Datenorganisation und Datenmodellierung

JProf. Dr. Gunnar Stevens
Human Computer Interaction
University of Siegen
gunnar.stevens@uni-siegen.de

Agenda

- Entwurf des Relationen Datenbankschema
  - Mapping von Entitätstypen
  - Nicht-elementaren Attributen
  - Relationen

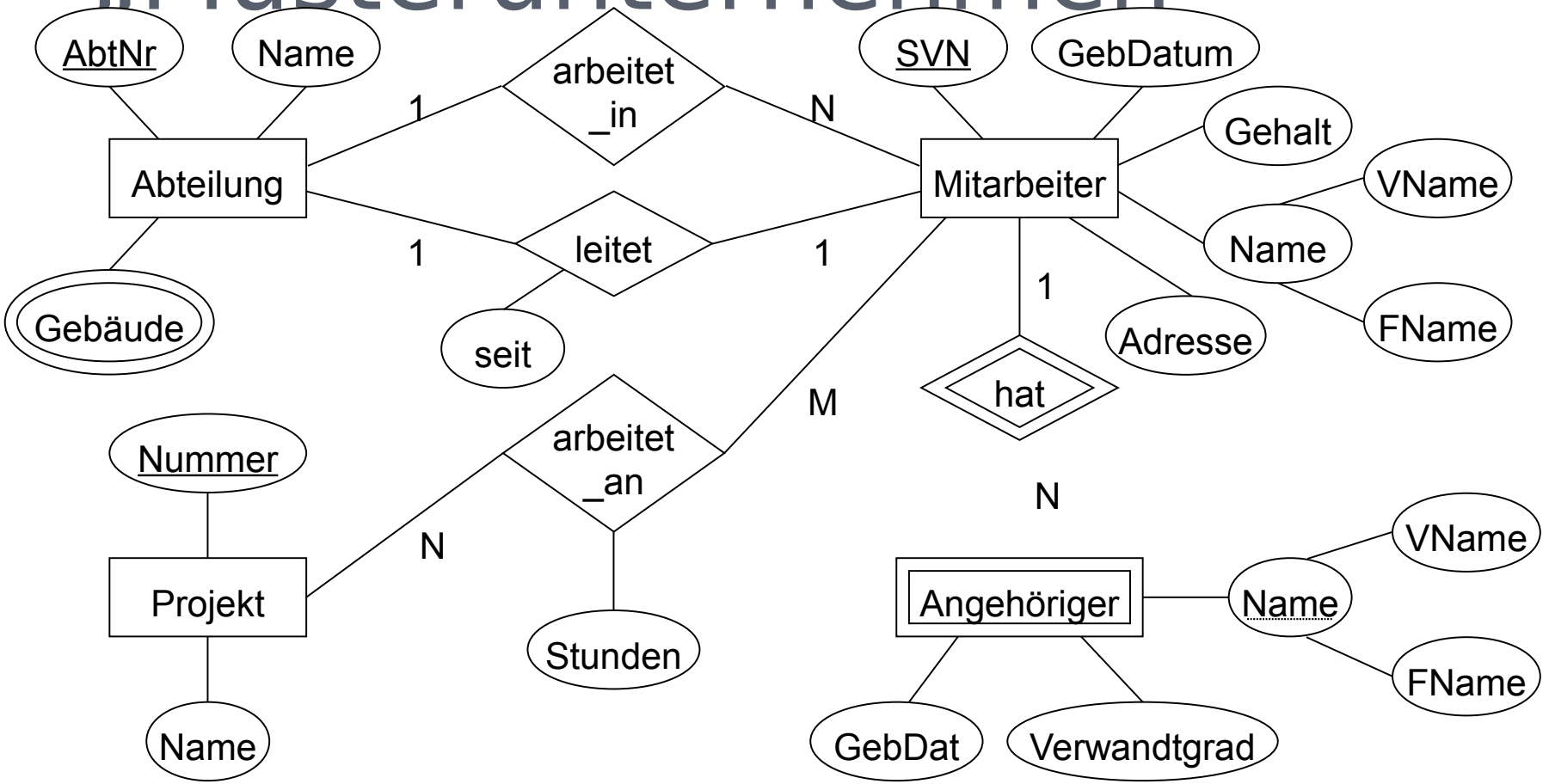
- Abfragen von Daten aus relationalen Datenbanken
  - SQL
    - SELECT [welche Attribute]
    - FROM [welchen Tabellen]
    - WHERE [welche Bedingungen müßen erfüllt sein]

