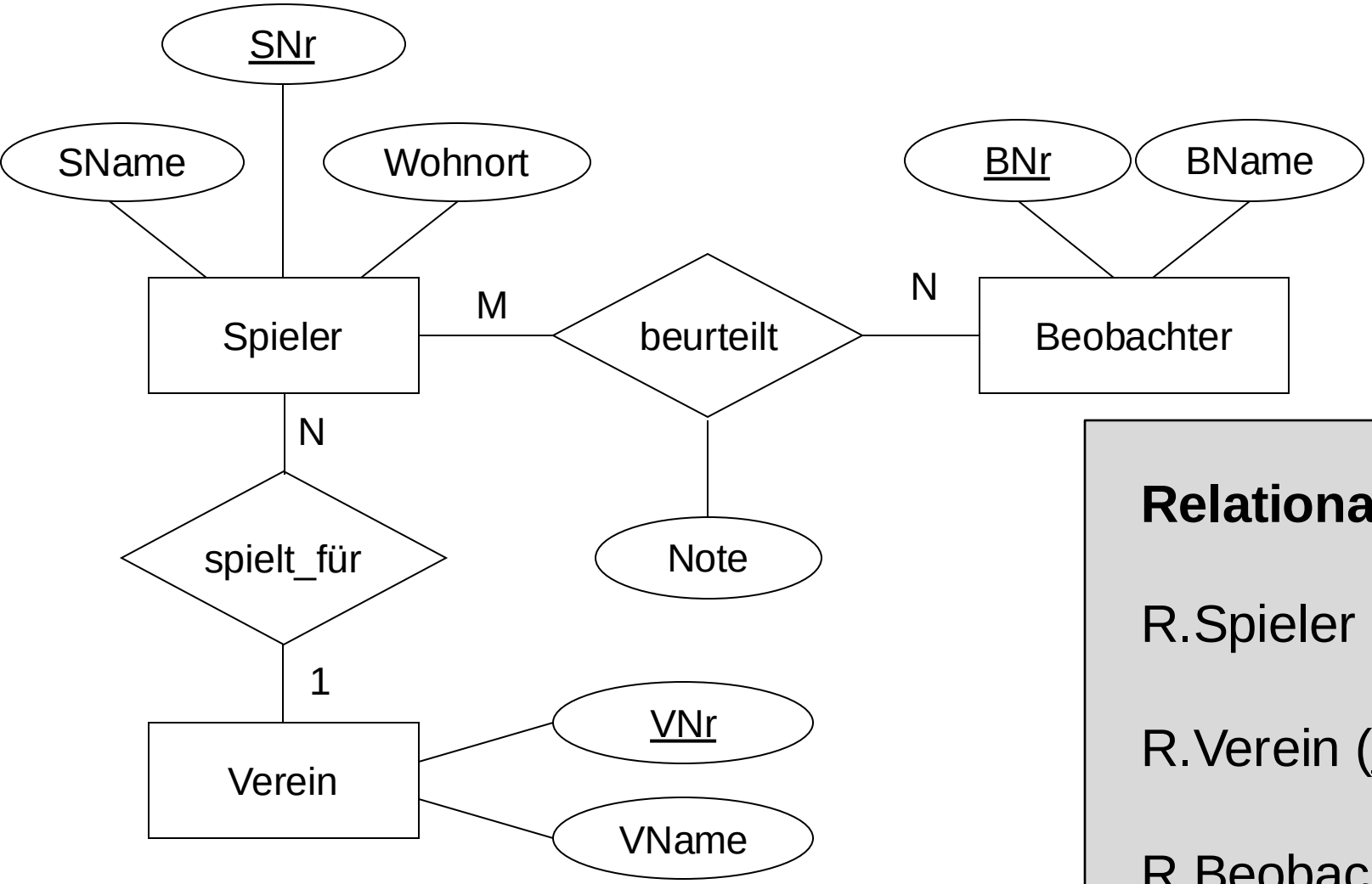
Fremdschlüssel

referentielle Integritäten

Übungsaufgabe: ERM „Fußball“



Übungsaufgabe: ERM „Fußball“



Primärschlüssel

Fremdschlüssel

referentielle Integritäten

Relationales DB-Schema „Fußball“

R.Spieler (SNr, SName, Wohnort, VNr)

