

4.4 SQL - Programmiersprachen-Anbindung

- 4.1 SQL – DDL und DML
- 4.2 SQL-Anfragen 1
- 4.3 SQL-Anfragen 2
- 4.4 Programmiersprachen-Anbindung
 - JDBC
 - Embedded SQL

SQL und Programmiersprachen

- SQL ist Datenbanksprache, keine Programmiersprache
 - DML nicht Turing-vollständig
- Möglichkeiten der Verknüpfung von SQL mit Programmiermechanismen
 - Erweiterung von SQL um Konstrukte von Programmiersprachen s. PL/SQL in Kap. 4.2
 - Integration in eine bestehende Programmiersprache
- Impedance mismatch (semantische Lücke):
Nicht-Zusammenpassen von Programmiersprachen und Datenbankkonzepten
 - Keine Mengenoperatoren in Programmiersprachen
 - Keine komplexeren Typkonstrukte (Records, Listen) in DB-Modellen
 - Keine NULL-Werte in Datentypen von Programmiersprachen

Formen der Einbindung von SQL

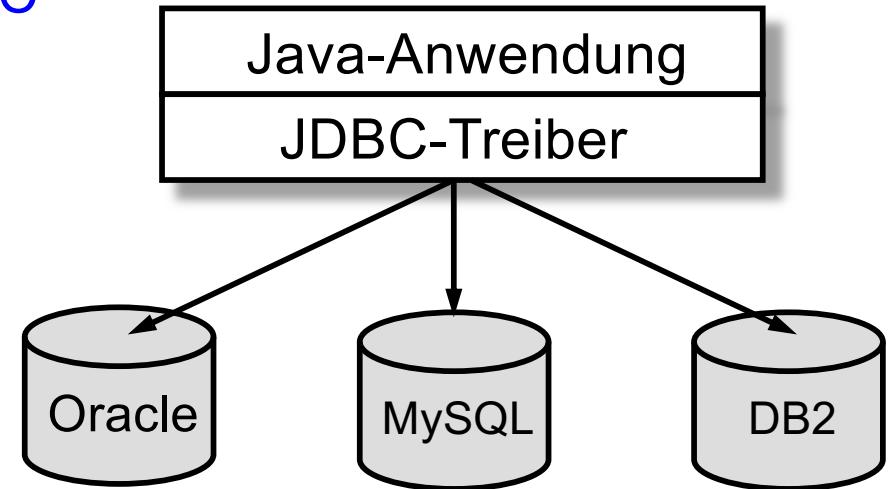
- Statische Einbindung
 - Anfragen sind zur Übersetzungszeit der Programme bekannt
 - Prüfung und Optimierung zur Übersetzungszeit
 - Häufig Verwendung eines Precompilers
- Dynamische Einbindung
 - Anfragen (zumindest einige) werden erst zur Laufzeit von Programmen erstellt
 - Übergabe der Anfrage als textueller Wert einer Variablen
 - Größere Flexibilität
 - Prüfung und Optimierung erst zur Laufzeit

Beispiele für Einbindung von SQL

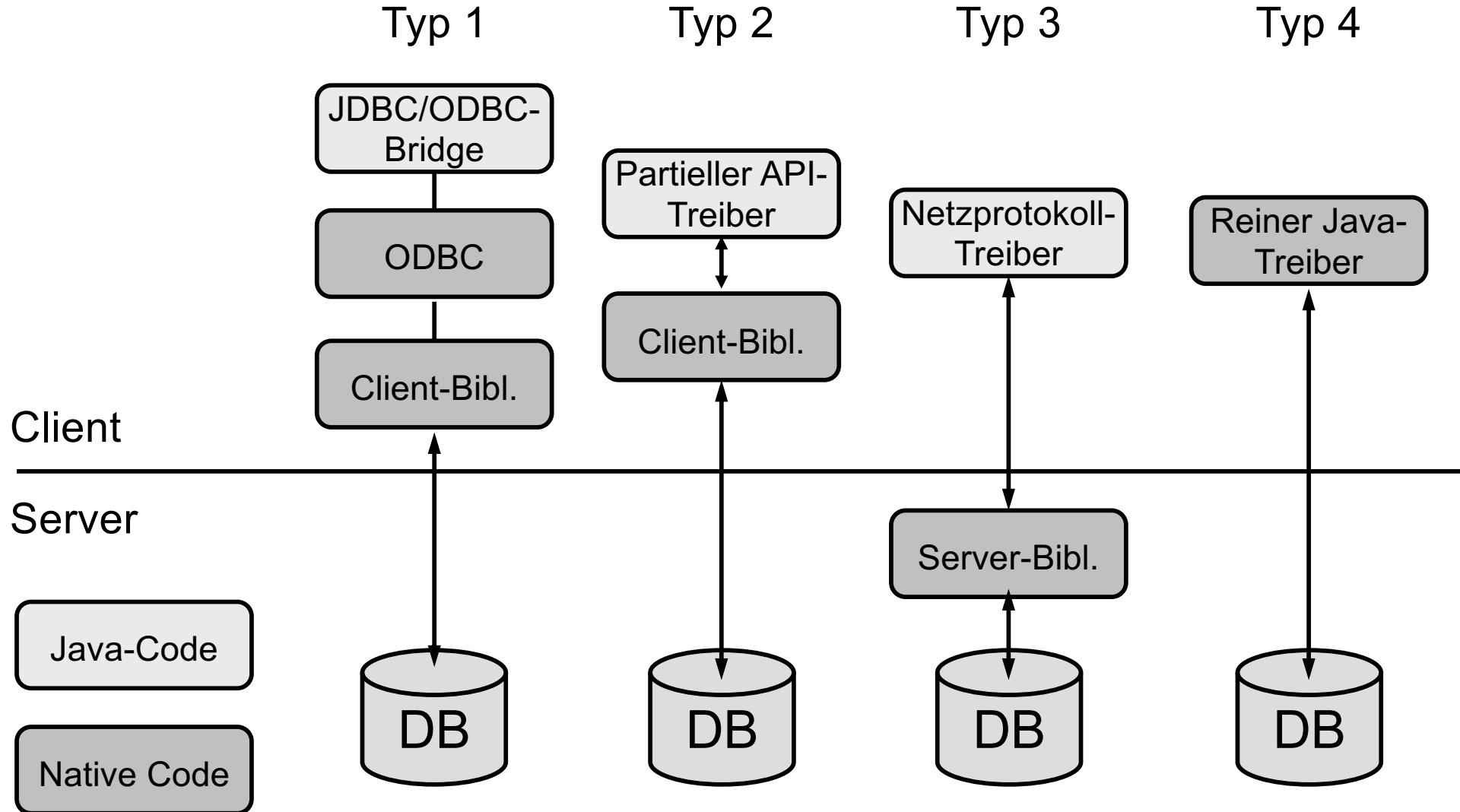
- ODBC
 - Standardisierte Schnittstelle namens Open Database Connectivity
- Embedded SQL (ESQL)
 - Statische Einbettung von SQL mit Hilfe eines Precompilers
 - Dynamische Einbettung für komplexere Suchanfragen
 - SQL92-Standard
- JDBC
 - Dynamische Einbettung von SQL in Java
- SQLJ
 - Statische Einbettung von SQL in Java
- Zahlreiche weitere Einbindungen
 - LINQ (Language Integrated Query): in .NET-Sprachen wie Visual Basic oder C#

JDBC als Standard für DB-Schnittstellen

- JDBC (Java Database Connectivity)
 - Objektorientiertes DB-Interface für Java, entwickelt von SUN
 - Ziel: standardisierter Zugriff auf beliebiges DBMS
 - Bei DBMS-Wechsel muss nur anderer Treiber eingebunden werden
 - Unterstützung von allen bedeutenden DBMS-Herstellern
- Übersichtlicher und einfacher benutzbar als ODBC
- Literatur
 - R.M. Mennor: Expert Oracle JDBC Programming, Apress, 2005 als E-Book in HTWG-Bibliothek



JDBC Treibertypen



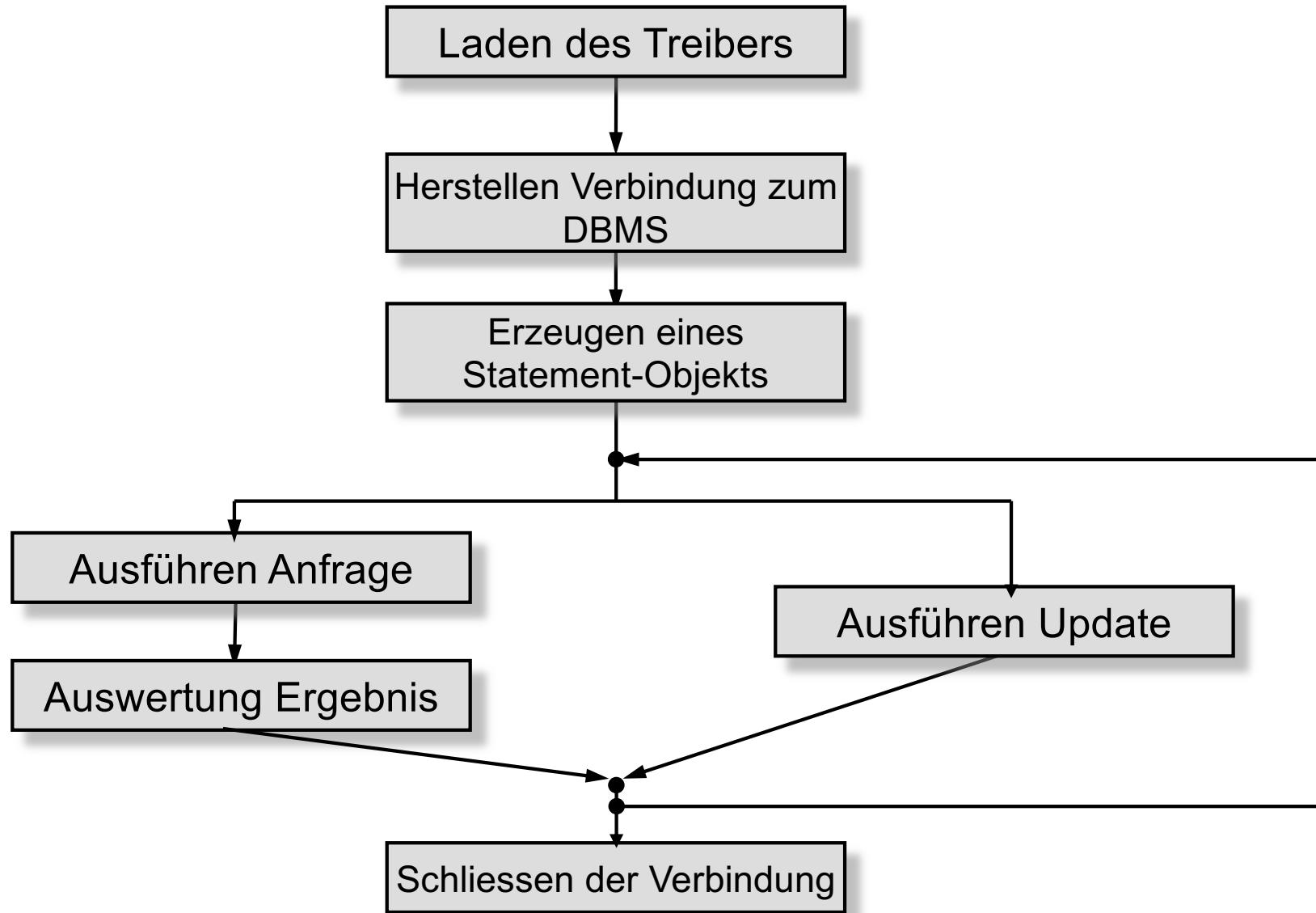
JDBC Treiber Typen

- Typ 1
 - Mapping auf anderes API, wie z.B. ODBC
 - Einschränkungen, z.B. in Portabilität
- Typ 2
 - Implementierung teilweise in Java, teilweise in native code
 - „Thick drivers“
 - Eingeschränkte Portabilität
 - Oracle's OCI (Oracle Call Interface) client-side driver

JDBC Treiber Typen

- Typ 3
 - Java-Treiber
 - Kommunikation mit Middleware Server über datenbankunabhängiges Protokoll
 - Middleware Server übersetzt auf datenbankabhängiges Protokoll
- Typ 4
 - Java-Treiber
 - Kommunikation direkt mit Datenbank
 - Keine Software auf Clientseite
 - Geringere Flexibilität
 - „Thin driver“
 - Für Übungen empfohlen: ojdbc8.jar

Phasen eines JDBC-Datenbankzugriffs



JDBC-Programm

Unvollständige Auszüge

```
import java.sql.*;  
  
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());  
  
String url= "jdbc:oracle:thin:@oracle19c.in.htwg-  
konstanz.de:1521:ora19c";  
  
Connection conn = DriverManager.getConnection( url, "name","passwd");  
Statement stmt = conn.createStatement();  
  
ResultSet rs = stmt.executeQuery(  
        "SELECT name, gehalt FROM pers WHERE anr = 'K55'");  
  
while (rs.next()) {  
    String r1 = rs.getString("name");  
    Int r2 = rs.getInt("gehalt");  
    system.out.println(r1 + " verdient " + r2);  
}  
  
rs.close(); stmt.close(); conn.close();
```

Include SQL

Treiber laden

Verbindung zur DB herstellen

Statement erzeugen

Anweisung absenden

Ergebnisse verarbeiten

JDBC-API

- Package `java.sql`
- `Connection`
 - Interface zu einer spezifischen Datenbank
 - Verbindungsauflaufbau ohne Benutzer und Passwort

```
static Connection getConnection (String url)
```
 - Verbindungsauflaufbau mit Benutzer und Passwort

```
static Connection getConnection  
(String url, String user, String password)
```
- `Statement`
 - Ausführung von SQL-Statements
 - Statement `s = conn.createStatement();`

JDBC-API

- **Klasse Statement – Suchen**

- `public ResultSet executeQuery(String sql)
throws SQLException;`

- **Klasse Statement – Updates**

- `public int executeUpdate(String sql)
throws SQLException;`
 - Rückgabewert: Row count für insert, update und delete-Statements, bzw. 0 falls Statement keine Rückgabe besitzt

- **ResultSet – Ergebnis einer SQL-Suche**

- Zugriff über Spaltenname oder Index
 - `ResultSet rs = s.executeQuery("sqlAusdruck");`
 - `String r1 = rs.getString("name"); // Spaltenname`
 - `String r2 = rs.getString(1); // Index`

JDBC-API

- Methoden der Klasse **SQLException**
 - `getMessage()`: Fehlermeldung mit ORA-Nummer
 - `getErrorCode()`: ORA-Nummer
 - `getSQLState()`: Code für SQL-Zustand
 - `printStackTrace()`: Exception Stack Trace
- Transaktionen
 - `conn.commit();`
 - `conn.rollback();`

Erweiterte Möglichkeiten von JDBC

- Abfrage Meta-Informationen (Data Dictionary)
 - Methode `Connection.getMetaData`
- Änderbare Ergebnismengen
 - Werte innerhalb einer Ergebnismenge können verändert werden
- Scrollbare Ergebnismengen
 - Cursor kann vorwärts und rückwärts positioniert werden
 - Absolute Positionierung
 - Cursor-Position kann ermittelt werden
- Transaktionsbehandlung
 - Commit und Rollback von Transaktionen

Scrollbar Cursor in JDBC 2.0

- Festlegen der Eigenschaften im Statement

- ```
Statement stmt =
conn.createStatement(int resultSetType,
int resultSetConcurrency);
```

- Parameter ResultSetType

| resultSetType                     | Bedeutung                                                                       |
|-----------------------------------|---------------------------------------------------------------------------------|
| ResultSet.TYPE_FORWARD_ONLY       | ResultSet kann nur einmal vom ersten bis zum letzten Element durchlaufen werden |
| ResultSet.TYPE_SCROLL_INSENSITIVE | ResultSet ist scrollbar, simultane Änderungen anderer Nutzer bleiben verborgen  |
| ResultSet.TYPE_SCROLL_SENSITIVE   | ResultSet ist scrollbar, Änderungen anderer Nutzer schlagen auf Werte durch     |

# Scrollbar Cursor in JDBC 2.0

- Parameter ResultSetConcurrency

| resultSetConcurrency       | Bedeutung                            |
|----------------------------|--------------------------------------|
| ResultSet.CONCUR_READ_ONLY | Ergebnis kann nicht verändert werden |
| ResultSet.CONCUR_UPDATABLE | Ergebnis kann editiert werden        |