Entwurf des Relationales Schema

Mapping des ER-Modell auf Relationales Schema

- Ein Ergebnis einer Analyse (eines Unternehmens) ist das Datenmodell (ggf. ER-Modell) als ein Ausgangspunkt für Entwicklung/Anpassung datenbankbasierter Anwendungen
- ER-Modell ist nicht unmittelbar für Aufbau eines Datenbankschemas geeignet
- Regeln zur Transformation eines Modells (Entititypen, Attribute, Beziehungen etc.) in ein relationales Schema (Tabellen, Schlüssel) sind notwendig

Beispiel „Musterunternehmen“



Umsetzung von Entitytypen und Attributen I

- 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 (AbtNr, Name, …
  Mitarbeiter (SVN, GebDat, Gehalt, Adresse, …
  Projekt (PNummer, Name)

Umsetzung von Entitytypen und Attributen II

- Ein schwacher Entitytyp E wird in eine Relation R umgesetzt,

A) die alle Attribute des Entitytypen 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

- **Zusammengesetzte Attribute** in Entitytyp S
  - Relation S wird um die einzelnen Bestandteile des zusammengesetzten Attributes als einzelne Spalten erweitert

- Beispiel:

  Mitarbeiter (SVN, ~~Name,~~ Vorname, Nachname, …

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Umsetzung von nicht elementaren Attributen

- **Mehrwertige Attribute** in Entitytyp S
  - 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

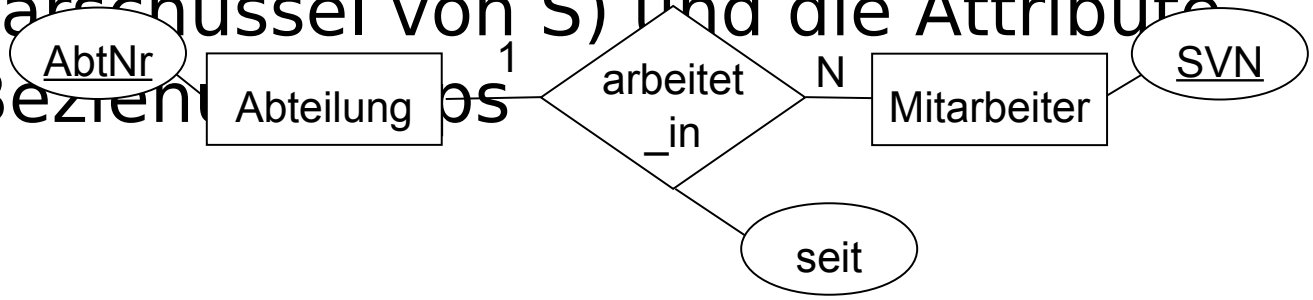
- Beispiel:

  Auto (Fahrzeug-Nr, ~~Autofarbe,~~ …)

  Autofarbe(Fahrzeug-Nr, Farbe)

Umsetzung von Beziehungstypen

- 1:N-Beziehungstyp zwischen Entity-Typen S und T
  - Erweiterung der zu T gehörenden Relation um ein Fremdschlüsselattribut (Primärschüssel von S) und die Attribute des Beziehungstyps



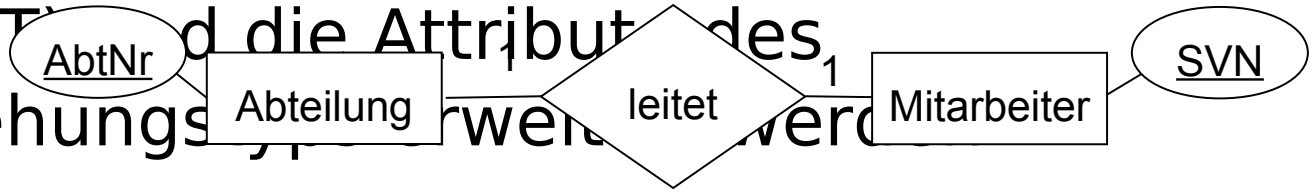
Beispiel:

R.Abteilung(AbtNr, …)

R.Mitarbeiter(AbtNr, seit, SVN, …)

Umsetzung von Beziehungstypen

- 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:

R.Abteilung(<u>AbtNr</u>, …)
R.Mitarbeiter(AbtNr, <u>SVN</u>, …)

oder

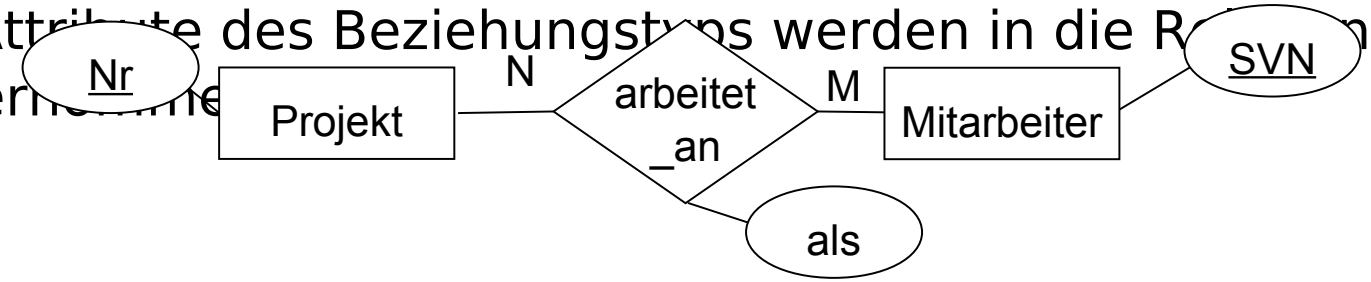
R.Abteilung(SVN, <u>AbtNr</u>, …)
R.Mitarbeiter(<u>SVN</u>, …)

Umsetzung von Beziehungstypen

- 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
  - 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

- 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:

R.Projekt(Nr, …)

R.Mitarbeiter(SVN, …)