R.Verein (VNr, VName)

R.Beobachter (BNr, BName)

R.Beurteilt (SNr, BNr, Note)

Gliederung

1. Entwurf des Relationalen Datenbankschema

    Mapping von Entitätstypen

    Nicht-elementare Attribute

    Relationen

2. Abfragen von Daten aus relationalen Datenbanken

    SQL

SQL – Structured Query Language

- „Strukturierte Abfragesprache“ für RDBMS
- Ende der 70er Jahre von IBM entwickelt („SEQUEL“)
- SQL-Standards: ANSI (American National Standard Institute) und ISO (International Standards Organization)
- Kommandos
  - zur Datendefinition (Data Definition Language, DDL) und
  - zur Daten-Manipulation (Data Manipulation Language, DML)
  - zur Datenabfrage (Data Query Language, DQL, Teil der DML) und
  - sowie Kommandos zum Vergeben von Zugriffsberechtigungen (Data Control Language, DCL)

SQL – Structured Query Language

- Funktionalität
  - Datenbankschemata anlegen
  - Datenbankinhalte manipulieren
  - Auswertungen von Datenbeständen durchführen („Reports“)
- RDBMS verfügen meist über mehrere Möglichkeiten/ Schnittstellen, SQL-Befehle zu verarbeiten:
  - Interaktiv in Kommandozeilenwerkzeugen (z.B. mysql bei MySQL)
  - Schnittstellen zu Programmiersprachen (ODBC für C/C++, JDBC zu Java)
  - Einbettung in Programmiersprachen („Embedded SQL“ in C/C++/Java)

SQL – Structured Query Language

Überblick über wichtige DML-/DDL-Anweisungen:

- Erzeugen von Tabellen durch CREATE TABLE

```
create table Mitarbeiter (name varchar(100), abtnr int);
```

- Einfügen in Tabellen durch INSERT

```
insert into Mitarbeiter values(‚Meier‘,5);
```

- Ändern von Tupeln durch UPDATE

```
update Mitarbeiter set abtnr=6 where name=‚Meier‘;
```

- Löschen von Tupeln durch DELETE

```
delete from Mitarbeiter where name=‚Meier‘;
```

Überblick über wichtige SQL-Anweisungen

Abfragen von Daten aus Tabellen durch die Anweisung SELECT

- Grundstruktur

```
SELECT <Attribut(e) A1,…, An>
FROM <Relation(en) R1, …, Rn>
WHERE <Bedingung (R1, …, Rn)>
```

„Selektiere aus dem Kreuzprodukt der Relationen R_1, ..., R_n alle Tupel, welche die Bedingung (R_1, ..., R_n) erfüllen und projiziere die so entstehende Relation auf die Attribute A_1, ..., A_n “

Beispiel

Abfragen von Daten aus Tabellen durch die Anweisung SELECT

- Grundstruktur

```
SELECT <Attribut(e) A1,…, An>
FROM <Relation(en) R1, …, Rn>
WHERE <Bedingung (R1, …, Rn)>
```

```
SELECT SNr,Vorname,Name
FROM Schüler
WHERE Name = 'Müller'
```

Schüler

SNr	Vorname	Name
4711	Paul	Müller
0815	Erich	Schmidt
7472	Sven	Lehmann
1234	Olaf	Müller
2313	Jürgen	Paulsen

→

SNr	Vorname	Name
4711	Paul	Müller
1234	Olaf	Müller

„DISTINCT“-Elimination von Duplikaten I

- Im Relationalen Modell ist das Ergebnis einer Operation wieder eine Menge
- Bei SQL können in der Ergebnistabelle gleiche Tupel auftreten, so dass die Mengeneigenschaft nicht mehr gegeben ist

```
SELECT Name FROM Schüler;
```

Schüler

SNr	Vorname	Name
4711	Paul	Müller
0815	Erich	Schmidt
7472	Sven	Lehmann
1234	Olaf	Müller
2313	Jürgen	Paulsen