# JDBC-Programm mit scrollbarem Cursor

```
Statement stmt =
 conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
 ResultSet.CONCUR_UPDATABLE) ;

ResultSet rs =
 stmt.executeQuery("SELECT * FROM pers");

rs.absolute(5); // Cursor auf Position 5 setzen
rs.first(); // Cursor auf erste Position setzen
rs.next(); // Cursor nächste Position setzen
rs.previous(); // Cursor vorherige Position setzen
rs.updateDouble("Gehalt", 40000); // Wert ändern

conn.commit(); // Commit
conn.rollback(); // Rollback
...
...
```

# JDBC – weitere Methoden

```
...

boolean b;

int pos = rs.getRow() // Ermittelt Cursor Position

rs.first() // Bewegt Cursor auf erstes Element
rs.beforeFirst() // Bewegt Cursor vor erstes Element
 // ergibt true bei leerem ResultSet
 // praktisch bei folgender Schleife
 // while(rs.next()) {...}

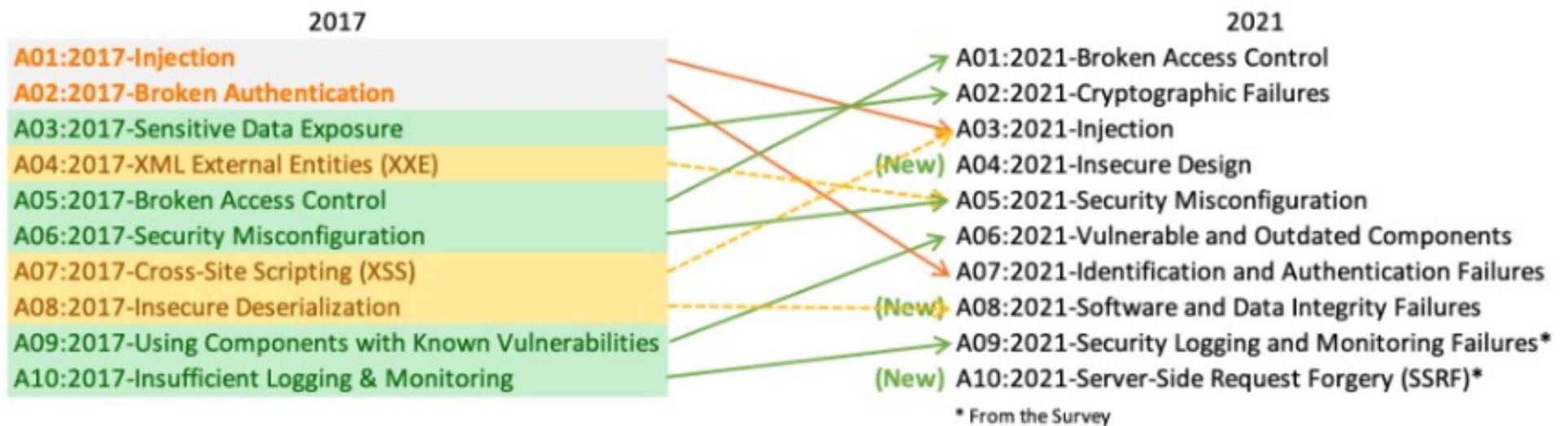
rs.last() // Bewegt Cursor auf letztes Element
rs.afterLast() // Bewegt Cursor auf Ende ResultSet

b = rs.isFirst() // Test ob Cursor auf erstem Element
b = rs.isBeforeFirst(); // Test ob Cursor vor erstem Element

b = rs.isLast(); // Test ob Cursor auf letztem Element
b = rs.isAfterLast(); // Test ob Cursor nach letztem Element
```

# SQL-Injection

- Open Web Application Security Project (OWASP)
  - SQL-Injection ist eine der am meisten ausgenutzten Sicherheitslücken bei Webanwendungen
  - In Top Ten noch vor Authentifikation
  - Übernahme von ganzen Servern durch Betriebssystemschnittstellen



Quelle: <https://owasp.org/www-project-top-ten/>

# SQL-Injection in Webseiten

Benutzereingabe im Browser



get-Methode

Aufruf Website

`http://webserver/cgi-bin/find.cgi?ID=118`



SQL-Anfrage im Server

`SELECT name FROM pers WHERE pnr=118`

HTML-Seite mit Ergebnis

- SQL-Anfragen in Webseiten
  - Weiterleitung von Parametern an SQL-Server
  - Ausführung der SQL-Anfrage
  - Generierung einer HTML-Seite mit dem Ergebnis der Anfrage
  - Vorsicht vor Sicherheitslücken, z.B. durch SQL-Injection !

# SQL-Injection

**Name:** Schopenhauer

**Passwort:** weissIchNichtAberEgal' or 'x' = 'x'

- Resultierende SQL-Anfragen

```
SELECT *
FROM Studenten s JOIN prüfen p ON s.matrNr = p.matrNr
WHERE s.name = 'Schopenhauer'
AND s.passwort = 'weissIchNichtAberEgal' OR 'x' = 'x'
```

Quelle: Kemper: Datenbanksysteme

# SQL-Injection

Name: Schopenhauer

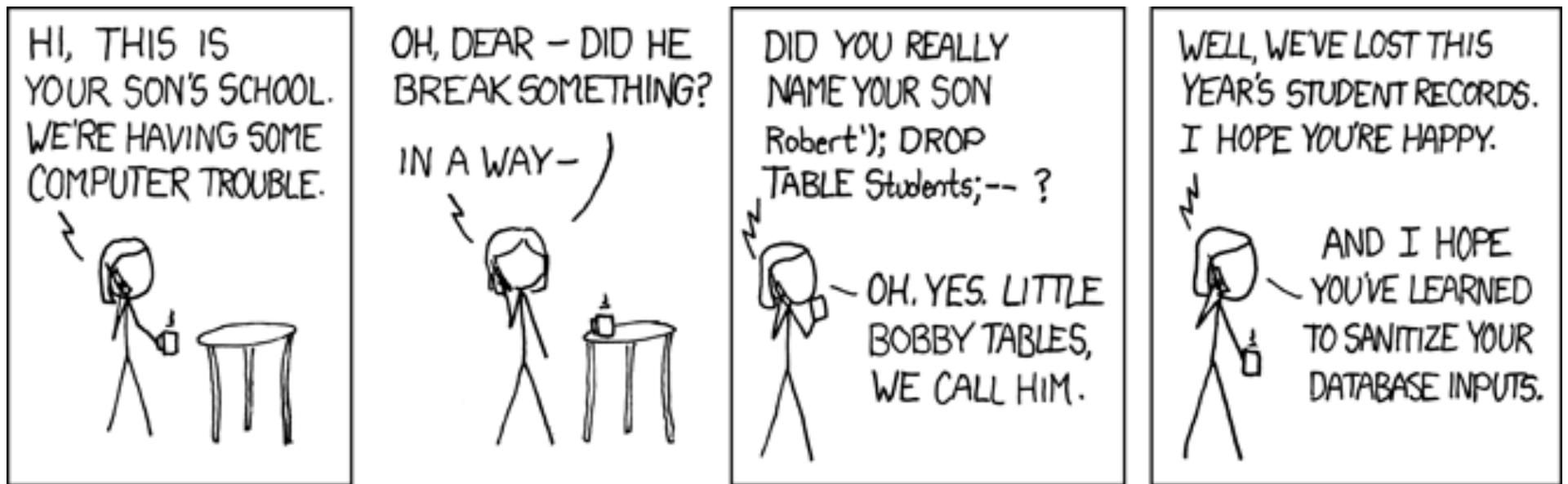
Passwort: Egal'; delete from prüfen where 'x' = 'x'

- Resultierende SQL-Anfragen

```
SELECT *
FROM Studenten s JOIN prüfen p ON s.MatrNr = p.MatrNr
WHERE s.Name = 'Schopenhauer' AND
s.Passwort = 'Egal'; DELETE FROM prüfen WHERE 'x' = 'x'
```

Quelle: Kemper: Datenbanksysteme

# Don't trust user input



Quelle: B. Karwin: SQL Antipatterns

# Prepared Statements in JDBC

- Prepared Statement
  - „Vorbereitete Anweisung“
  - Wird vorübersetzt, daher Geschwindigkeitsvorteil
  - Enthält noch keine Parameterwerte
  - Übergabe durch Platzhalter
  - Verhindern von SQL-Injections

```
PreparedStatement ps = Connection.prepareStatement(
 "SELECT gehalt, beruf FROM pers WHERE name=?");

ps.setString(1, persname);

ResultSet rs = ps.executeQuery();
```

# Parameter binding

- SQL-Query ohne Parameter binding
  - Erzeugung des Strings eventuell durch Konkatenation (mit „+“)

```
query = "SELECT * FROM Abt
 WHERE name = ''
 + stringvar1
 + '' AND ort = ''
 + stringvar2 + '";
```

- SQL-Query mit Parameter binding
  - Nachträgliches Binden der variablen Anteile

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM Abt
WHERE name = ? AND ort = ?";
...
c.setString(1, "Entwicklung");
c.setString(2, "Konstanz");
```

# JDBC und Large Objects

- Locator
  - Logischer Pointer auf die SQL LOBs statt direkter Daten
- Zugriff von JDBC auf LOBs
  - Zugriff auf LOBs durch Locator
  - Klassen `oracle.sql.BLOB` und `oracle.sql.CLOB`
  - Zugriff auf Locator durch `ResultSet.getObject()`

```
ResultSet rs =
 stmt.executeQuery
 ("SELECT blob_col, clob_col FROM lob_table");
while (rs.next())
{
 oracle.sql.Blob blob = (BLOB)rs.getObject(1);
 oracle.sql.Clob clob = (CLOB)rs.getObject(2);
 ...
}
```

# JDBC und Large Objects

- Lesen und Schreiben von BLOB und CLOB-Daten
  - Verwendung von Methoden zum Lesen von LOBs in Input Stream bzw. Schreiben von Output Stream in LOBs
  - Lesen aus einem BLOB: `getBinaryStream()`
  - Schreiben in ein BLOB: `getBinaryOutputStream()`
  - Lesen aus einem CLOB: `getCharacterStream()`
  - Schreiben in ein CLOB: `getCharacterOutputStream()`

```
java.io.OutputStream outstream;

byte[] data = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

// write the array of binary data to a BLOB
outstream = ((BLOB)my_blob).getBinaryOutputStream();
outstream.write(data);
...
```

Quelle: Oracle Database Online Documentation

# JDBC und Large Objects

- Wichtig
  - Die “Write“-Methoden schreiben direkt in die Datenbank, wenn man auf den OutputStream schreibt
  - Kein Update/Commit notwendig um die Daten zu schreiben

```
java.io.Writer writer

// read data into a character array
char[] data = {'0','1','2','3','4','5','6','7','8','9'};

// write the array of character data to a CLOB
writer = ((CLOB)my_clob).getCharacterOutputStream();
writer.write(data);
writer.flush();
writer.close();
...
```

Quelle: Oracle Database Online Documentation

# JDBC und Large Objects

- Weitere Methode von `java.sql.Blob`
  - `byte[] getBytes(long pos, int length)`
- Weitere Methode von `java.sql.Clob`
  - `String getSubString(long pos, int length)`

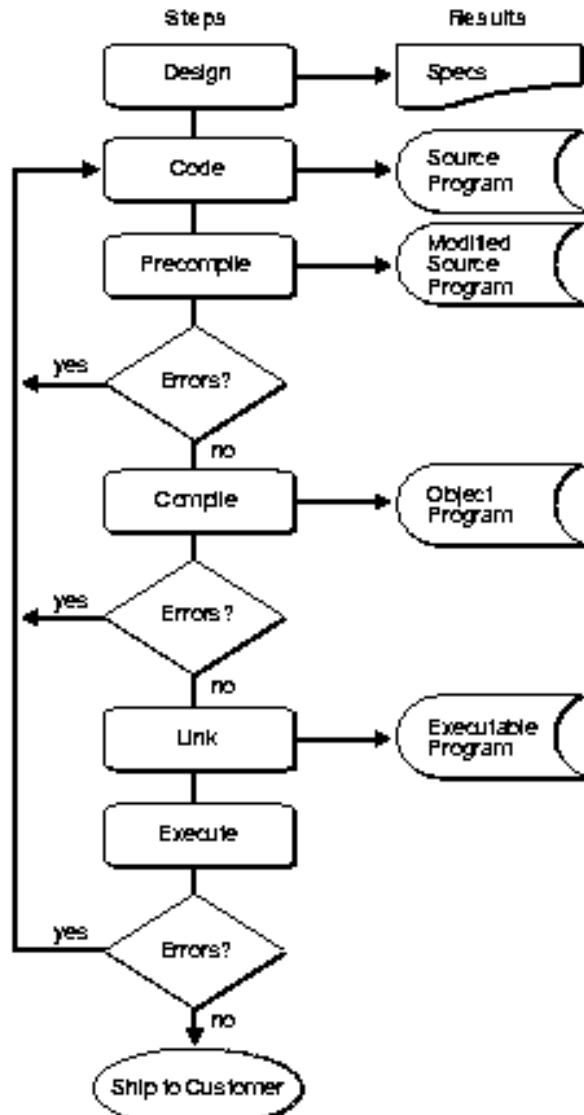
Quelle: Oracle Database Online Documentation

## 4.3 SQL

### Programmiersprachen-Anbindung

- Grundlagen
- Fortgeschrittene Konzepte
- Programmiersprachen-Anbindung
  - JDBC
  - Embedded SQL

# Embedded SQL Anwendungsentwicklung



Quelle: Oracle Database Online Documentation

# Embedded SQL

## Ein einfaches Beispielprogramm

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL WHENEVER SQLERROR GOTO fehler;

cout << "Geben Sie das neue Gehalt an: ";
cin >> gehalt;

EXEC SQL UPDATE pers
 SET Gehalt = :gehalt
 WHERE pnr = 1234;

EXEC SQL COMMIT WORK; // Sichern seit letztem Commit

return 0;

fehler:
cout << "Fehler beim Update der Relation Personal\n";
EXEC SQL ROLLBACK WORK; //Rücknahme seit letztem Commit
```

# Embedded SQL (ESQL)

- Host-Variable
  - Variable der Wirtssprache, die auch in SQL-Anweisungen verwendet werden kann
  - Gesonderte Deklaration

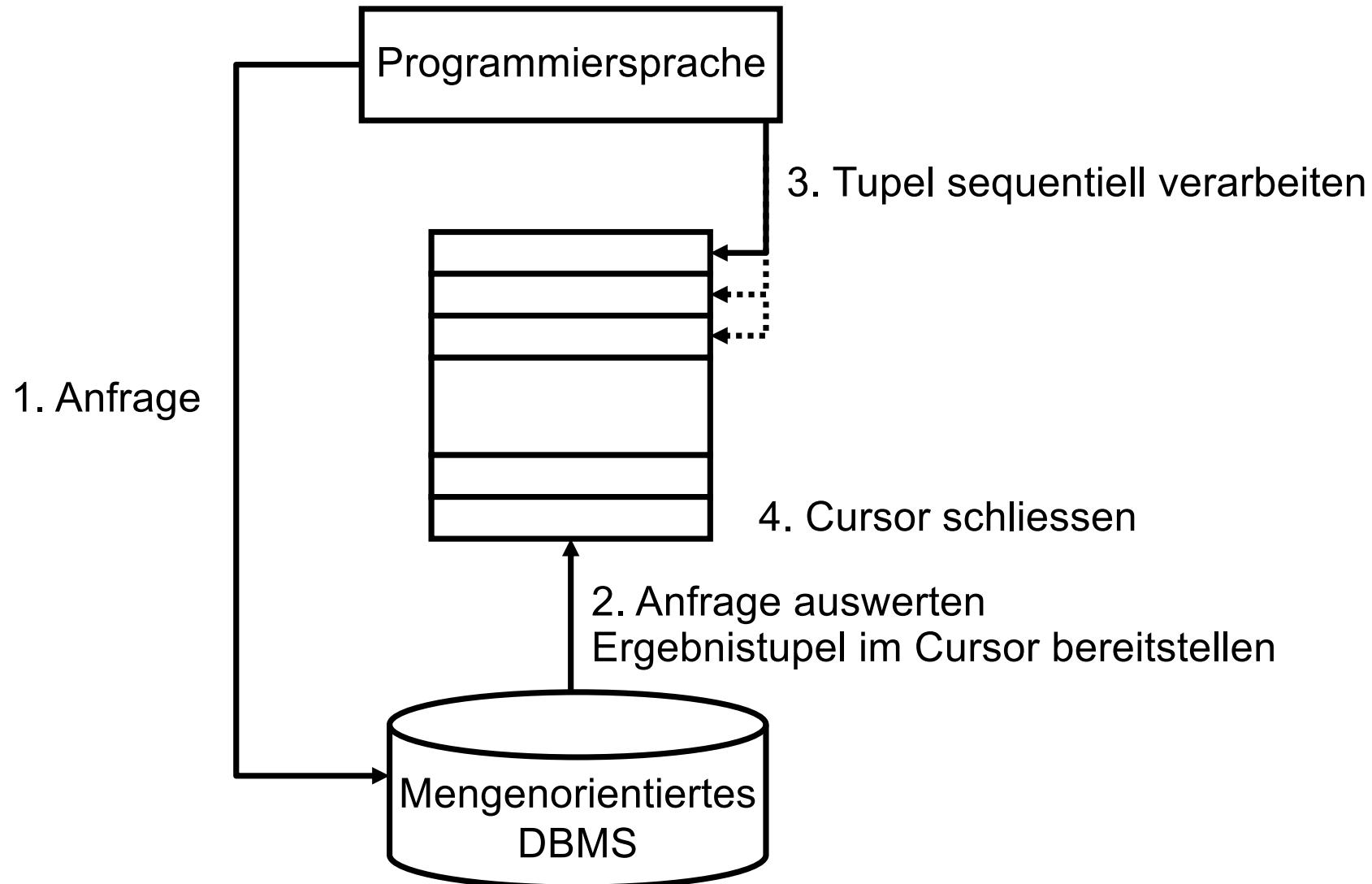
```
EXEC SQL BEGIN DECLARE SECTION
...
EXEC SQL END DECLARE SECTION;
```
  - Verwendung in SQL :Variablenname
- Variablen zur Speicherung von Fehlercodes
  - SQLCODE: Fehlercode, Fehler falls >0
  - SQLSTATE: Genormter Statuscode, der Erfolg oder Fehlermeldung eines Befehls angibt