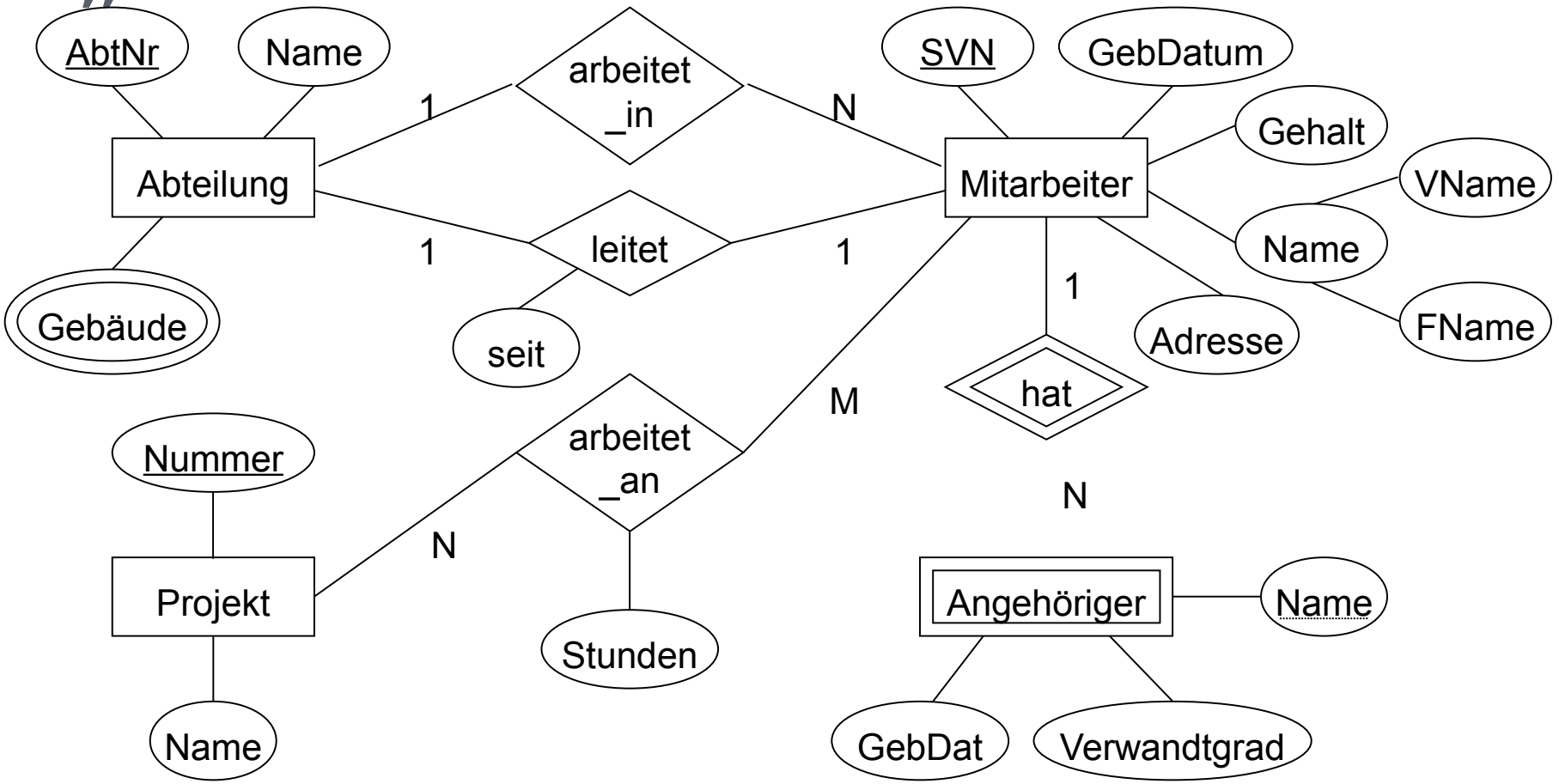
R.arbeitet_an(Nr, SVN, als)

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

- 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

Beispiel „Musterunternehmen“



Relationenschema „Musterunternehmen“

R.Abteilung (AbtNr, AbtName, SVNLeiter, seit)