→

Name
Müller
Schmidt
Lehmann
Müller
Paulsen

„DISTINCT“-Elimination von Duplikaten II

- Zur Elimination doppelter Tupel in der Ergebnistabelle kann das optionale Schlüsselwort **DISTINCT** im SELECT-Befehl verwendet werden
- Beispiel:

```
SELECT DISTINCT name FROM Schüler
```

Schüler

SNr	Vorname	Name
4711	Paul	Müller
0815	Erich	Schmidt
7472	Sven	Lehmann
1234	Olaf	Müller
2313	Jürgen	Paulsen

→

Name
Müller
Schmidt
Lehmann
Paulsen

„ORDER BY“ – Sortierte Ausgabe von Tupeln

- Die „ORDER BY“-Klausel **sortiert** die Ergebnistabelle nach Spalten
- Jede Sortierung kann aufsteigend (ASC) oder absteigend (DESC) erfolgen
- Wenn mehrere Sortierspalten angegeben werden, erfolgt eine geschachtelte Sortierung:
  - zunächst wird nach dem ersten Attribut sortiert
  - bei Gleichheit nach dem zweiten Kriterium etc.

Schüler

SNr	Vorname	Name
4711	Paul	Müller
0815	Erich	Schmidt
7472	Sven	Lehmann
1234	Olaf	Müller
2313	Jürgen	Paulsen

→

NO ORDER

Name
Müller
Schmidt
Lehmann
Paulsen

ASC

Name
Lehmann
Müller
Paulsen
Schmidt

DESC

Name
Schmidt
Paulsen
Müller
Lehmann

Beispiele

- Ermittle alle Titelnummern, den zugehörigen Typ und die jeweilige Verlagsnummer und sortiere die Ergebnistabelle (aufsteigend) nach der Verlagsnummer!

```
SELECT title_id, type, pub_id FROM titles
ORDER BY pub_id
```

- Ermittle alle Titelnummern, den zugehörigen Typ und die jeweiligen Verlagsnummern und sortiere die Ergebnistabelle in absteigender Reihenfolge nach der Verlagsnummer, dann nach dem Typ (aufsteigend) und schließlich nach der Titelnummer (aufsteigend)!

```
SELECT title_id, type, pub_id FROM titles
ORDER BY pub_id DESC, type ASC, title_id ASC
```

„GROUP BY“ – Gruppierung von Tupeln

Mit „GROUP BY“ wird die Ergebnistabelle **in Gruppen eingeteilt**. In der Regel wird nach den Werten einer Spalte gruppiert und die Tupel der Gruppen mit Funktionen weiterverarbeitet.

Gruppenwertfunktionen:

- **SUM**([ALL|DISTINCT] value) → Summe der Werte für eine Gruppe von Zeilen
- **AVG**([ALL|DISTINCT] value) → Durchschnittswert für eine Gruppe von Zeilen
- **COUNT**([ALL|DISTINCT] value) → Anzahl der Zeilen mit value-Ausdruck nicht NULL
- **MAX**([ALL|DISTINCT] value) → Höchster/Größter Wert für eine Zeilengruppe
- **MIN** ([ALL|DISTINCT] value) → Kleinster Wert für eine Zeilengruppe

Beispiel zu „GROUP BY“

Ermittle die Gesamtverkaufszahlen der Verlage!

Pub_ID	Total_Sales
43	5
43	10
43	20
48	40
48	10
48	10

```
SELECT pub_id, SUM(total_sales)
FROM titles GROUP BY pub_id
```

Pub_ID	Total_Sales
43	35
48	100

„FROM“ – Abfrage von Tabellen

- Der „FROM“-Teil wird in jedem SQL-Befehl benötigt, in dem Bezug auf Daten von mindestens einer Tabelle genommen wird.
  - Sind mehrere Tabellen anzugeben, sind diese durch Kommata zu trennen
  - Im „FROM“-Teil dürfen vollständige Bezeichnungen angegeben werden (Datenbankname.Besitzer.Tabellenname.Spalte).
  - Tabellennamen können mit einem „Alias“ belegt werden