# Embedded SQL

## Einführung Cursor

- Kopieren des Ergebnisses einer Anfrage nur möglich, wenn höchstens ein Tupel zurückgeliefert wird
  - ```
EXEC SQL SELECT AVG(gehalt)
    INTO :avggehalt
    FROM pers;
```
- Einführung eines Cursors
 - Ein Cursor ist eine (Zeiger-) Variable, welche explizit für jeden SELECT-Befehl, dessen Antworttabelle mit dem Cursor durchlaufen werden soll, deklariert werden muß
- Ein Cursor wird benötigt
 - wenn ein SQL-Befehl mehr als ein Tupel als Antwort liefert
- Ein Cursor wird nicht benötigt
 - bei INSERT-, UPDATE, DELETE-Befehlen
 - wenn höchstens ein Tupel als Antwort geliefert wird

Cursor-Konzept



Cursor-Konzept

- Cursor-Deklaration
 - EXEC SQL DECLARE cursor-name CURSOR FOR query
[FOR {READ ONLY | UPDATE [OF SPALTE] }]
- Cursor-Eröffnung
 - EXEC SQL OPEN cursor-name
- Übergabe der Tupelwerte an Host-Variable
 - EXEC SQL FETCH cursor-name INTO :var1 [, :var2]
- Update eines aktuellen Tupels
 - EXEC SQL UPDATE table-name SET column-name = expr1
WHERE CURRENT OF cursor-name
- Löschen eines aktuellen Tupels
 - EXEC SQL DELETE FROM table-name
WHERE CURRENT OF cursor-name
- Schließen des Cursors
 - EXEC SQL CLOSE cursor-name

Embedded SQL

Beispielprogramm 2

```
EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR uid[20];
  VARCHAR pwd[20];
  VARCHAR ort[20];
  VARCHAR name[20];
  VARCHAR beruf[20];
EXEC SQL END DECLARE SECTION;

strcpy(ort.arr, "Konstanz");

EXEC SQL INCLUDE sqlca;
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

EXEC SQL DECLARE C1 CURSOR FOR
  SELECT pers.name, pers.beruf
  FROM pers, abt
  WHERE pers.anr = abt.anr AND ort = :ort;
EXEC SQL OPEN C1;

EXEC SQL WHENEVER NOT FOUND DO break;
for (;;)
{
  EXEC SQL FETCH C1 INTO :name, :beruf;
  printf ("%s %s\n", name.arr, beruf.arr);
}

EXEC SQL CLOSE C1;
```

Embedded SQL

Vergleich alternativer Programmierstile

```
...  
EXEC SQL INCLUDE sqlca;  
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;  
  
EXEC SQL DECLARE C1 CURSOR FOR  
  SELECT pers.name, pers.beruf  
  FROM pers, abt  
  WHERE pers.anr = abt.anr AND ort = :ort;  
EXEC SQL OPEN C1;  
  
EXEC SQL WHENEVER NOT FOUND DO break;  
for (;;) {  
  EXEC SQL FETCH C1 INTO :name, :beruf;  
  printf ("%s %s\n", name.arr, beruf.arr);  
}  
...
```

Alternative 1

```
...  
EXEC SQL INCLUDE sqlca;  
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;  
  
EXEC SQL DECLARE C1 CURSOR FOR  
  SELECT pers.name, pers.beruf, pers.ort  
  FROM pers, abt  
  WHERE pers.anr = abt.anr;  
EXEC SQL OPEN C1;  
  
EXEC SQL WHENEVER NOT FOUND DO break;  
for (;;) {  
  EXEC SQL FETCH C1 INTO :name, :beruf, :ort;  
  if (strcmp(ort.arr, "Konstanz"))  
    printf ("%s %s\n",  
           name.arr, beruf.arr);  
}  
...
```

Alternative 2

- Unterschiedliche Programmierstile
 - Test auf Ort wird einmal innerhalb SQL, einmal ausserhalb durchgeführt
 - Welche Alternative ist besser ???

Statische vs. dynamische Einbindung

Statische Einbindung	Dynamische Einbindung
- Anfragen müssen zum Übersetzungszeitpunkt bereits bekannt sein	+ Erstellung von Anfragen zur Laufzeit + flexibel
+ leichter lesbar und verständlich	- schwerer lesbar
+ Syntaktische und semantische Korrektheit wird bereits zum Übersetzungszeitpunkt geprüft	
+ Performancevorteil, da Optimierung von Anfragen bereits zum Übersetzungszeitpunkt	

Weitere Ansätze

- LINQ
 - Komponente von Microsofts .NET-Framework

```
var query = from article in this.Articles  
            where article.Name.StartsWith("A")  
            orderby article.ID  
            select article;  
  
foreach (var article in query) {  
    Console.WriteLine (article.Name );  
}
```

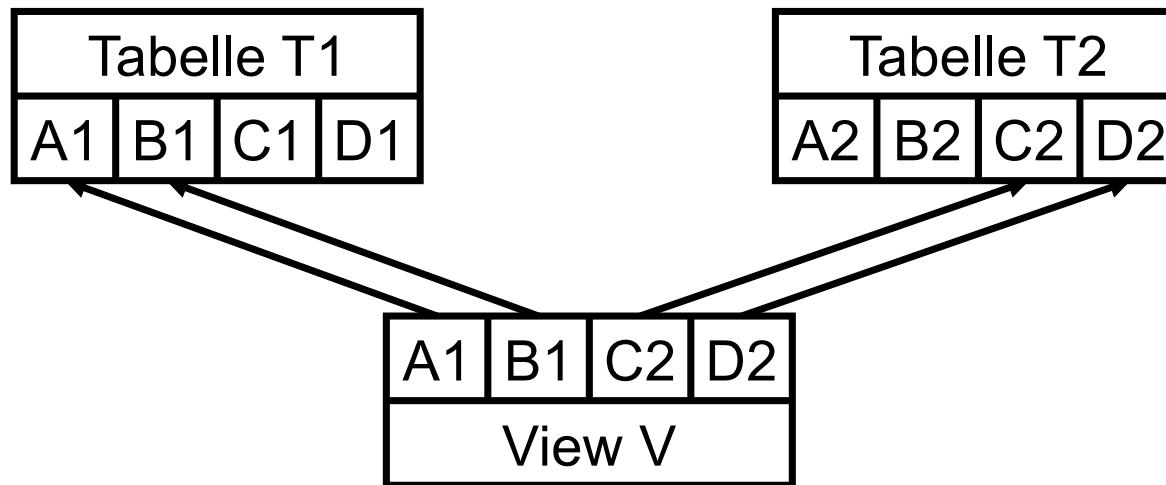
- Hibernate
 - Object-Relational Mapping (OR-Mapping, oder kurz ORM)
 - Mapping Java Klassen auf Tabellen
- JDO (Java Data Objects)
 - Framework zur persistenten Speicherung von Java-Objekten

Übersicht

- 4-3 SQL-Anfragen 2
 - Views
 - Join-Typen
 - Trigger
 - Indexierung
 - Large Objects
 - Data Dictionary
 - Hierarchische Anfragen

Views (Sichten, virtuelle Tabellen)

- Zusammenstellung von Daten einer gewünschten Sichtweise
 - Datenschutz
 - Ausblenden unwichtiger Informationen (Filter)
- Views enthalten keine Daten, sondern Zeiger



Views

Definition und Ausgabe

```
CREATE VIEW Gehalt_der_Chefs
AS
SELECT pnr, name, gehalt
FROM pers p1
WHERE EXISTS (SELECT *
               FROM pers p2
               WHERE p2.vnr=p1.pnr);
```

- Ausgabe

```
SELECT *
FROM Gehalt_der_Chefs;
```

Views

Definition

- Read only-Views
- Viewspalten müssen definiert werden bei
 - SQL-Funktionen
 - Arithmetischen Operationen

```
CREATE VIEW Gehaelter_Abteilungen(anr, abtGehalt)
AS
SELECT anr, SUM(gehalt)
FROM pers
GROUP BY anr
WITH READ ONLY;
```

- View löschen

```
DROP VIEW Gehaelter_Abteilungen;
```

Views

Eigenschaften

- View-Änderungen bewirken Änderungen in Originaltabellen
 - Aktualisierung mit Prüfoption (WITH CHECK OPTION)
- Änderungen nur möglich, wenn
 - Sicht bezieht sich auf eine einzige Tabelle
 - Spalten enthalten keine Funktionen oder arithm. Operationen
 - Keine Verwendung von GROUP-BY
 - Kein read-only definiert
- Vorteile
 - Lösung komplexer Probleme in Teilschritten
 - Kapselung von Logik
 - Übersichtlichkeit, Abstraktion
 - Zugriffsschutz

Views

Aktualisieren mit Prüfoption

- Einbringen eines Datensatzes in ein View, der dort nicht vorhanden sein sollte
- Der Datensatz kann nicht mehr ausgegeben werden

```
CREATE VIEW Frauen
AS
SELECT persnr, pname, geschlecht
  FROM persdat
 WHERE geschlecht = 'W';

INSERT INTO Frauen
  VALUES (45, 'Wedefeld', 'M');
```

Views

Aktualisieren mit Prüfoption

- Prüfoption zur Überprüfung, ob Datensatz in Basistabelle eingebracht wird
- Syntax:

WITH CHECK OPTION

```
CREATE VIEW Frauen
AS
SELECT persnr, pname, geschlecht
  FROM persdat
 WHERE geschlecht = 'W'
 WITH CHECK OPTION;
```

```
INSERT INTO Frauen
  VALUES (45, 'Wedefeld', 'M');
```

Views

Beispiele

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

Pers							
<u>pnr</u>	name	jahrg	eindat	gehalt	beruf	<u>anr</u>	<u>vnr</u>
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
<u>anr</u>	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Auf die Tabelle PERS soll eine Sicht erzeugt werden, die nur Programmierer mit weniger als 60.000 € Verdienst enthält.

Views

Beispiele

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Lösche die Sicht ARME_PROGRAMMIERER.

Datenbank-Anfragen

Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

Pers							
<u>pnr</u>	name	jahrg	eindat	gehalt	beruf	<u>anr</u>	<u>vnr</u>
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
<u>anr</u>	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Welcher Vorgesetzte hat die meisten Mitarbeiter?

Result Limits

- Möglicher Lösungsweg der letzten Anfrage
 - Absteigende Sortierung der Ergebnistupel nach Anzahl Vorgesetzte
 - Abbruch nach erstem Element
- Oracle
 - ```
SELECT * FROM pers
 ORDER BY gehalt DESC
 FETCH FIRST ROW ONLY;
```
- Nachteile
  - Nur eine Ausgabe, auch wenn mehrere Tupel das Maximum erzielen
  - Unterschiedliche Implementierung in verschiedenen DBMS
- Realisierung in anderen Datenbanksystemen
  - MySQL: `SELECT * FROM pers LIMIT 5`
  - MS SQL Server: `SELECT TOP 5 * FROM pers`

# Lösung des Beispiels mit WITH-Klausel

- With-Klausel
  - View, der auf eine Anweisung begrenzt ist
  - Wird nicht dauerhaft gespeichert
  - Wird direkt vor eigentlichem Select geschrieben
  - Auch mehrere Views können so definiert werden

```
WITH Anzahl AS
 (SELECT vnr, COUNT(*) AS anz
 FROM Pers
 WHERE vnr IS NOT NULL
 GROUP BY vnr)
SELECT vnr
FROM Anzahl a
WHERE a.anz = (SELECT MAX(anz) FROM Anzahl);
```

# Übersicht

- 4.3 SQL-Anfragen 2
  - Views
  - Join-Typen
  - Trigger
  - Indexierung
  - Large Objects
  - Data Dictionary
  - Hierarchische Anfragen

An SQL query walks into a bar and sees two tables.  
He walks up to them and asks 'Can I join you?'

# Inner- und Outer-Joins

| Pers.name | Abt.name    |
|-----------|-------------|
| Coy       | Personal    |
| Mueller   | Entwicklung |
| Schmidt   | Buchhaltung |
| Abel      | Personal    |
| Junghans  | Entwicklung |
| ...       | ...         |

Gebe alle Mitarbeiter und ihr Abteilungsname aus:

```
SELECT name, name
FROM pers, abt
WHERE pers.anr = abt.anr
```

UNION

```
SELECT name, NULL as name
FROM pers
WHERE anr IS NULL
```



Alle Mitarbeiter, denen eine  
Abteilung zugeordnet ist



Alle Mitarbeiter, denen keine  
Abteilung zugeordnet ist

# Join-Typen in SQL99

- Cross-Join
  - Entspricht kartesischem Produkt ohne Bedingungen
  - Wird in der Praxis selten verwendet
- Join mit ON
  - Explizite Festlegung der Spalten für eine Join-Bedingung durch ON-Klausel, falls Spalten unterschiedliche Namen haben
- Natural-Join
  - Alle Spalten mit gleichen Namen und Typ werden in Join-Bedingung eingebunden
- USING
  - Festlegung der Join-Bedingung durch USING-Klausel
  - Beispiel: `pers JOIN abt USING (anr)`

# Join-Typen

- Inner-Join
  - Enthält nur Daten aus den Tabellen, aus denen eine gemeinsame Übereinstimmung existiert
- Outer-Join
  - Beinhaltet auch Daten, die keine verwandten Daten aus einer anderen Tabelle existiert
  - Drei Typen:
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- Syntax Join-Statement
  - FROM QuellTabelle1 [Join-Typ] Tabelle [ON (Join Bedingung)]

# Outer-Joins

## Inner-Join:

```
SELECT pers.name, abt.name
FROM pers INNER JOIN abt ON pers.anr = abt.anr;
```

## Left-Outer-Join (Left Join):

```
SELECT pers.name, abt.name
FROM pers LEFT OUTER JOIN abt ON pers.anr = abt.anr;
```

## Right-Outer-Join (Right Join):

```
SELECT pers.name, abt.name
FROM pers RIGHT OUTER JOIN abt ON pers.anr = abt.anr;
```

## Full-Outer-Join (Full Join):

```
SELECT pers.name, abt.name
FROM pers FULL OUTER JOIN abt ON pers.anr = abt.anr;
```

# Inner-Join

Die folgenden Joins sind äquivalent

```
SELECT p.name, a.name
FROM pers p INNER JOIN abt a
ON p.anr = a.anr
WHERE a.name = 'Personal';
```

```
SELECT p.name, a.name
FROM pers p INNER JOIN abt a
ON p.anr = a.anr
AND a.name = 'Personal';
```