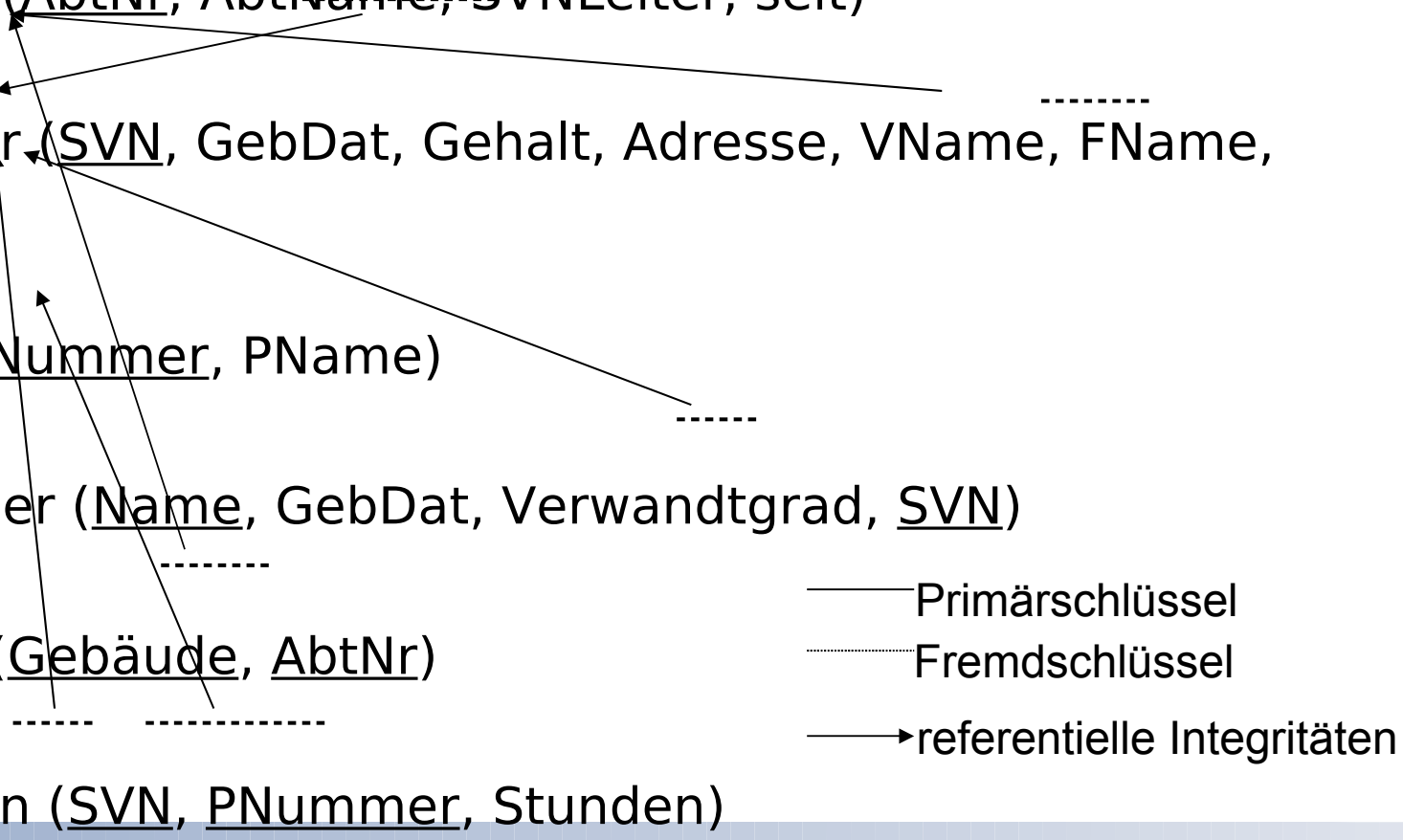
R.Mitarbeiter (SVN, GebDat, Gehalt, Adresse, VName, FName, AbtNr)

R.Projekt (PNummer, PName)

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

R.Gebäude (Gebäude, AbtNr)

R.arbeitet_an (SVN, PNummer, Stunden)