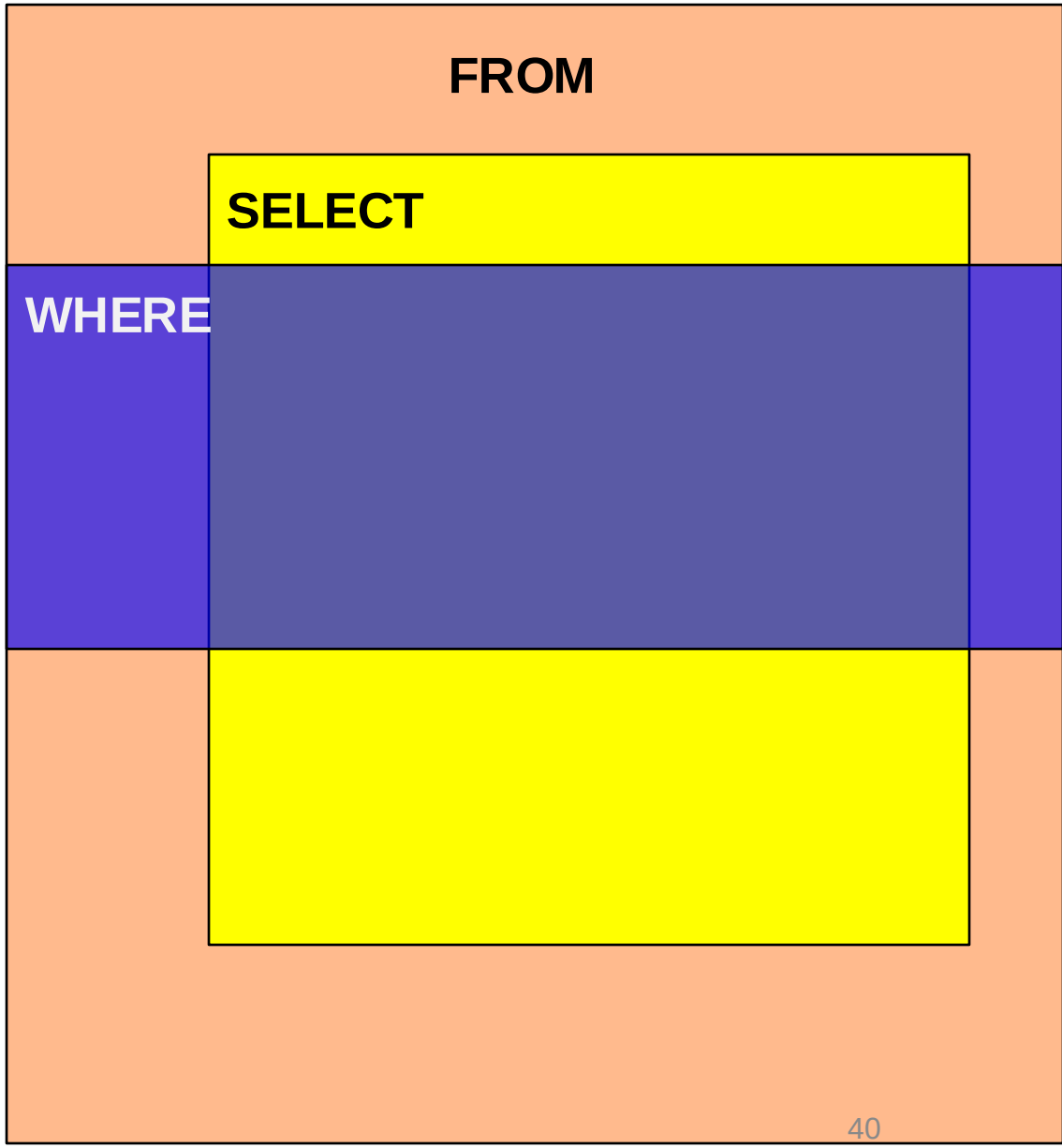
Beispiel-Abfrage

```
SELECT p.pub_id, p.pub_name FROM publishers p
```

Bedingte Auswahl aus Tabellen – Die „WHERE“-Klausel

- Bis jetzt nur Möglichkeit der Auswahl von Attributen (Projektion) und Verarbeitung aller Zeilen
- Typischerweise sind nicht alle Zeilen einer Tabelle für Auswertungen notwendig/sinnvoll
- Erweiterung um Auswahl von Zeilen (Selektion) nach gegebenen Bedingungen