# Weitere Darstellung für Joins

Syntax von Oracle vor Version 9i, **bitte nicht mehr verwenden !**

## Left-Outer-Join

```
SELECT x.name, y.name
FROM pers x, abt y
WHERE x.anr(+) = y.anr;
```

## Right-Outer-Join

```
SELECT x.name, y.name
FROM pers x, abt y
WHERE x.anr = y.anr(+);
```

## Full-Outer-Join

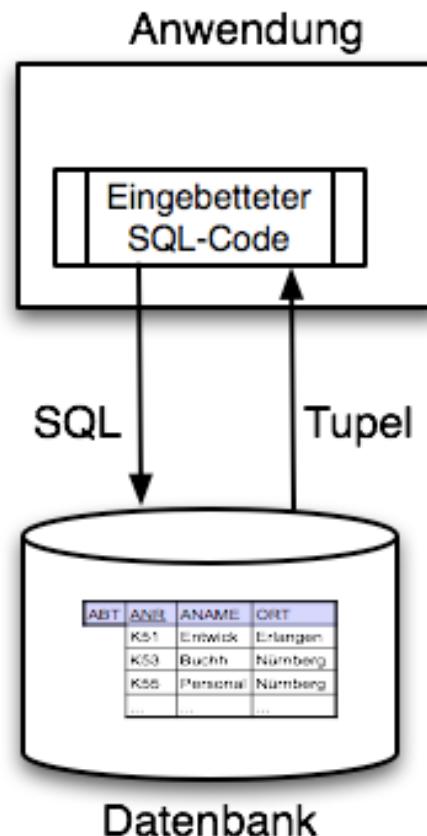
```
SELECT x.name, y.name
FROM pers x, abt y
WHERE x.anr(+) = y.anr(+);
```

# Übersicht

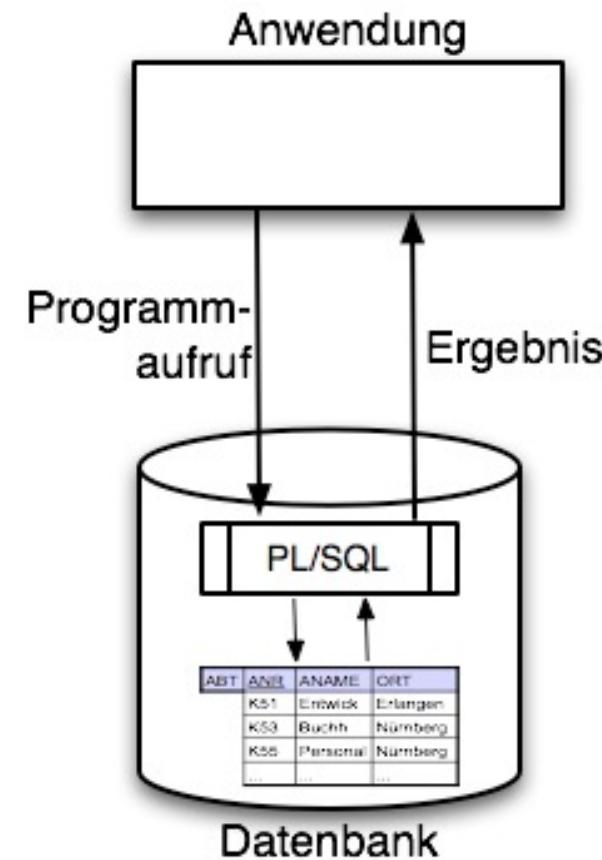
- SQL-Anfragen 2
  - Views
  - Join-Typen
  - Trigger
  - Indexierung
  - Large Objects
  - Data Dictionary
  - Rekursion

# PL/SQL - Datenbankprogrammierung in Oracle

Datenbankoperationen bisher



Datenbankoperationen mit  
Stored Procedures und PL/SQL



# Stored Procedures

- Stored Procedures (gespeicherte Prozeduren)
  - Speichern und Ausführen von Anwendungslogik direkt im Datenbanksystem
- Vorteile
  - Performance, da weniger Datentransfer
  - Vermeidung von Code-Redundanzen
  - Erhöhung der Datensicherheit
  - Einhaltung von komplexen Integritätsregeln
- Prozeduren und Funktionen in Oracle
  - CREATE PROCEDURE bzw. CREATE FUNCTION
  - EXECUTE

# PL/SQL - Datenbankprogrammierung in Oracle

- PL/SQL: Prozedurales Sprach-Superset von SQL
- Sprache mit Programmiermöglichkeiten (Schleifen, bedingten Verzweigungen, explizite Fehlerbehandlung)
- Gruppierung in Blöcke
- Ein Block umfasst drei Bereiche
  - Deklarationen: Definition von Variablen
  - Executable-Befehle: Ausführung von Befehlen
  - Exception-Behandlung: Verarbeitung von Fehlerbedingungen
- Mengenwertige Abfragen durch Cursor (s. Folien über Embedded SQL)

# PL/SQL-Programm

## Beispiel

```
declare
 pi constant NUMBER(9,7) := 3.1415927;
 radius INTEGER(5);
 area NUMBER(14,2);
 some_variable NUMBER(14,2);
begin
 radius := 2;
 loop
 some_variable := 1/(radius-7);
 area := pi*power(radius,2);
 insert into AREAS values (radius, area);
 radius := radius+1;
 exit when area>1000;
 end loop;
exception
 when ZERO_DIVIDE
 then insert into AREAS values (radius,0);
end;
```

Deklaration von Variablen

Kontrollstrukturen und SQL-Befehle

Fehlerbehandlungen

# Triggerkonzept

- Trigger sind gespeicherte Prozeduren, welche bei einem bestimmten Ereignis ausgelöst werden
  - Automatische Durchführung bei der Ausführung einer INSERT-, UPDATE- oder DELETE-Anweisung
  - Anwendung z.B. zur Gewährleistung der Integrität
  - Aufruf von SQL bzw. PL/SQL-Prozeduren (in Oracle)
- Trigger-Elemente ECA aktiver Datenbanken

|           |                           |
|-----------|---------------------------|
| Event     | Regelauslösendes Ereignis |
| Condition | Bedingung                 |
| Action    | Auszuführende Aktionen    |

# Triggerkonzept in SQL

- Ereignis (Event)
  - Löschen von Tupel einer Tabelle (`DELETE`)
  - Hinzufügen von Tupel in eine Tabelle (`INSERT`)
  - Ändern von Attributen eines bestehenden Tupels (`UPDATE`)
  - ...
- Bedingung (Condition)
  - Die Bedingung wird überprüft, wenn das Ereignis aufgetreten ist
- Rumpf (Action)
  - Aktivität wird nur ausgeführt, wenn Bedingung erfüllt ist
  - Auszuführende SQL-Anweisungen

# Triggerkonzept in SQL

- Aktivierungszeitpunkt
  - BEFORE: Aktivierung vor der ereignisauslösenden Anweisung
  - AFTER: Aktivierung nach der ereignisauslösenden Anweisung
  - INSTEAD OF: Aktivierung anstatt der Anweisung
- Transitions-Variable
  - NEW: Zugriff auf den Attributwert *nach* der Änderung
  - OLD: Zugriff auf den Attributwert *vor* der Änderung
  - Im PL/SQL-Block ist den Schlüsselwörter ein NEW und OLD ein Doppelpunkt ":" voranzustellen (z.B. :new.name)
- Zeilen-/Anweisungsebene
  - FOR EACH ROW: Trigger auf Zeilenebene, d.h. Ausführung pro relevante Zeile
  - FOR EACH STATEMENT: Trigger auf Anweisungsebene, d.h. Ausführung pro Befehl

# Triggerkonzept in SQL

- Syntax

- CREATE TRIGGER triggername  
AFTER | BEFORE | INSTEAD OF  
INSERT | DELETE | UPDATE  
[ OF column ] ON tabellenname  
{ FOR EACH STATEMENT | FOR EACH ROW }  
WHEN bedingung  
SQL-Statement | PL/SQL-Code

*Zeitpunkt*

*Ereignis*

*Bedingung*  
*Aktionen*

- Einführung in SQL3

- Rekursive Trigger

- Trigger können selbst weitere Trigger aktivieren
  - Eine Terminierung zyklischer Trigger ist nicht gewährleistet !

# Date Check Constraints

- Umsetzung tabellenübergreifender Constraints
- Lösung des Constraints "buchungsstart > SYSDATE"
  - Mit Check-Constraint nicht möglich
  - Constraint soll auch nur zum Speicherzeitpunkt überprüft werden
  - Lösung mit Trigger zum Einfügezeitpunkt

```
CREATE OR REPLACE TRIGGER BuchungInZukunft
BEFORE INSERT ON Buchung
FOR EACH ROW
begin
 IF (:new.buchungsstart < sysdate)
 THEN RAISE_APPLICATION_ERROR(-20001,
 'Booking must start in the future');
 END IF;
END;
/
```

# Triggerkonzept

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

| Pers       |          |       |          |         |                  |            |            |
|------------|----------|-------|----------|---------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt  | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51        |            |
| 829        | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1997  |          | 80.000  | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...     | ...              | ...        | ...        |

| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | name        | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchhaltung | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

Im Attribut ORT sollen ab sofort nur noch Großbuchstaben verwendet werden.

# Triggerkonzept

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

| Pers       |          |       |          |         |                  |            |            |
|------------|----------|-------|----------|---------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt  | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51        |            |
| 829        | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1997  |          | 80.000  | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...     | ...              | ...        | ...        |

| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | name        | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchhaltung | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

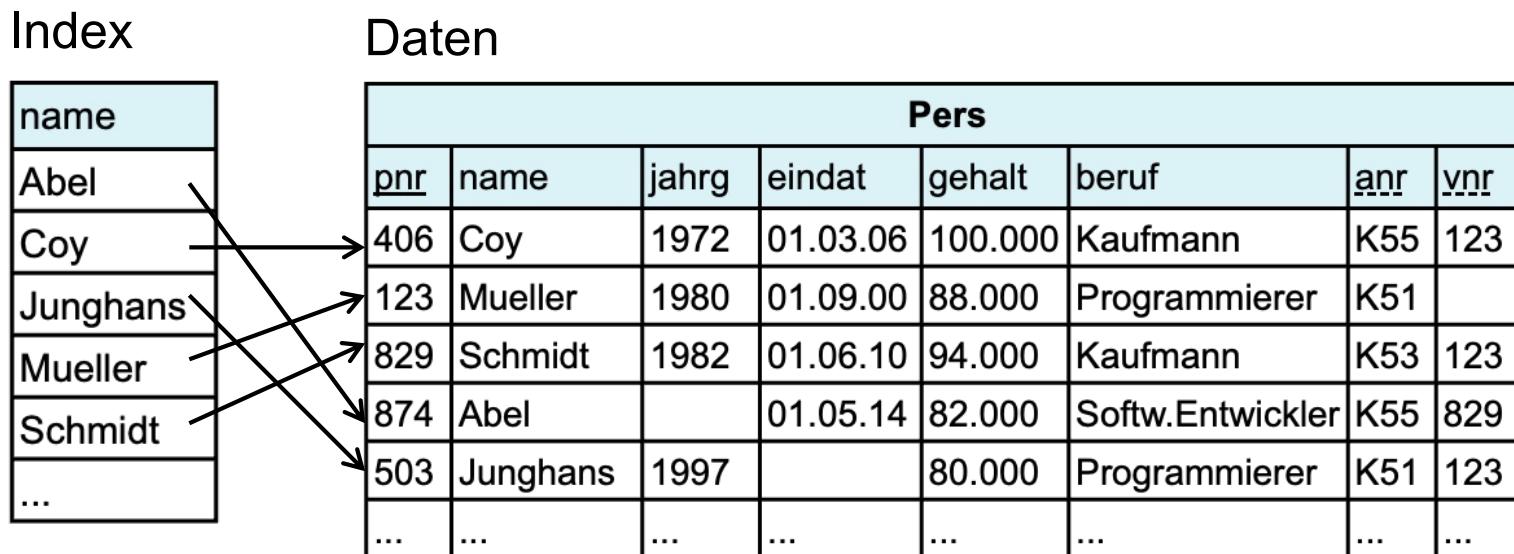
Zu jeder Gehaltserhöhung wird ein Bonus von 100 € dazugerechnet

# Übersicht

- SQL-Anfragen 2
  - Views
  - Join-Typen
  - Trigger
  - Indexierung
  - Large Objects
  - Data Dictionary
  - Hierarchische Anfragen

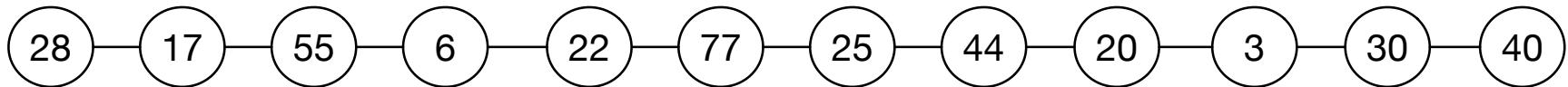
# Indexierung

- Indexierung = Erstellen eines sortierten Schlüssels für Spalten einer Tabelle
- Anfrage
  - `SELECT *  
FROM pers  
WHERE name = 'Mueller';`



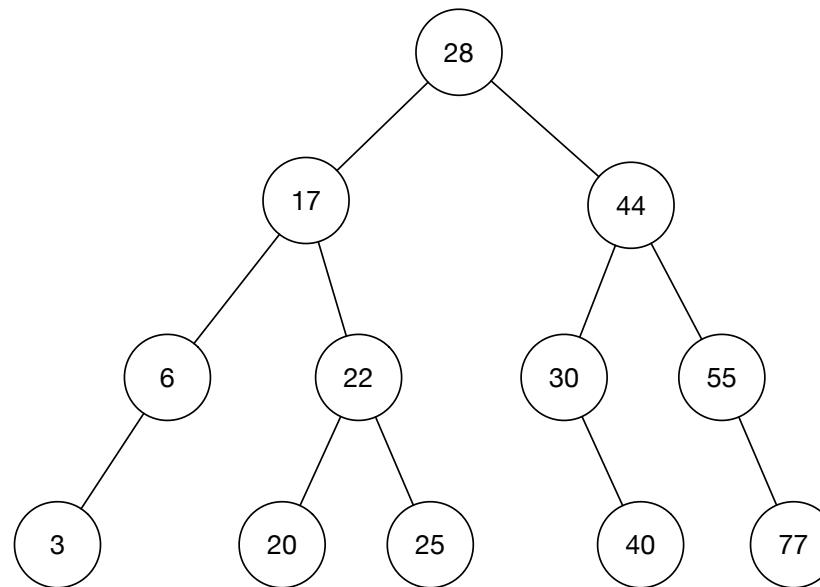
# B-Bäume - Kurzeinführung

- Auf der Festplatte sind die Daten sequentiell gespeichert
- Wieviel Zugriffe auf Elemente benötigt man im unteren Beispiel um die Zahl 30 zu finden, wenn
  - Jede Zahl nur einmal vorkommt?
  - Jede Zahl mehrmals vorkommen kann?
- Wie viel Zugriffe benötigt man bei n Elementen?



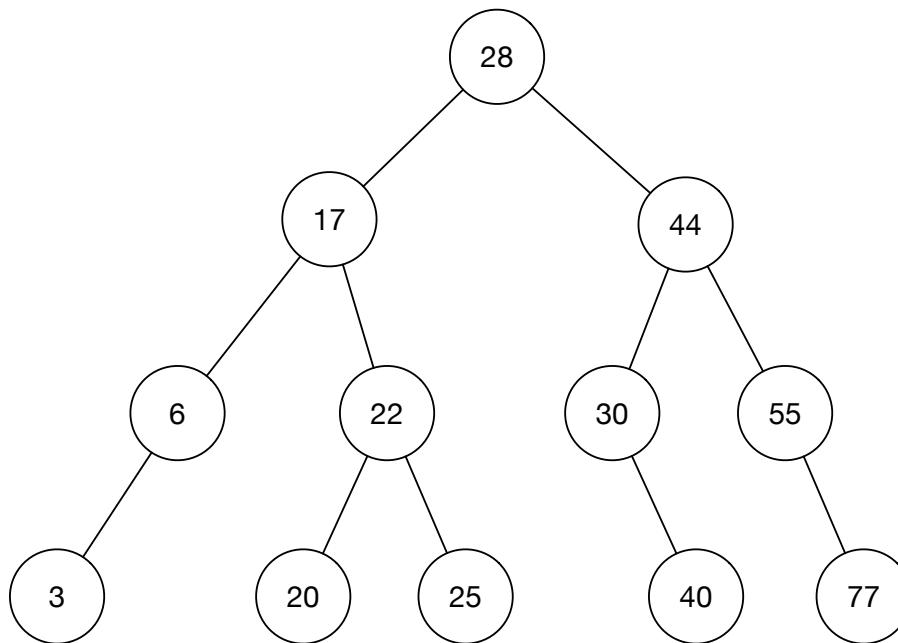
# B-Bäume - Kurzeinführung

- Anordnung der Elemente in Form eines Baumes
  - Alle Elemente im linken Teilbaum sind kleiner als Wurzel
  - Alle Elemente im rechten Teilbaum sind größer als Wurzel
- Suche nach einem Element von oben nach unten
- Bei jedem Vergleich kann ein gesamter Teilbaum ausgeschlossen werden



# B-Bäume - Kurzeinführung

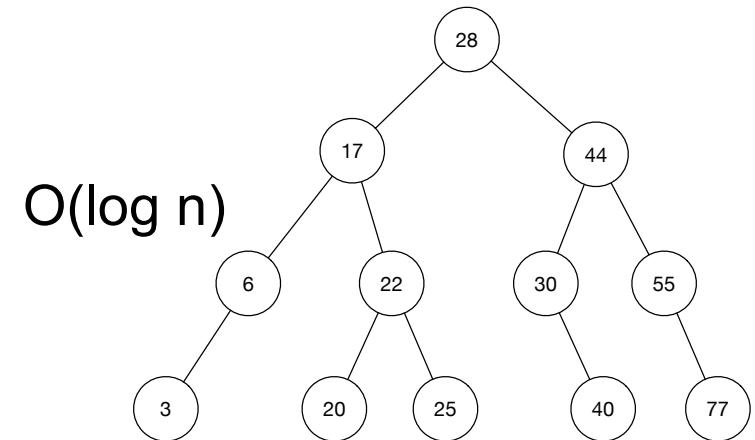
- Wieviel Zugriffe auf Elemente benötigt man im unteren Beispiel um die Zahl 40 zu finden
- Wieviel Zugriffe benötigt man, wenn man n Elemente durchsuchen muss?