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

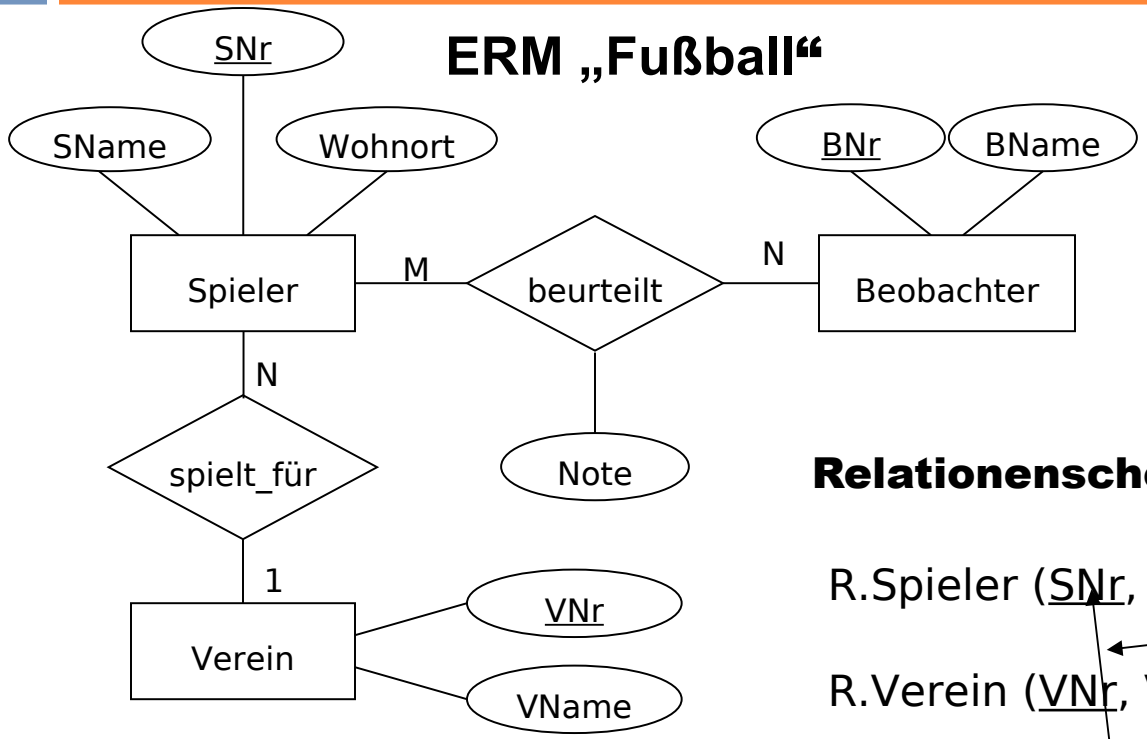
Transformation eines ER-Schemas

Spezialisierung/Generalisierung

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.

Rollennamen werden als Fremdschlüsselnamen verwendet (entweder in den für Beziehungstypen gebildeten Relationen oder für die Fremdschlüsselattribute, die direkt in für Entitytypen gebildete Relationen eingeflossen sind)

ERM und Relationenschema



Relationenschema „Fußball“ (3. NF)

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

R.Verein (VNr, VName)

R.Beobachter (BNr, BName)

R.Beurteilt (SNr, BNr, Note)

Primärschlüssel

Fremdschlüssel

referentielle Integritäten

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Datenzugriff mittels SQL

SQL – Structured Query Language

- „strukturierte Abfragesprache“ für RDBMS
- Ende der 70er Jahre von IBM entwickelt („SEQUEL“)
- Standardabfragesprache bei RDBMS
- 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