WHERE-Klausel
zur Formulierung von Bedingungen



„WHERE“-Klausel zur Auswahl von Tupeln

- Definition von Bedingungen in der „WHERE“-Klausel eines SELECT, UPDATE oder DELETE-Befehls
- Syntax (Auszug):
  - `WHERE [NOT] expression comparison_operator expression`
  - `WHERE [NOT] expression [NOT] LIKE “matching_string”`
  - `WHERE [NOT] expression is [NOT] NULL`
  - `WHERE [NOT] expression [NOT] IN ({value_list | subquery})`
  - `WHERE [NOT] EXISTS (subquery)`
  - `WHERE [NOT] column_name join_operator column_name`

„WHERE“-Klausel zur Auswahl von Tupeln

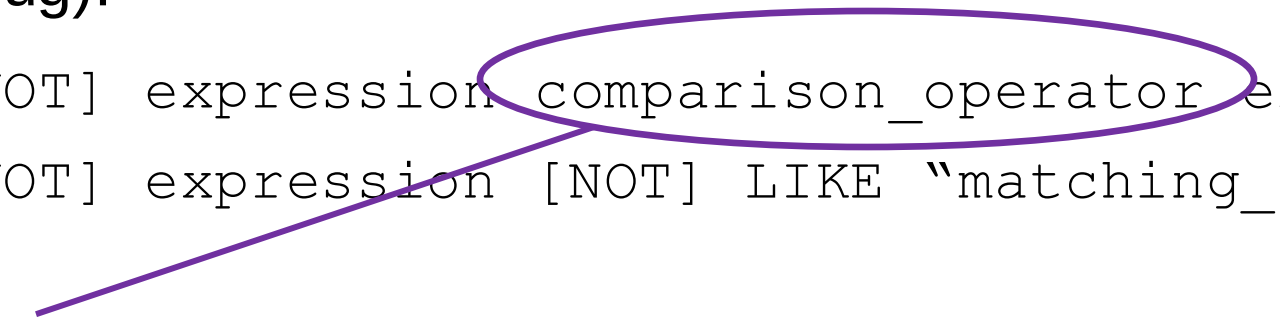
- Definition von Bedingungen in der „WHERE“-Klausel eines SELECT, UPDATE oder DELETE-Befehls
- Syntax (Auszug):
  - `WHERE [NOT] expression comparison_operator expression`
  - `WHERE [NOT] expression [NOT] LIKE “matching_string”`

expression

Spaltenname, eine Konstante, eine Funktion oder eine Kombination aus diesen, die über arithmetische oder bitweise Operatoren oder eine Subquery verknüpft werden

„WHERE“-Klausel zur Auswahl von Tupeln

- Definition von Bedingungen in der „WHERE“-Klausel eines SELECT, UPDATE oder DELETE-Befehls
- Syntax (Auszug):
  - `WHERE [NOT] expression comparison_operator expression`
  - `WHERE [NOT] expression [NOT] LIKE “matching_string”`



comparison_operators (Vergleichsoperatoren) (=, >, <, >=, <=, !=, <>, !>, !<)

Bei einem Vergleich von Zeichenketten (char bzw. varchar) bedeutet ‘<‘ näher am Anfang und ‘>‘ näher am Ende des Alphabets.
Bei einem Vergleich von Datumsangaben (datetime) bedeutet ‘<‘ früher und ‘>’ später

„WHERE“-Klausel zur Auswahl von Tupeln

- Definition von Bedingungen in der „WHERE“-Klausel eines SELECT, UPDATE oder DELETE-Befehls
- Syntax (Auszug):
  - `WHERE [NOT] expression comparison_operator expression`
  - `WHERE [NOT] expression [NOT] LIKE “matching_string”`

Wildcards „Jokerzeichen“ bei Stringvergleichen (Bei [NOT] LIKE)