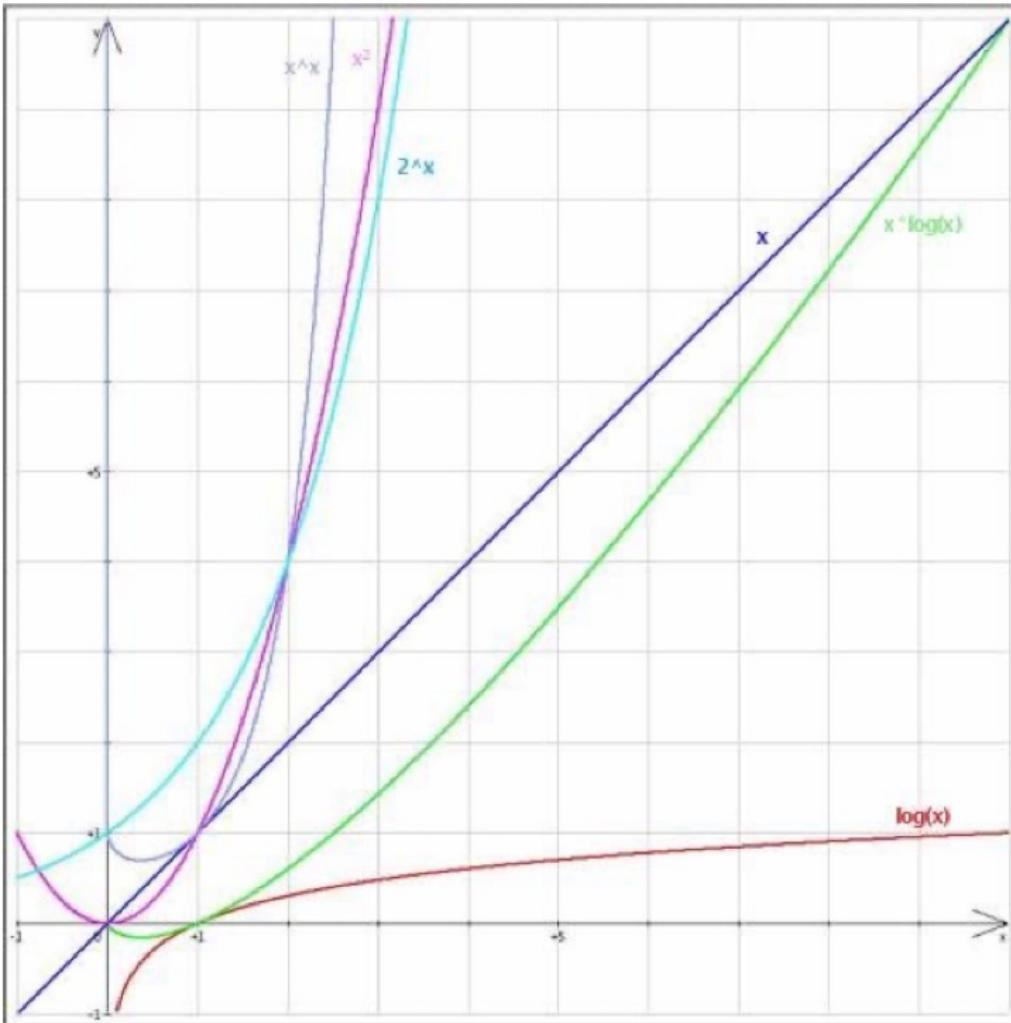
# B-Bäume - Kurzeinführung

- Wieviel Zugriffe auf Elemente benötigt man im unteren Beispiel um die Zahl 30 zu finden wenn man n Elemente durchsuchen muss?
  - Bei jedem Zugriff auf ein Element wird die Anzahl der zu untersuchenden Elemente um die Hälfte reduziert
  - Logarithmischer Zusammenhang

$O(n)$

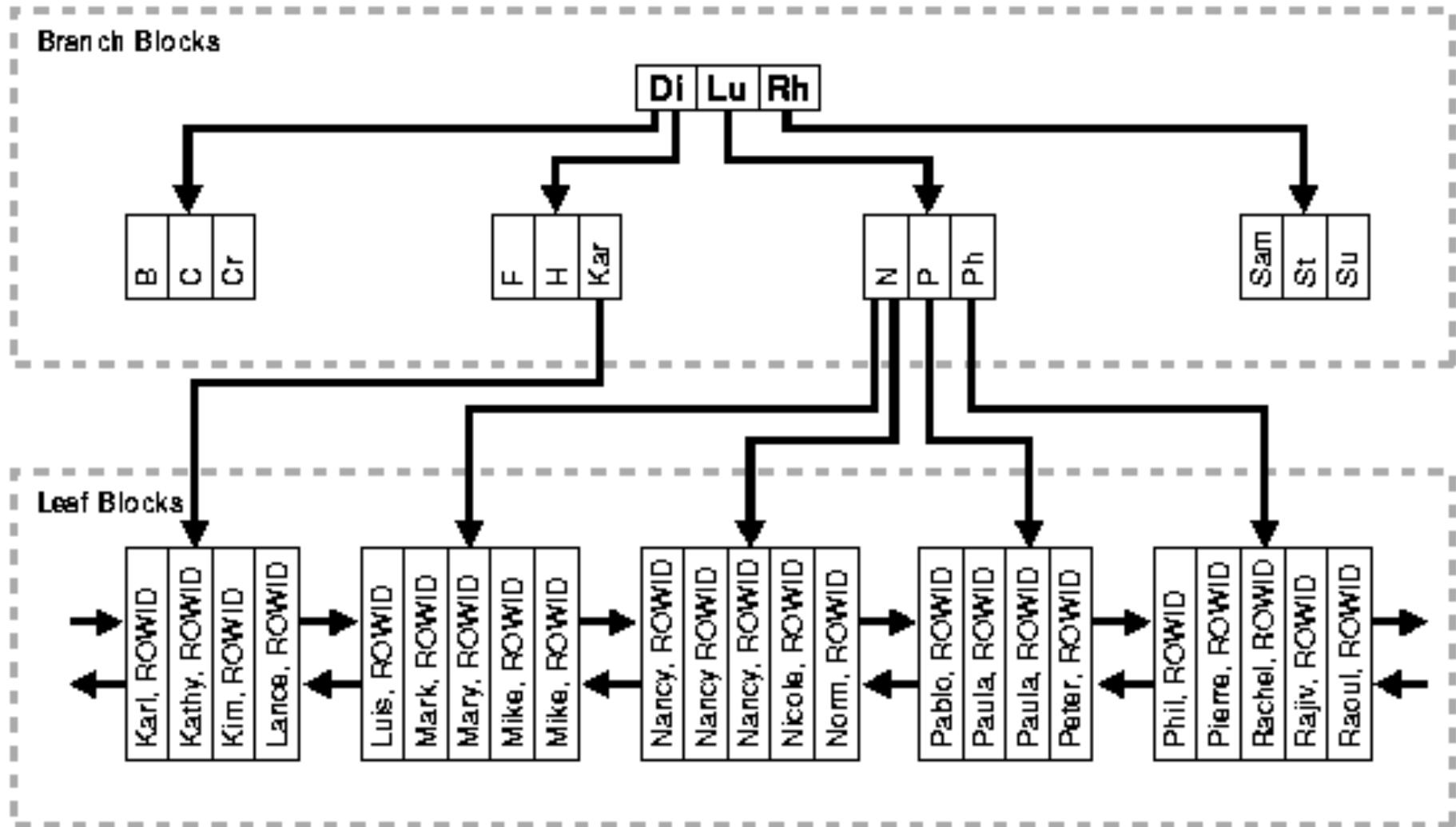


# Komplexitätsklassen



$$\log_2(1.000.000) \approx 20$$
$$\log_2(1.000.000.000) \approx 30$$

# Interne Struktur von Indizes in Oracle



Quelle: Oracle9i Database Online Documentation

# Indexierung

## Performance-Auswirkungen

- Schnellere Suchmethoden, z.B. binäres Suchen
- Keine Auswirkung auf das Ergebnis
  - ORDER-BY muß weiterhin angegeben werden
- Anpassen des Index notwendig nach
  - INSERT
  - UPDATE
  - DELETE
- Indexierung
  - beschleunigt Suchen
  - verlangsamt Aktualisieren

# Indexierung

## Syntax

- Anlegen eines Index
  - `CREATE [UNIQUE] INDEX indexname  
ON tabellenname  
(spaltenname [ASC/DESC]  
, spaltenname [ASC/DESC], ...);`
- Primärindex: Index auf Primärschlüssel  
Sekundärindex: Index auf Nicht-Primärschlüssel-Attribute
- UNIQUE INDEX
  - Es wird verhindert, dass in eine Spalte doppelte Werte eingegeben werden
  - Verwendung bei Spalten mit Primärschlüssel
- Löschen eines Index
  - `DROP INDEX indexname;`

# Index

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Für die Tabelle PERS soll ein Index auf dem Attribut NAME angelegt werden.

# Indexierung

- Ein Index bringt nicht nur Vorteile
  - Daher wichtig: Auswahl der Attribute mit Index
- In den meisten Datenbanksystemen wird auf dem Primärschlüssel automatisch ein Index definiert
- Auf welchem Attribut sollte im Beispiel ein Index definiert werden?
  - ```
SELECT gehalt, beruf, eindat
      FROM pers
     WHERE name = 'Mueller';
```
- Antwort
 - Die Spalten gehalt, beruf und eindat werden nur ausgegeben
 - Ein Index auf diesen Spalten bringt daher keinen Vorteil
 - Da nach name gesucht wird, ist für dieses Attribut ein Index sinnvoll
 - Name ändert sich auch nicht allzu häufig

Übersicht

- SQL-Anfragen 2
 - Views
 - Join-Typen
 - Trigger
 - Indexierung
 - Large Objects
 - Data Dictionary
 - Hierarchische Anfragen

Large Objects (LOBs)

- Internal LOBs
 - BLOB – binary large object
 - CLOB – character large object
 - NCLOB – Texte in nationalen Zeichensatz
- Externe LOBs
 - BFILE: Verweis auf eine externe Datei, die außerhalb der Datenbank gespeichert ist
 - Meist read only
- Wichtiger Unterschied
 - Internal LOBs verwenden Copy Semantics
 - Externe LOBs verwenden Reference Semantics
 - Interne LOB fallen unter Transaktionsmanagement

Large Objects (LOBs)

- Definition von Tabellen mit internal LOBs

```
CREATE TABLE Person
( name      varchar2(30),
  pic       blob(16M));
```

- Zugriff auf LOBs

```
SELECT name, length(pic)  FROM Person
WHERE name = 'Müller';

SELECT p1.name as name1, p2.name as name 2
FROM Person p1, Person p2
WHERE p1.name < p2.name
AND p1.pic = p2.pic
```

Large Objects

- Einschränkungen bei LOBs
 - Operatoren <, <=, >, >= sind nicht definiert
 - LOBs können kein Primärschlüssel sein
 - Kein Index auf LOBs
 - Kein GROUP BY, ORDER BY auf LOBs
- Lesen und Schreiben von Daten aus / in LOBs
 - Siehe nächstes Kapitel unter JDBC

Übersicht

- SQL-Anfragen 2
 - Views
 - Join-Typen
 - Trigger
 - Indexierung
 - Large Objects
 - Data Dictionary
 - Hierarchische Anfragen

Das Data Dictionary von Oracle

- Aufgaben
 - Speicherung aller Informationen, die zur Verwaltung der Objekte in der Datenbank benötigt werden
 - Tabellen- und Spaltennamen werden in Großbuchstaben konvertiert
- Views für Abfrage von Informationen
 - "Landkarten"-Views
 - Tabellen, Views, Spalten
 - Constraints
 - Indizes
 - Zugriffsrechte
 - etc.

Dictionary Views in Oracle

- "User"-Views
 - Präfix "USER"
 - Alle Informationen, die dem Account gehören, der Abfrage ausführt
- "ALL"-Views
 - Präfix "ALL"
 - USER-Datensätze zuzüglich Informationen über Objekte, für die öffentliche oder private Berechtigungen vergeben wurden
- "DBA"-Views
 - Präfix "DBA"
 - Alle Datenbankobjekte ohne Rücksicht auf Benutzer

Einige Beispiele für Dictionary Views

- **USER_TABLES**
 - Tabellen eines Benutzers
- **USER_TAB_COLS**
 - Attribute einer Tabellen eines Benutzers
- **USER_TRIGGERS**
 - Trigger eines Benutzers
- **USER_TAB_PRIVS**
 - Zugriffsrechte auf eine Tabelle eines Benutzers
- **USER_VIEWS**
 - Views eines Benutzers

Das Data Dictionary von Oracle

Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Welche Constraints sind für Tabelle pers definiert?

Übersicht

- SQL-Anfragen 2
 - Views
 - Join-Typen
 - Trigger
 - Indexierung
 - Large Objects
 - Data Dictionary
 - Hierarchische Anfragen

Hierarchische Anfragen in Datenbanken

- Beispiele für rekursive Datenmodelle
 - Familienbeziehungen (z.B. Vater, Sohn) oder Vorgesetztenbeziehung
 - Graphen, Bäume, rekursive Listen
 - Stücklisten in der Konstruktion und Fertigung
 - Fahrplanverbindungen
- Direkte Rekursion
 - Tabelle besitzt Fremdschlüsselbeziehung zu sich selbst
- Indirekte Rekursion
 - Rekursion durch zyklische Fremdschlüsselbeziehungen

Hierarchische Anfragen in Datenbanken

- Abfrage des Chefs von Herrn Abel
 - ```
SELECT p2.name
FROM pers p1, pers p2
WHERE p1.vnr = p2.pnr
AND p1.name = 'Abel';
```
- Abfrage der Chefs von Herrn Abel (Stufe 2)
  - ```
SELECT p2.name, p3.name
FROM pers p1, pers p2, pers p3
WHERE p1.vnr = p2.pnr
AND p2.vnr = p3.pnr
AND p1.name = 'Abel';
```
- Abfrage der Chefs einer Person über beliebige Stufen
 - Rekursive Abfrage

Hierarchische Anfragen durch Sichten in SQL99

- Wiederverwendung mehrfach verwendeteter Teilanfragen
 - Syntax: `WITH Viewname [Cols] AS (<query>)`
- Entspricht temporäre Sicht für Dauer der Anfrageausführung
- Hierarchische Anfragen
 - Aufspannung von Graphen durch Referenzen
- Aufbau hierarchische Anfragen
 - Initialisierung (initial subquery)
 - Rekursion (rekursive subquery)
 - UNION ALL (rekursive Ausführung)

Hierarchische Abfragen in SQL99

- SQL-Standard, lauffähig in Oracle
- Rekursion durch rekursive Verwendung einer Sicht
- Ermittlung aller direkten und indirekten Chef-Beziehungen

```
WITH Chefs (pnr, vnr) AS
  ((SELECT pnr, vnr
    FROM Pers
   WHERE name = 'Abel')

 UNION ALL

  (SELECT Pers.pnr, Pers.vnr
    FROM Chefs, Pers
   WHERE Chefs.vnr = Pers.pnr))

 SELECT pnr
   FROM Chefs;
```

} Initialisierung
(Sicht)