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

SQL – Structured Query Language

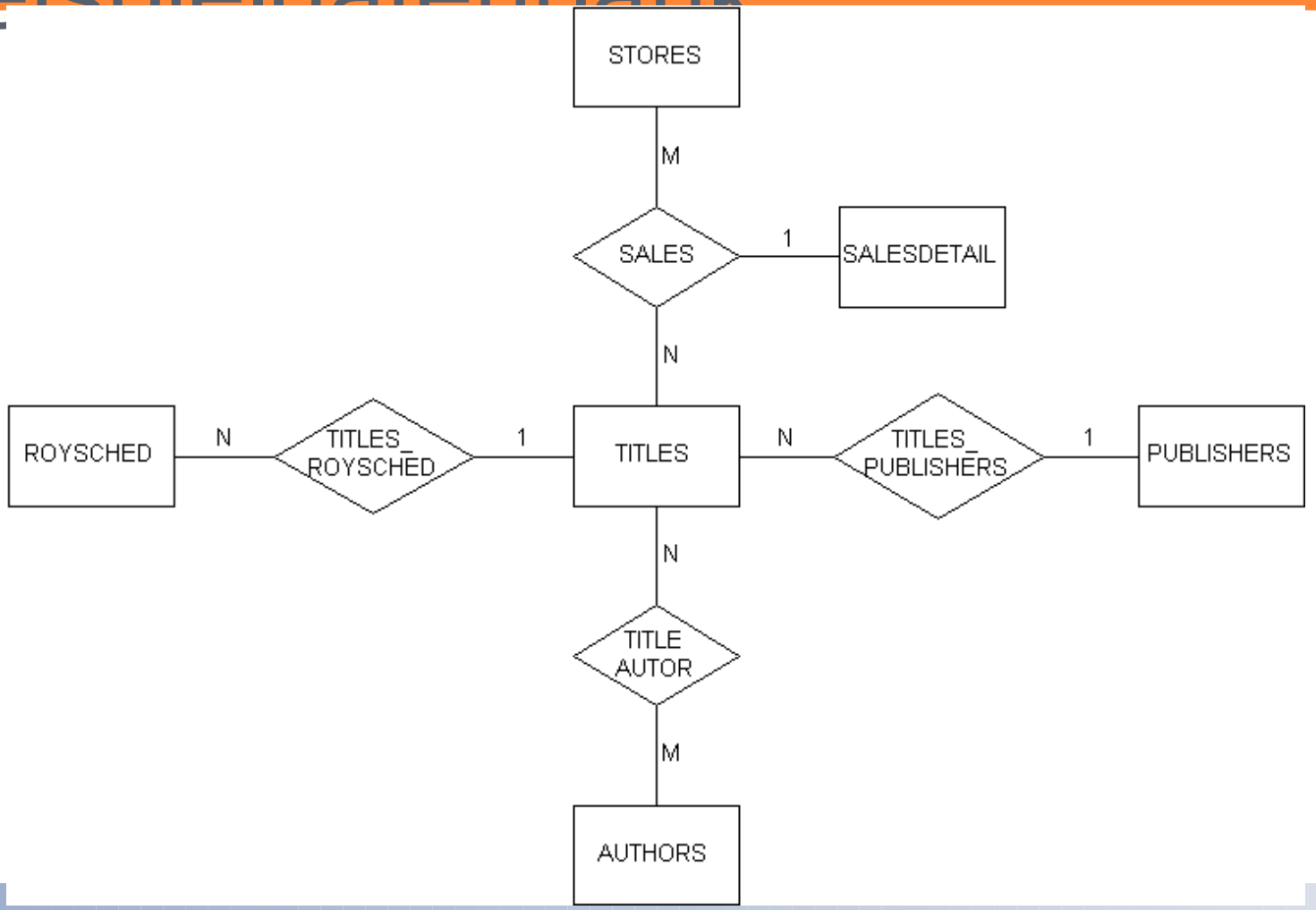
- Funktionalität:
  - Datenbankschemata anlegen
  - Datenbankinhalte manipulieren
  - Auswertungen von Datenbeständen durchführen
- Aufbereitung der Ergebnisse der Auswertungen in „Reports“
- RDBMS verfügen meist über mehrere Möglich-keiten/Schnittstellen, SQL-Befehle zu verarbeiten:
  - Interaktiv in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

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

SQL – Structured Query Language

- Überblick über wichtige DML-/DDL-Anweisungen:
  - Erzeugen von Tabellen durch CREATE TABLE
    - Beispiel: create table Person (name varchar(100), abt int);
  - Einfügen in Tabellen durch INSERT
    - Beispiel: insert into Person values(‚Meier‘,5);
  - Ändern von Tupeln durch UPDATE
    - Beispiel: update Person set abt=6 where name=‚Meier‘;
  - Löschen von Tupeln durch DELETE
    - Beispiel: delete from Person where name=‚Meier‘;
- Überblick über wichtige DQL-Anweisung
  - Abfragen von Daten aus Tabellen durch SELECT
    - **Liefert eine (unbenannte) Ergebnistabelle !**

ER-Schema der Beispieldatenbank



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Relationenschema zur Datenbank „pubs2“

R.authors (au_id, au_lname, au_fname, phone, address, city, state, country, postalcode)

R.titles (title_id, title, type, pub_id, price, advance, total_sales, notes, pubdate, contract)

R.stores (stor_id, stor_name, stor_address, city, state, country, postalcode, payterms)

R.publishers (pub_id, pub_name, city, state)

R.roysched (title_id, lorange, hirange, royalty)

R.salesdetail (stor_id, title_id, ord_num, qty, discount)

R.titleauthor (au_id, title_id, au_ord, royaltyper)