% (Prozentzeichen) bedeutet eine Zeichenkette mit keinem oder mehr Zeichen
_ (Unterstrich) bedeutet ein einzelnes Zeichen
[ ] bedeutet ein einzelnes Zeichen innerhalb des spezifizierten Bereichs ([a-f]) oder der Menge ([abcdef])
[^] bedeutet ein einzelnes Zeichen nicht innerhalb des spezifizierten Bereichs (^[a-f]) oder der Menge (^[abcdef])

Beispiele zur Selektion

Auswählen aller Autoren, die im Staat Californien leben

```
SELECT au_fname, au_lname, city
FROM authors
WHERE state ='CA'
```

Auswählen aller Autoren, deren Nachname *nicht* mit ‚A' anfängt

```
SELECT au_fname, au_lname, city
FROM authors
WHERE NOT (au_fname ='A%')
```

Beispiele zur Selektion

Auswählen aller Autoren, die in Palo Alto oder Oakland leben:

```
SELECT au_fname, au_lname, city
FROM authors
WHERE city='Palo Alto' OR city='Oakland'
```

Auswählen aller Bücher, die im Titel das Wort „Computers“ verwenden:

```
SELECT title, price
FROM titles
WHERE title LIKE '%Computers%'
```

Gruppenfunktionen – Verschachtelte Abfragen

Ermittle den Buchtitel mit dem höchsten und den mit dem niedrigsten Preis!

```
SELECT title Buchtitel, price Preis
FROM titles
WHERE price = (SELECT max(price) FROM titles)
OR price = (SELECT min(price) FROM titles)
```

Spielplatz für SQL-Befehle:

https://www.w3schools.com/sql/trysql.asp?filename=trysql_asc

Verknüpfung von Tabellen durch Joins

Bislang waren nur Abfragen **einer** Tabelle möglich. Aber ggf. sind Abfrage über **mehrere** Tabellen notwendig.

Prinzipielle Durchführung eines Joins

Annahme: Verknüpfung zweier Tabellen

1. Erstellung des kartesischen Produkts der beteiligten Tabellen
2. Verarbeitung/Prüfung der Join-Bedingung (WHERE-Klausel)

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2137	Kant	C4	7
...	...	...	...

hören	
MatrNr	VorlNr
26120	5001
29555	5001
...	...

Professoren×hören					
Professoren				hören	
PersNr	Name	Rang	Raum	MatrNr	VorlNr
2125	Sokrates	C4	226	26120	5001
...	...	...	...	...	...
2125	Sokrates	C4	226	29555	5001
...	...	...	...	...	...
2137	Kant	C4	7	29555	5001

Verknüpfung von Tabellen durch Joins

Bislang waren nur Abfragen **einer** Tabelle möglich. Aber ggf. sind Abfrage über **mehrere** Tabellen notwendig.

Prinzipielle Durchführung eines Joins

Annahme: Verknüpfung zweier Tabellen

1. Erstellung des kartesischen Produkts der beteiligten Tabellen
2. Verarbeitung/Prüfung der Join-Bedingung (WHERE-Klausel)

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2137	Kant	C4	7
...	...	...	...

hören	
MatrNr	VorlNr
26120	5001
29555	5001
...	...

Professoren × hören					
Professoren				hören	
PersNr	Name	Rang	Raum	MatrNr	VorlNr
2125	Sokrates	C4	226	26120	5001
...	...	...	...	...	...
2125	Sokrates	C4	226	29555	5001
...	...	...	...	...	...
2137	Kant	C4	7	29555	5001

Tabelle A hat 7 Einträge, Tabelle B hat 8 Einträge.
Wieviel Einträge hat dann das Kreuzprodukt A x B ?

Verknüpfung von Tabellen durch Joins

Prinzipielle Durchführung eines Joins

Kosten der Erstellung des kartesischen Produkts zweier Tabellen mit n bzw. m Einträgen = **n * m**

Deshalb Versuch der Optimierung durch WHERE-Klausel Auswertung und Ausnutzung von Tabellen-Indexen

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

Verknüpfung von Tabellen durch Joins

Join-Klassen

Equi-Join („SQL-Standardfall“)

- Join-Operator ist „=“
  In der Ergebnisrelation erscheinen beide am Join direkt beteiligten Spalten