} Weitere Rekursions-
stufen

} Bildung Endergebnis

Hierarchische Abfragen in SQL99

- Ermittle alle Chefs von Hr. Abel

```
WITH Chefs (pnr, vnr) AS
((SELECT pnr, vnr
  FROM Pers
 WHERE name = 'Abel')

UNION ALL

(SELECT Pers.pnr, Pers.vnr
  FROM Chefs, Pers
 WHERE Chefs.vnr = Pers.pnr))

SELECT p.name
  FROM Chefs c, Pers p
 WHERE c.pnr = p.pnr;
```

Oracle-proprietäre Lösung hierarchischer Anfragen

- Oracle-proprietäre Lösung
- Ausgangspunkt der Rekursion durch
 - START WITH
- Verbindung von Eltern-Objekten zu Kindern durch
 - CONNECT BY ... PRIOR
- Beispielanfrage: Ermittle alle Chefs von Herrn Abel
 - ```
SELECT name
 FROM Pers
 CONNECT BY pnr = PRIOR vnr
START WITH name = 'Abel';
```
- Position von PRIOR bestimmt die Richtung der Rekursion
- Ohne START WITH wird Rekursion von allen Tupeln gestartet

# Hierarchische Anfragen in Oracle

- Regeln
  - Position von PRIOR relativ zu CONNECT-BY-Ausdruck bestimmt, welcher Ausdruck die Root und welcher die Zweige des Baums definiert
  - Eine WHERE-Klausel schließt einzelne Elemente des Baums aus, jedoch nicht deren Nachkommen
  - Eine CONNECT BY-Klausel schließt auch deren Nachkommen aus
  - Reihenfolge beim Einsatz von CONNECT BY:  
SELECT  
FROM  
WHERE  
CONNECT BY  
START WITH  
ORDER BY

# Hierarchische Anfragen

## Beispiel 20

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Ermittle alle Chefs von Herrn Abel, die mehr als 90.000 verdienen

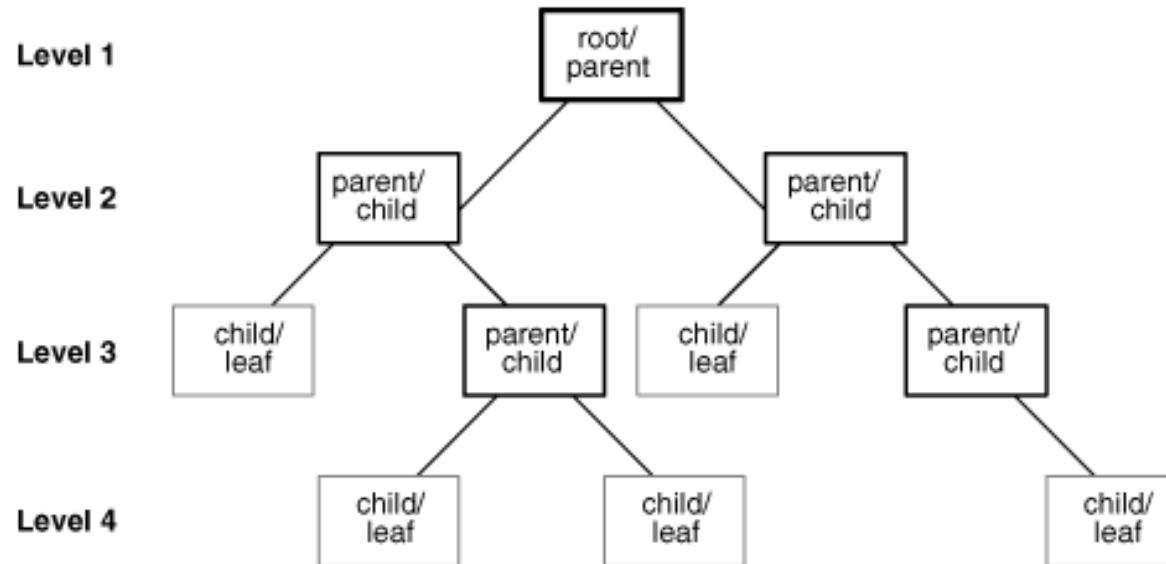
# Zyklen in hierarchischen Anfragen

- Zyklen
  - Falls Müller Vorgesetzter von Schmidt, Schmidt Vorgesetzter von Abel und Abel Vorgesetzter von Müller wäre, entsteht ein Zyklus
  - Abbruch in Anfrage mit Fehlermeldung "CONNECT BY loop in user data"
  - Fehler kann durch NOCYCLE verhindert werden

```
SELECT name
FROM pers
CONNECT BY NOCYCLE pnr = PRIOR vnr
START WITH name = 'Abel';
```

# Pseudospalten bei hierarchischen Anfragen

- Level
  - Tiefe, Level 1 für Wurzel
- CONNECT\_BY\_ISLEAF
  - Ausgabe 1 falls Knoten Blatt ist, ansonsten 0
- CONNECT\_BY\_ISCYCLE
  - Ausgabe 1 falls ein Child auch Vorfahre ist (Zirkelbezug), ansonsten 0



Quelle: docs.oracle.com

# Anwendung von LEVEL

- Suche nach allen Chefs von Abel
  - In bisheriger Lösung war Abel der Startpunkt und damit in der Lösungsmenge enthalten
  - Begrenzung der Lösungsmenge auf die Chefs von Abel durch Verwendung LEVEL

```
SELECT name
FROM Pers
WHERE LEVEL > 1
CONNECT BY NOCYCLE pnr = PRIOR vnr
START WITH name = 'Abel';
```

- Begrenzung Rekursionstiefe

```
...
CONNECT BY NOCYCLE pnr = PRIOR vnr AND level <= 8
START WITH name = 'Abel';
```

# Hierarchische Anfragen in Oracle

- Sortierung
  - ORDER SIBLINGS BY name
  - Sortierung der Child-Knoten
- Beispiel

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id
ORDER SIBLINGS BY last_name;
```

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id
ORDER SIBLINGS BY last_name;
```

| LAST_NAME | EMPLOYEE_ID | MANAGER_ID | LEVEL |
|-----------|-------------|------------|-------|
| King      | 100         |            | 1     |
| Cambrault | 148         | 100        | 2     |
| Bates     | 172         | 148        | 3     |
| Bloom     | 169         | 148        | 3     |
| Fox       | 170         | 148        | 3     |
| Kumar     | 173         | 148        | 3     |
| Ozer      | 168         | 148        | 3     |
| Smith     | 171         | 148        | 3     |
| De Haan   | 102         | 100        | 2     |
| Hunold    | 103         | 102        | 3     |
| Austin    | 105         | 103        | 4     |
| Ernst     | 104         | 103        | 4     |
| Lorentz   | 107         | 103        | 4     |
| Pataballa | 106         | 103        | 4     |
| Errazuriz | 147         | 100        | 2     |
| Ande      | 166         | 147        | 3     |
| Banda     | 167         | 147        | 3     |
| ...       |             |            |       |

Quelle: docs.oracle.com

# Hierarchische Anfragen

## Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

| Pers       |          |       |          |         |                  |            |            |
|------------|----------|-------|----------|---------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt  | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51        |            |
| 829        | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1997  |          | 80.000  | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...     | ...              | ...        | ...        |

| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | name        | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchhaltung | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

Wer verdient mehr als einer seiner direkten oder indirekten Chefs?  
Gebe Name des Mitarbeiters und des Chefs, sowie ihre Gehälter aus.

# 4.2 SQL-Anfragen 1

- 4.1 SQL – DDL, DML und DQL
- **4.2 SQL-Anfragen 1**
- 4.3 SQL-Anfragen 2
- 4.4 Programmiersprachen-Anbindung

# Datenbank-Anfragen (DQL)

- Formulierung von Anfragen durch SELECT-Statement
- Aufbau
  - SELECT     *Attribute*
  - FROM       *Tabellen*
  - WHERE      *Selektionsbedingung*

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Finde die Namen aller Angestellten.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Finde Namen und Gehalt der Angestellten in Abteilung K55.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Liste die verschiedenen Personalnummern der Vorgesetzten auf.

# Operatoren in SELECT-Anfragen

- Logische Operatoren
  - AND Und-Verknüpfung
  - OR Oder-Verknüpfung
  - NOT Verneinung
- Vergleichsoperatoren
  - =, !=, <, >, <=, >= Vergleich
  - IN, NOT IN Mengenvorgabe
  - BETWEEN x AND y Intervallvorgabe
  - LIKE p Maskierung
    - '%': beliebige Anzahl Zeichen
    - '\_': ein einziges Zeichen
  - IS NULL leerer Wert

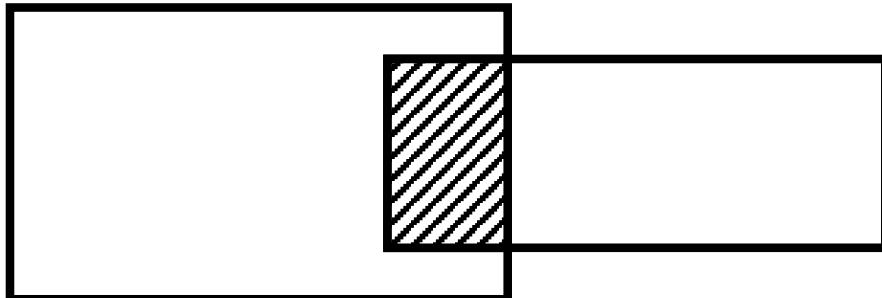
# Mengen-Operatoren in SELECT-Anfragen

- Mengen-Operatoren

- UNION
- INTERSECT
- MINUS

Vereinigungsmenge  
Schnittmenge  
„OHNE“-Menge

A INTERSECT B



SELECT ...  
FROM ...  
WHERE...

INTERSECT

SELECT ...  
FROM ...  
WHERE...

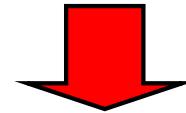
# Abfragen über mehrere Tabellen

- Verknüpfung zweier Tabellen durch Auflistung der Tabellen in der **FROM-Klausel**
- Verbindung der Tabellen in der **WHERE-Klausel**
- Hinzufügen des Tabellennamens, falls Attributname nicht mehr eindeutig
- Beispiel: Ermittle alle Mitarbeiter, die in der Abteilung 'Personal' arbeiten
  - ```
SELECT Pers.name
      FROM Pers, Abt
     WHERE Pers.anr = Abt.anr
       AND Abt.name = 'Personal';
```
- Reihenfolge der aufgelisteten Tabellen
 - *Keine Auswirkung auf Performance*
 - Reihenfolge sollte sich an Lesbarkeit orientieren

Abfragen über mehrere Tabellen

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg



$\bowtie_{\text{pers.anr} = \text{abt.anr}}(\text{pers}, \text{abt})$

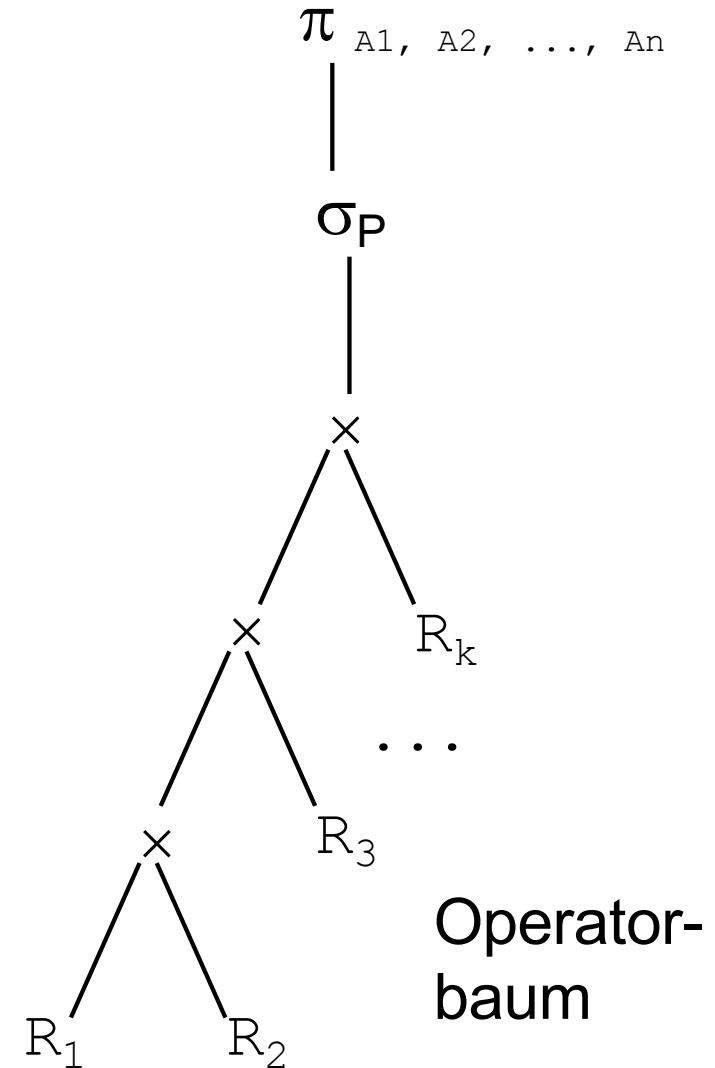
Pers										
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr	anr	name	ort
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123	K55	Personal	Nürnberg
123	Mueller	1980	01.09.00	88.000	Programmierer	K51		K51	Entwicklung	Erlangen
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123	K53	Buchhaltung	Nürnberg
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829	K55	Personal	Nürnberg
503	Junghans	1997		80.000	Programmierer	K51	123	K51	Entwicklung	Erlangen

Übersetzung SQL-Anfragen in Relationenalgebra

```
SELECT A1, A2, ..., An  
FROM R1, ..., Rk  
WHERE P;
```

entspricht

```
 $\pi_{A_1, \dots, A_n} (\sigma_P (R_1 \times \dots \times R_k))$ 
```



Datenbank-Anfragen

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Finde die Namen der Angestellten aus den Abteilungen K51, K54, K55.

Datenbank-Updates

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr, name, ort})

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Versetze alle Angestellten aus den Abteilungen K52 bis K54 nach Abteilung K51, wenn sie vor 1974 geboren sind.

Semantische Datenintegrität

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Programmierer arbeiten in Erlangen.

Datenbank-Anfragen

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Finde die Namen der Angestellten und die Namen der Abteilungen, denen sie angehören.

Geschachtelte Anfragen (nested queries)

- Eine SELECT-Anfrage kann eingeschlossene SELECT-Anfragen enthalten
- Anwendung, wenn eine Anfrage vom Suchergebnis einer anderen Anfrage abhängt
- Beispiel für eine einfache Unterabfragen in der WHERE-Klausel
 - Ermittle alle Mitarbeiter, die mehr als Herr Müller verdienen
- Probleme
 - Was passiert, wenn es mehrere Müller gibt?
 - Was passiert, wenn es keinen Müller gibt?

```
SELECT Pers.name  
FROM Pers  
WHERE gehalt > (SELECT gehalt FROM Pers  
                  WHERE name = 'Mueller');
```

Geschachtelte Anfragen mit ALL, ANY

- Operator ALL bzw. ANY
 - Unterabfrage mit ALL bedeutet, dass Bedingung für alle Elemente der Menge gelten muss
 - Unterabfrage mit ANY bedeutet, dass Bedingung für mindestens ein Element der Menge gelten muss
- Beispiel: Ermittle alle Mitarbeiter, die mehr als alle Mitarbeiter der Abteilung 'K51' verdienen

```
SELECT name
  FROM Pers
 WHERE gehalt > ALL (SELECT gehalt
                        FROM Pers
                       WHERE anr = 'K51');
```

- Anmerkung: Viele Anfragen lassen sich auf verschiedene Weise formulieren (z.B. sowohl über Joins als Unterabfragen)

Geschachtelte Anfragen mit IN und NOT IN

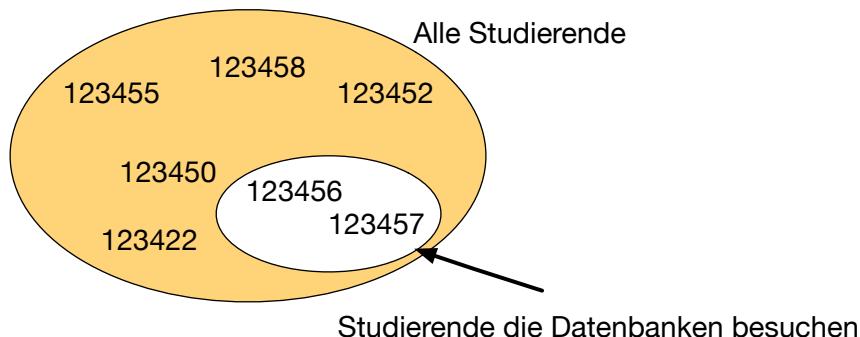
- Beispiel: Ermittle alle Studierende, die Datenbanksysteme 1 nicht besuchen
- Folgende Anfrage liefert ein falsches Ergebnis
 - ```
SELECT matrikelnummer
FROM BesuchtVorlesung
WHERE vorlesung != 'Datenbanksysteme 1';
```

| Studierende    |             |
|----------------|-------------|
| matrikelnummer | name        |
| 123456         | Tanja Meier |
| 123457         | Dirk Müller |
| 123458         | Franz Huber |
| 123459         | Gaby Pohl   |
| ...            | ...         |

| BesuchtVorlesung |                                 |
|------------------|---------------------------------|
| matrikelnummer   | vorlesung                       |
| 123456           | Algorithmen und Datenstrukturen |
| 123456           | Datenbanksysteme 1              |
| 123456           | Betriebssysteme                 |
| 123457           | Datenbanksysteme 1              |
| ...              | ...                             |