~~R.sales (stor_id, title_id)~~

Beispiel-Datenbank

authors
au_id id(11) not null,
au_lname varchar(40) not null,
au_fname varchar(20) not null,
phone char(12) not null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null

titleauthor
au_id id(11) not null,
title_id tid(6) not null,
au_ord tinyint null,
royaltyper int null

roysched
title_id tid(6) not null,
lorange int null,
hirange int null,
royalty int null

salesdetail
stor_id char(4) not null,
ord_num varchar(20) not null,
title_id tid(6) not null,
qty smallint not null,
discount float not null

titles
title_id tid(6) not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null,
price money null,
advance money null,
total_sales int null,
notes varchar(200) null,
pubdate datetime not null,
contract bit not null

publishers
pub_id char(4) not null,
pub_name varchar(40) null,
city varchar(20) null,
state char(2) null

stores
stor_id char(4) not null,
stor_name varchar(40) null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null

Grundstruktur „SELECT“-Anweisung

Syntax

SELECT <Attribut(e) A1, ..., An>

FROM <Relation(en) R1, ..., Rn>

WHERE <Bedingung (R1, ..., Rn)>

„Selektiere aus dem Kreuzprodukt der Relationen R1, ..., Rn alle Tupel, welche die Bedingung (R1, ..., Rn) erfüllen und projiziere die so entstehende Relation auf die Attribute A1, ..., An “

Beispiel

SELECT au_lname, au_fname, city

FROM authors

WHERE state= CA‘

„DISTINCT“–Elimination von Duplikaten

- Im Relationalen Modell ist das Ergebnis einer Mengenoperation wieder eine Menge
- Bei SQL können in der Ergebnistabelle gleiche Tupel auftreten, so dass die Mengeneigenschaft nicht mehr gegeben ist
- Zur Elimination doppelter Tupel in der Ergebnistabelle kann das optionale Schlüsselwort DISTINCT im SELECT-Befehl verwendet werden
- Beispiel:
  SELECT DISTINCT au_id FROM titleauthor

„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.

Beispiele zu „ORDER BY“

- 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

„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

Beispiele zu „GROUP BY“

- Ermittle die Gesamtverkaufszahlen der Verlage!

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

„FROM“ – Abfrage von Tabellen

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

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

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Bedingte Auswahl aus Tabellen

- 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)

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

„WHERE“-Klausel

- *expression* ist ein Spaltenname, eine Konstante, eine Funktion oder eine Kombination aus diesen, die über arithmetische oder bitweise Operatoren oder eine Subquery verknüpft werden
- Vergleichsoperatoren (*comparison_operators*) sind (=, >, <, >=, <=, !=, <>, !>, !<). 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
- *column_name* ist der Name der verwendeten Spalte im Vergleich

Wildcards bei Stringvergleichen

- Jokerzeichen (Wildcards) werden mit dem Keyword [NOT] LIKE verwendet, um Zeichen-ketten (Strings) zu vergleichen:
  - % (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, die *nicht* im Staat Californien leben:

```
SELECT au_fname, au_lname, city
FROM authors
WHERE NOT (state =‘CA’)
```

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%'
```

Beispiel zu Gruppenfunktionen

- 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)
```

Verknüpfung von Tabellen durch Joins

Prinzipielle Durchführung eines Joins

Annahme: Verknüpfung zweier Tabellen

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

- ☞ 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

Beispiel: Kreuzprodukt

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

x

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

Beispiel: natural join

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

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Verknüpfung von Tabellen durch Joins

Join-Klassen

- Equi-Join
  - Join-Operator ist „=“
  - In der Ergebnisrelation erscheinen *beide* am Join direkt beteiligten Spalten
- Natural Join
  - Join-Operator ist „=“
  - In der Ergebnisrelation erscheint *eine* der am Join direkt beteiligten Spalten
- Join mit Zusatzbedingung
  - Reihenfolge von Join- und Zusatzbedingung nicht wichtig
  - Beliebige Anzahl von Zusatzbedingungen

Resultat					
A	B	C	C	D	E
a_1	b_1	c_1	c_1	d_1	e_1

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

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

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

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
```

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
```

- 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
```

Beispiele: Joins III

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

```
SELECT au1.au_fname, au1.au_lname, au2.au_fname,
au2.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;
```

Beispiele: Joins IV

- Joins über mehr als zwei Relationen

Suche alle Buchtitel vom Typ „traditionelles Kochen“ und liste Vor- und Nachnamen der zugehörigen Buchautoren sowie den Titel auf!

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

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] ...]