Equi-Join

Resultat					
A	B	C	C	D	E
a_1	b_1	c_1	c_1	d_1	c_1

Natural Join

- Join-Operator ist „=“
  In der Ergebnisrelation erscheint eine der am Join direkt beteiligten Spalten

Natural-Join

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	c_1

Join mit Zusatzbedingung

- Reihenfolge von Join- und Zusatzbedingung nicht wichtig;
  Beliebige Anzahl von Zusatzbedingungen

Beispiele: Joins I

- Liste die einzelnen Buchtitel und die dazugehörigen Verleger auf.

```
SELECT title, pub_name
FROM titles, publishers
WHERE titles.pub_id = publishers.pub_id
```

- Gib die Namen der Autoren und die von ihnen verfassten Titel aus.

```
SELECT au_lname, au_fname, title
FROM authors, titles, titleauthor
WHERE titles.title_id = titleauthor.title_id
AND authors.au_id = titleauthor.au_id
```

Beispiele: Joins II

- Equi-Join

```
SELECT * FROM authors, publishers
WHERE authors.city=publishers.city
```

- Natural Join

```
SELECT publishers.pub_id, publishers.pub_name,
publishers.state, authors.*
FROM authors, publishers
WHERE authors.city=publishers.city
```

Beispiele: Joins II

- Join mit Zusatzbedingung: Selektiere Titel und Verlag aller Bücher, für die ein Vorschuss von mehr als $7500 gewährt wurde!

```
SELECT title, pub_name, advance FROM titles,
publishers WHERE titles.pub_id = publishers.pub_id
AND advance > 7500
```

Verknüpfung von Tabellen durch Joins

- Joins innerhalb einer Tabelle („Self-Join“)
- Joins, die nicht auf Gleichheit basieren
  - Das Keyword DISTINCT dient zur Elimination doppelter Tupel
  - Allgemein ist ein „Not-Equal-Join“ nützlich zur Restriktion der Ergebnisrelation von Self-Joins
- Joins über mehr als zwei Relationen
  - Die Auswahl der Attribute muss nicht aus jeder der angesprochenen Tabellen eine Spalte enthalten
  - Die Auswahl der Attribute muss nicht die Join-Attribute enthalten

Beispiel: Self Join

Self-Join: Suche alle Autoren aus Oakland und Californien, die im gleichen Postleitzahlenbereich wohnen!

```
SELECT au1.au_fname, au1.au_lname, au2.au_fname, u2.au_lname
FROM authors au1, authors au2
WHERE au1.city="Oakland" AND au2.city="Oakland"
AND au1.state="CA" AND au2.state="CA"
AND au1.postalcode = au2.postalcode;
```

Vollständige Syntax des SELECT-Befehls

```
SELECT [DISTINCT | ALL]
{ * |       { [user.] {table | view | snapshot} .* | expr [ [AS] column_alias] }
  [,        { [user.] {table | view | snapshot} .* | expr [ [AS] column_alias] } ] …
FROM
    {[user.] {table [PARTITION (partition_name) | @dblink] | [view | snapshot] [@dblink]} [table_alias] | [THE]
    (subquery) [table_alias] | TABLE (nested_table_column) [table_alias]}
   [,{[user.] {table [PARTITION (partition_name) | @dblink] | [view | snapshot] [@dblink]} [table_alias] | [THE]
    (subquery) [table_alias] | TABLE (nested_table_column) [table_alias]}]…
[WHERE condition]
[[START WITH Condition] CONNECT BY condition | GROUP BY expr [, expr] …
    [HAVING condition] ] … [{ UNION | UNION ALL | INTERSECT | MINUS} SELECT command]
[ORDER BY {expr | position | column_alias} [ASC | DESC]
          [,{expr | position | column_alias} [ASC | DESC] ] …
| FOR UPDATE [OF         [ [user.] {table. | view.} ] column]
                    [,[user.] {table. | view.} ] column] ...]
```

Zusammenfassung

Mittels Transformation-Regeln lässt sich ein ER-Modell in ein relationales Schema eindeutig überführen.

Umsetzung von Entitätstypen und Attributen in Tabellen und Spalten

Structured Query Language (SQL) als strukturierte Abfragesprache für RDBMS

Nächste Termine

Nächste Vorlesung:

11.12.2024