# Geschachtelte Anfragen mit IN und NOT IN

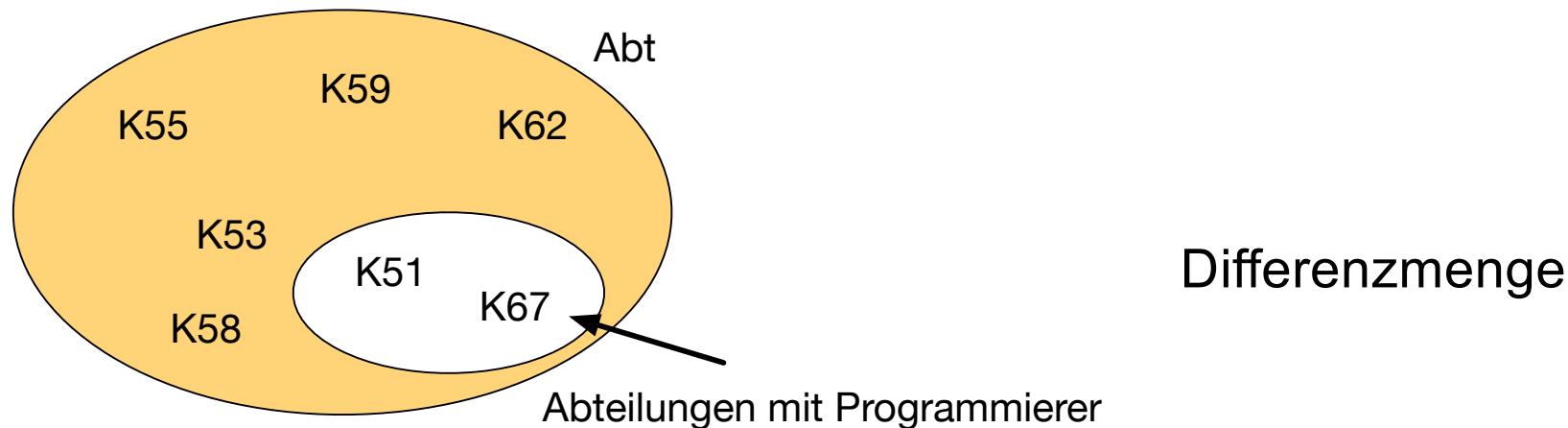
- Operator IN bzw. NOT IN
  - IN überprüft, ob ein Wert in einer Untertabelle enthalten ist
  - NOT IN überprüft, ob ein Wert nicht in Untertabelle enthalten ist
- Ermittle alle Studierende, die Datenbanksysteme 1 nicht besuchen
  - ```
SELECT matrikelnummer
  FROM Studierende s
 WHERE s.matrikelnummer NOT IN
       (SELECT b.matrikelnummer
        FROM BesuchtVorlesung b
        WHERE vorlesung = 'Datenbanksysteme 1');
```



Geschachtelte Anfragen mit IN und NOT IN

- Beispiel: Ermittle alle Abteilungen, in denen keine Programmierer arbeiten

```
- SELECT name  
  FROM Abt  
 WHERE anr NOT IN (SELECT anr  
                      FROM Pers  
                     WHERE beruf = 'Programmierer');
```



Geschachtelte Anfragen mit IN und NOT IN

- Alternative Lösung

- `SELECT name`

- `FROM Abt`

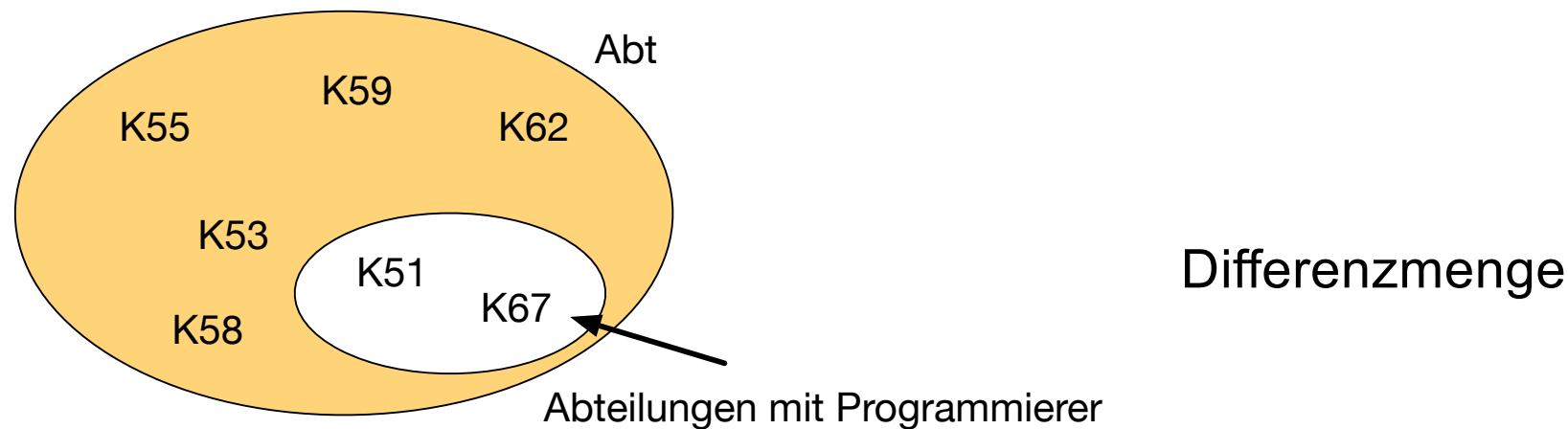
- `MINUS`

- `SELECT a.name`

- `FROM Pers p, Abt a`

- `WHERE p.anr = a.anr`

- `AND p.beruf= 'Programmierer';`



Geschachtelte Anfragen mit EXISTS

- Operator EXISTS
 - Operator EXISTS überprüft, ob ein aktueller Wert einer Abfrage existiert
- Beispiel: Ermittle alle Mitarbeiter, die nicht das Maximum verdienen
 - ```
SELECT Pers.name
FROM Pers p1
WHERE EXISTS (SELECT *
 FROM Pers p2
 WHERE p1.gehalt < p2.gehalt);
```
- Verwendung von Aliase
  - Damit in oberer Anfrage das Gehalt der betrachteten Person in die Subquery übergeben werden kann, muss ein Alias definiert werden
  - Aliase werden in der FROM-Klausel definiert

# Vermeidung korrelierter Unteranfragen

- Korrelierte Unterabfragen führen zu stark unperformanten Abfragen
- Beispiel zur Vermeidung korrelierter Unteranfragen
  - Anfrage: Welche Abteilungen sind nicht leer?

```
SELECT anr
FROM Abt a
WHERE EXISTS (SELECT NULL FROM Pers p
 WHERE a.anr = p.anr);
```

Korrelation zur Hauptanfrage

```
SELECT DISTINCT p.anr
FROM Pers p
```

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Welche Abteilungen sind leer?

# Unterabfragen in INSERT und UPDATE

- Unterabfragen in INSERT und UPDATE
  - Insert- und Update-Funktionen können ebenfalls Unterabfragen enthalten
  - Die Anweisung wird für alle ermittelten Werte durchgeführt
- Beispiel
  - Setze das Gehalt aller Mitarbeiter, die weniger verdienen als Herr Mueller auf das durchschnittliche Gehalt der Abteilung 'K53'

```
UPDATE Pers
SET gehalt = (SELECT AVG(gehalt)
 FROM Pers
 WHERE anr = 'K53')
WHERE gehalt < (SELECT gehalt
 FROM Pers
 WHERE name = 'Mueller');
```

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr, name, ort})

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Finde die Namen der Angestellten, die in Abteilungen in Erlangen arbeiten.

# Datenbank-Updates

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Lösche alle Angestellten, die für Abteilungen in Erlangen arbeiten.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Finde alle Angaben über diejenigen Angestellten, die im gleichen Jahr geboren sind und das gleiche Gehalt beziehen wie der Mitarbeiter Coy. Der Name Coy sei eindeutig.

# Funktionsaufrufe in SELECT-Anfragen

- Funktionsaufrufe in SQL-Statements

```
- SELECT UPPER(name), LOWER(beruf)
 FROM Pers;

- SELECT TO_CHAR(eindat, 'DD-MON-YY')
 FROM Pers;

- SELECT name, eindat
 FROM Pers
 WHERE MONTHS_BETWEEN(SYSDATE, eindat) > 24
```

# Funktionen in SELECT-Anfragen

- Aggregatfunktionen

|                           |                                    |
|---------------------------|------------------------------------|
| – MIN (spaltenname)       | Minimalwert                        |
| – MAX (spaltenname)       | Maximalwert                        |
| – COUNT (*)               | Anzahl der vorhandenen Zeilen      |
| – COUNT (spname)          | Anzahl der Zeilen ungleich NULL    |
| – COUNT (DISTINCT spname) | Anzahl Zeilen mit untersch. Werten |
| – SUM (spaltenname)       | Summe                              |
| – AVG (spaltenname)       | Mittelwert                         |

- Rechenoperationen

– +, -, \*, /

Addition, Subtraktion, Multiplikation,  
Division

- Ausgabe-Reihenfolge

– ORDER BY spname ASC/DESC Sortierung

# Weitere wichtige Funktionen in SQL

- COALESCE, NVL
  - Ersatzwert für Null-Werte, verhindert damit die Ausgabe NULL
  - COALESCE (gehalt, 0)
  - COALESCE (beruf, 'unbekannt')
  - Short Circuit:
    - Zweiter Parameter wird nur berechnet, wenn notwendig
- ROUND
  - Runden von Zahlen
  - Zweiter Parameter gibt Anzahl Nachkommastellen an
  - Beispiel Berechnung MwSt:
    - ROUND (20\*0.19, 2)

# Datum und Zeit in Oracle

- Komplexe Behandlung, da kulturspezifische Belange berücksichtigt werden müssen, Zeitzonen, etc.
- Datum und Datumsarithmetik in Oracle
  - DATE speichert Tag, Monat und Jahr, aber aus historischen Gründen auch Stunden, Minuten, Sekunden
  - Addition eines Datums mit 1 ergibt Datum mit dem nächsten Tag
  - Subtraktion eines Datums von einem anderen ergibt Anzahl Tage
  - SYSDATE liefert aktuelles Datum  
`SELECT SYSDATE FROM DUAL;`
- Timestamp
  - Für feinere Zeitintervalle als 1 Sekunde
  - `SELECT SysTimeStamp FROM DUAL;`
  - Weiterer Datentyp: TIMESTAMP WITH TIME ZONE

# Datum und Zeit in Oracle

## Formatierungsmöglichkeiten

- Ein Datum wird nie so ausgegeben, wie es intern gespeichert wird
  - Umwandlung in eine von Menschen lesbare Form
- Ein- und Ausgabe mit Hilfe von Umwandlungsfunktionen und Datumsformaten
  - MM Monat
  - MON Dreistellige Angabe für Monatsname: AUG
  - DD Tag
  - YYYY Jahr
  - YY Jahr zweistellig
  - HH Stunden
  - MI Minuten
  - SS Sekunden

# Datum und Zeit in Oracle

## Formatierungsmöglichkeiten

- Umwandlungsfunktionen
  - Umwandlung Datum → String: TO\_CHAR
  - Umwandlung String → Datum: TO\_DATE
- TO\_CHAR-Beispiel: Ausgabe der Uhrzeit
  - Format: TO\_CHAR(*string*[, '*format*'])
  - SELECT TO\_CHAR(SysDate, 'HH:MI:SS') FROM DUAL
  - Vorsicht: MM enthält Monat !
- TO\_DATE-Beispiel: Umrechnung Benutzereingabe in Datum
  - Format: TO\_DATE(*string*[, '*format*'])
  - Beispiel: TO\_DATE('11.05.2021', 'MM.DD.YYYY')

# Datum und Zeit in Oracle

## Formatierungsmöglichkeiten

- Festlegen des Datumsformats für eine komplette Connection
  - `ALTER SESSION SET nls_date_format = 'DD.MM.YYYY';`
  - Vorsicht: Das Datumformat ist damit nur innerhalb einer Session festgelegt
  - Nach disconnect oder exit von sqldeveloper oder Neustart eines Programms muss der Befehl wiederholt werden
- Abfrage des Jahrs in einem Datum
  - `... WHERE TO_CHAR(eintrittsdatum, 'YY') = '19' ;`
  - Oder noch einfacher (ähnlich: day, month, hour, minute, second):
  - `... where extract(year from eindat) = 1980;`
  - Extract sollte verwendet werden, wenn man mit Wert rechnen will, oder wenn nach dem Jahr sortiert werden soll

# Rechnen mit Datum Oracle

- Rechnen mit Anzahl Tagen
  - Addition/Subtraktion einer Anzahl Tagen zu einem Datum
  - `SELECT SYSDATE + 1 FROM DUAL;`
  - Berechnung der Anzahl Tage zwischen zwei Datumsangaben
  - `SELECT SYSDATE - EINDAT FROM PERS;`
- Berechnung Zeitintervallen (**INTERVAL**)
  - `SELECT MONTHS_BETWEEN(SYSDATE, EINDAT) FROM PERS;`
  - `SELECT SYSDATE + INTERVAL '1' YEAR FROM DUAL;`
  - `SELECT SYSDATE + INTERVAL '2' MONTH FROM DUAL;`
  - Für zahlreiche weitere Funktionen siehe Handbuch

# Rechnen mit Datum Oracle

- Rundungsfunktionen
  - Date beinhaltet auch die Zeit
  - Zwei Datumsangaben von aufeinanderfolgenden Tagen könnten sich daher um nur eine Minute unterscheiden
  - „Abschneiden“ der Zeit
  - `SELECT trunc(SYSDATE) FROM DUAL;`

# String-Funktionen in Oracle

## Eine Auswahl

|        |                                                                                                          |
|--------|----------------------------------------------------------------------------------------------------------|
|        | verkettet zwei Strings                                                                                   |
| ASCII  | liefert für den ersten Buchstaben des Strings die dezimale Darstellung in Datenbankzeichensatz           |
| CONCAT | verkettet zwei Strings (genau wie   )                                                                    |
| INSTR  | findet den Standort eines Zeichens in einem String                                                       |
| LENGTH | gibt die Länge des Strings aus                                                                           |
| LOWER  | wandelt alle Buchstaben in einem String in Kleinbuchstaben aus                                           |
| LPAD   | bringt einen String auf eine best. Länge, indem auf der linken Seite eine Zeichenfolge hinzugefügt wird  |
| LTRIM  | schnidet auf der linken Seite eines Strings Zeichen ab                                                   |
| RPAD   | bringt einen String auf eine best. Länge, indem auf der rechten Seite eine Zeichenfolge hinzugefügt wird |
| RTRIM  | schnidet auf der rechten Seite eines Strings Zeichen ab                                                  |
| SUBSTR | schnidet aus einer Zeichenkette einen bestimmten Ausschnitt aus                                          |
| TRIM   | schnidet alle Vorkommen eines oder mehrerer Zeichen auf der linken und rechten Seite eines Strings ab    |
| UPPER  | wandelt alle Buchstaben in einer Zeichenkette in Großbuchstaben um                                       |

# Numerische Funktionen in Oracle

## Eine Auswahl

|                             |                                                                    |
|-----------------------------|--------------------------------------------------------------------|
| ABS(value)                  | Absoluter Wert                                                     |
| CEIL(value)                 | kleinste Integer, der größer oder gleich value ist                 |
| EXP(value)                  | Exponent e von value                                               |
| FLOOR(value)                | Größter Integer, der kleiner oder gleich value ist                 |
| LN(value)                   | natürlicher Logarithmus von value                                  |
| LOG(value)                  | Logarithmus von value zur Basis 10                                 |
| MOD(value, divisor)         | Modulo                                                             |
| COALESCE(value, substitute) | Ersatz für value, wenn value gleich NULL ist                       |
| POWER(value, exp)           | Exponentialfunktion                                                |
| ROUND(value, prec)          | Rundung von value auf prec (Nachkommastellen)                      |
| SIGN                        | 1 wenn value positiv, -1 wenn value negativ, 0 wenn value NULL ist |
| AVG(value)                  | Durchschnittsfunktion                                              |
| COUNT(value)                | Anzahl Zeilen in einer Spalte                                      |
| MAX(value)                  | Maximum                                                            |
| MIN(value)                  | Minimum                                                            |
| SUM(value)                  | Summe aller Werte                                                  |
| VARIANCE(value)             | Varianz aller Werte                                                |

# Datums-Funktionen in Oracle

## Eine Auswahl

|                                 |                                                                              |
|---------------------------------|------------------------------------------------------------------------------|
| ADD_MONTHS                      | addiert eine Anzahl Monate zu einem Datum                                    |
| CURRENT_DATE                    | liefert das aktuelle Datum für die Zeitzone der Sitzung                      |
| CURRENT_TIMESTAMP               | liefert den aktuellen Zeitstempel mit der aktiven Zeitzoneninformation       |
| EXTRACT(timeunit FROM datetime) | extrahiert einen Teil eines Datums aus einem Datumswert                      |
| GREATEST(date1, date2,...)      | sucht aus einer Datumsliste das letzte (aktuellste) Datum heraus             |
| LEAST(date1, date2, ...)        | sucht aus einer Datumsliste das erste Datum heraus                           |
| LAST_DAY(date)                  | liefert das Datum des letzten Tags des Monats zurück, in den das Datum fällt |
| LOCALTIMESTAMP                  | liefert den lokalen Zeitstempel in der aktiven Zeitzone                      |
| MONTHS_BETWEEN(date2, date1)    | gibt das Ergebnis von date2-date1 als Monatsangabe zurück                    |
| NEW_TIME(date,'this','other')   | gibt das Datum und die Zeit in einer anderen Zeitzone zurück                 |
| ROUND(date, 'format')           | rundet ein Datum                                                             |
| SYSTIMESTAMP                    | liefert das Systemdatum, inklusive der Sekundenbruchteile und Zeitzone       |
| SYSDATE                         | liefert das aktuelle Datum und die aktuelle Zeit                             |
| TO_CHAR(date,'format')          | formatiert das Datum nach den Vorgaben in format                             |

# Konvertierungs-Funktionen in Oracle

## Eine Auswahl

|            |                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------|
| BIN_TO_NUM | konvertiert einen Binärwert in sein numerisches Äquivalent                                      |
| CAST       | Castet einen Typ in einen anderen                                                               |
| CONVERT    | übersetzt eine Zeichenkette von einem nationalem Zeichensatz in einen anderen                   |
| DECODE     | decodiert Datentyp CHAR, VARCHAR2 oder NUMBER auf Basis der Werte verschiedener Zeichenketten   |
| TO_CHAR    | konvertiert den Datentyp NUMBER oder DATE, so dass er sich wie ein String verhält               |
| TO_DATE    | konvertiert die Datentypen NUMBER, CHAR oder VARCHAR2, so dass sie sich wie ein Datum verhalten |
| TO_NUMBER  | konvertiert den Datentyp CHAR oder VARCHAR2, so dass er sich wie eine Zahl verhält              |
| TRANSLATE  | übersetzt die Zeichen in einer Zeichenkette in verschiedene Zeichen                             |

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr, name, ort})

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Finde die Namen der Angestellten, die mehr verdienen als alle Mitarbeiter in Abteilung 'K55'.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr, name, ort})

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Finde die Namen der Angestellten, die mehr als ihre direkten Vorgesetzten verdienen. Gib auch die zugehörigen Namen der Vorgesetzten an.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr, name, ort})

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Finde die Namen aller Chefs.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Liste alle Angestellten auf, deren Name mit 'A' beginnen.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr, name, ort})

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Liste die Angestellten aus Abteilung K55 nach ihren Gehältern aufsteigend und ihren Namen absteigend sortiert auf.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers       |          |       |          |         |                  |            |            |
|------------|----------|-------|----------|---------|------------------|------------|------------|
| <u>pnr</u> | name     | jahrg | eindat   | gehalt  | beruf            | <u>anr</u> | <u>vnr</u> |
| 406        | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55        | 123        |
| 123        | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51        |            |
| 829        | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53        | 123        |
| 874        | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55        | 829        |
| 503        | Junghans | 1997  |          | 80.000  | Programmierer    | K51        | 123        |
| ...        | ...      | ...   | ...      | ...     | ...              | ...        | ...        |

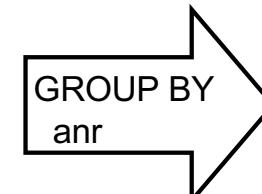
| Abt        |             |          |
|------------|-------------|----------|
| <u>anr</u> | name        | ort      |
| K51        | Entwicklung | Erlangen |
| K53        | Buchhaltung | Nürnberg |
| K55        | Personal    | Nürnberg |
| ...        | ...         | ...      |

- a) Liste alle Personen auf, die mehr als der Durchschnitt in der Firma verdienen.
- b) Liste alle Personen auf, die mehr als der Durchschnitt in ihrer jeweiligen Abteilung verdienen.

# Gruppenverarbeitung

- Gruppenverarbeiten (GROUP BY)
  - Zusammenfassen von Tupeln mit gleicher Eigenschaft
  - Beispiel: Welche Gehälter haben die Abteilungen zu zahlen?
    - ```
SELECT anr, SUM(gehalt) as summe
FROM Pers
GROUP BY anr;
```

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

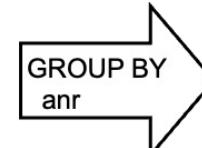


anr	summe
K51	168.000
K53	94.000
K55	182.000
...	...

Erlaubte Anfragen mit GROUP BY

- In SELECT-Klausel erlaubte Elemente
 - Konstante oder Funktion ohne Parameter (z.B. SysDate)
 - Gruppenfunktion, wie SUM, AVG, MIN, MAX, COUNT
 - Ein mit einem Ausdruck in der GROUP BY-Klausel übereinstimmender Wert
- Im Beispiel **nicht** erlaubt
 - `SELECT anr, name, SUM(gehalt) as summe
FROM Pers
GROUP BY anr;`

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...



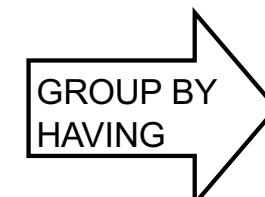
anr	summe
K51	168.000
K53	94.000
K55	182.000
...	...

Gruppenverarbeitung

Auswahl von Gruppen

- Auswahl von Gruppen (HAVING)
 - Definition von Bedingungen, die Gruppen erfüllen sollen
 - Beisp.: Welche Abteilungen zahlen mehr als 100.000 Euro Gehalt?
 - ```
SELECT anr, SUM(gehalt) as summe
FROM Pers
GROUP BY anr
HAVING SUM(gehalt) > 100000;
```

| Pers |          |       |          |         |                  |     |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |     |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |     |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |     |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |     |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |     |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... | ... |



| anr | summe   |
|-----|---------|
| K51 | 168.000 |
| K55 | 182.000 |
| ... | ...     |

# Auswertungsreihenfolge SELECT

FROM

Berechnung Ausgangstabelle

WHERE

Selektion von Tupel

GROUP BY

Gruppierung

HAVING

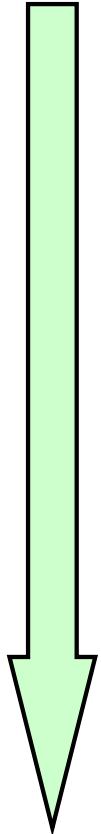
Selektion von Gruppen

SELECT

Projektion, Berechnung Aggregatfunktionen

ORDER BY

Sortierung der Tupel



# SELECT Syntax

## Übersicht

- Syntax einer SELECT-Anweisung

```
Select-expression ::=
 SELECT [ALL|DISTINCT] list-of-select-items
 FROM list-of-table-references
 [WHERE condition]
 [GROUP BY expr [, expr]...]
 [HAVING condition]
 [ORDER BY expr [ASC, DESC...]];
```

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Liste alle Abteilungsnummern und das Durchschnittsgehalt ihrer Angestellten auf (Monatsgehalt).

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Gruppiere die Angestellten nach Abteilungen und innerhalb der Abteilungen nach Berufen. Gib die Anzahl der Mitarbeiter und das durchschnittliche Monatsgehalt in jeder Gruppe an.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Liste alle Abteilungsnummern auf, die mindestens zwei Programmierer beschäftigen.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Wie viele verschiedene Berufe gibt es in Abteilung K55.

# Datenbank-Anfragen

## Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

| Pers |          |       |          |         |                  |     |     |
|------|----------|-------|----------|---------|------------------|-----|-----|
| pnr  | name     | jahrg | eindat   | gehalt  | beruf            | anr | vnr |
| 406  | Coy      | 1972  | 01.03.06 | 100.000 | Kaufmann         | K55 | 123 |
| 123  | Mueller  | 1980  | 01.09.00 | 88.000  | Programmierer    | K51 |     |
| 829  | Schmidt  | 1982  | 01.06.10 | 94.000  | Kaufmann         | K53 | 123 |
| 874  | Abel     |       | 01.05.14 | 82.000  | Softw.Entwickler | K55 | 829 |
| 503  | Junghans | 1997  |          | 80.000  | Programmierer    | K51 | 123 |
| ...  | ...      | ...   | ...      | ...     | ...              | ... | ... |

| Abt |             |          |
|-----|-------------|----------|
| anr | name        | ort      |
| K51 | Entwicklung | Erlangen |
| K53 | Buchhaltung | Nürnberg |
| K55 | Personal    | Nürnberg |
| ... | ...         | ...      |

Wie viele Personen arbeiten in den Abteilungen durchschnittlich?

# NULL-Werte

- Nullwerte
  - NULL kann für Attribute beliebigen Datentyps auftreten
  - NULL ist nicht identisch mit 0 oder ''
- Mögliche Bedeutungen von Nullwerten
  - Unbekannter Wert
  - Nicht existierender Wert
  - Bedeutungsloser Wert
- Anfrage mit unklarer Bedeutung
  - ```
SELECT COUNT(*)
  FROM Pers
 WHERE vnr IS NULL;
```

Berechnungen mit NULL-Werten

- Arithmetische Ausdrücke (z.B. Addition, Multiplikation)
 - Resultat ist NULL, wenn eine der Operanden NULL ist
 - Beispiele: $A + \text{NULL} = \text{NULL}$, $0 * \text{NULL} = \text{NULL}$
- Aggregatfunktionen (z.B. SUM, AVG)
 - Aggregatfunktionen ignorieren NULL in ihren Argumenten
- Vergleichsoperationen
 - Resultat ist NULL, wenn eine der Operanden NULL ist
 - Beispiele: $A > \text{NULL} \rightarrow \text{NULL}$
- Logische Ausdrücke ("dreiwertige Logik")

OR	true	NULL	false
true	true	true	true
NULL	true	NULL	NULL
false	true	NULL	false

AND	true	NULL	false
true	true	NULL	false
NULL	NULL	NULL	false
false	false	false	false

NOT	
true	false
NULL	NULL
false	true

SELECT-Anfragen mit NULL-Werten

- Abfrage von Nullwerten
 - IS NULL: Suche nach Elementen mit Nullwerten
 - IS NOT NULL: Suche nach Elementen, bei denen irgendwelche Daten (ungleich NULL) vorhanden sind
- In WHERE-Bedingung werden nur Tupel weitergereicht, die true sind
- In Gruppierungen wird NULL als eigener Wert erfasst und als eigene Gruppe gebildet
 - Resultat ist NULL, wenn eine der Operanden NULL ist
 - Gruppierung ohne NULL:
 - SELECT jahrg, AVG(gehalt)
 - FROM Pers
 - WHERE jahrg IS NOT NULL
 - GROUP BY jahrg;

SELECT-Anfragen mit NULL-Werten

- **VORSICHT:** SELECT-Anfragen können in Verbindung mit Nullwerten unerwartete Ergebnisse liefern !
 - "You can never trust the answers you get from a database with nulls" (C.J. Date: SQL and Relational Theory)

```
SELECT count(*)
FROM Pers
WHERE gehalt > 100000
      OR gehalt <= 100000;
```

```
SELECT SUM(gehalt)
FROM Pers;
```

- Mögliche Maßnahmen
 - Möglichst viele Attribute mit NOT NULL definieren
 - Outer-Joins vorsichtig anwenden

Datenbank-Anfragen

Wiederholung

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

-- Welche Mitarbeiter sind aus Konstanz oder Nürnberg?

```
SELECT name FROM Pers, Abt  
WHERE Pers.anr = Abt.anr  
AND ort = 'Konstanz' OR ort = 'Nürnberg';
```

Was ist denn hier falsch???

Ergebnis:

NAME
1 Mueller
2 Coy
3 Schmidt
4 Abel
5 Junghans
6 Mueller
7 Coy
8 Schmidt
9 Abel
10 Junghans

Datenbank-Anfragen

Wiederholung

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

Pers								
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr	
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123	
123	Mueller	1980	01.09.00	88.000	Programmierer	K51		
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123	
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829	
503	Junghans	1997		80.000	Programmierer	K51	123	
...	

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

- 1: Ermittle die Menge an Gehälter, die Mitarbeiter in Nürnberg beziehen
- 2: An welchen Orten sind Programmierer beschäftigt? An welchen Orten keine?
- 3: Erhöhe das Gehalt von allen Programmierer, die seit 30 Jahren in der Firma sind, um 10%
- 4: Liste alle Personen auf, die einen Chef haben, der jünger ist als sie selbst
- 5: Wie viel Mitarbeiter hat Herr Mueller?
- 6: Stelle alle Abteilungen zusammen, die mehr als zehn Programmierer haben.
- 7: Welche Mitarbeiternamen kommen mehrmals vor?

Datenbank-Anfragen

Wiederholung

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Welcher Chef hat Mitarbeiter, die nicht in seiner Abteilung arbeiten?

4.1 SQL – DDL und DML

- 4.1 SQL – DDL und DML
 - Einführung
 - Datendefinition (DDL)
 - Datenbankmodifikation (DML)
 - Zugriffsüberwachung
- 4.2 SQL-Anfragen 1
- 4.3 SQL-Anfragen 2
- 4.4 Programmiersprachen-Anbindung

Literaturempfehlung

- G. Kuhlmann, F. Müllmerstadt: SQL – Der Schlüssel zu relationalen Datenbanken, Rowohlt Taschenbuch Verlag, 2004
- K.E. Kline: SQL in a nutshell – A Desktop Quick Reference, O'Reilly, 3. Auflage, 2009
- B. Brumm: Beginning Oracle SQL for Oracle Database 18c, Apress, 2019
als E-Book in HTWG-Bibliothek
- L. de Haan, D. Fink, T. Gorman, I. Jørgensen, K. Morton: Beginning Oracle SQL, Apress, 2009
als E-Book in HTWG-Bibliothek
- L. de Haan: Mastering Oracle SQL and SQL*Plus, Apress, 2005
als E-Book in HTWG-Bibliothek
- Oracle 19c SQL Language Reference
 - <http://oracle19c.in.hwg-konstanz.de/sqlrf/index.html>



Structured Query Language (SQL)

Einführung

- Entwicklung zu Beginn der 70er Jahre, Standardisierung durch ANSI
- 1986 SQL-86, Alias: SQL0
- 1989 SQL-89, Alias: SQL1
- 1992 SQL-92, Alias: SQL2
- 1999 SQL-99, Alias: SQL3
 - Objekt-relationale Erweiterungen
 - Aktive Regeln, etc.
- 2003 SQL:2003, Alias: SQL4
 - SQL/XML: XML-Datentyp, Zugriff auf XML-Daten
 - SQL/MED: Management of External Data
- 2008 SQL:2008
- 20nn: ...

SQL Eigenschaften

- Nichtprozedurale, deskriptive Sprache
 - Fragesteller gibt Frage, aber keinen Algorithmus zur Lösung vor
 - Gegensatz 3GL: Java, C, PASCAL, etc.

1. Generation: Maschinensprache
2. Generation: Assembler
3. Generation: Problemorientierte Sprachen (Java, C, C++, Basic)
4. Generation: Nichtprozedurale Sprachen (SQL)

SQL als nichtprozedurale Sprache

Problemorientierte Sprache (3GL)	SQL (4GL)
<pre>open(buecher); while (!EOF (buecher)) { read(buch); if (buch.leihfrist > 0) if (buch.schlagwort = 'SQL') print(buch.autor, buch.titel); } close(buecher);</pre>	<pre>SELECT autor, titel FROM buecher WHERE leihfrist > 0 AND schlagwort = 'SQL';</pre>

4.1 SQL – DDL und DML

- 4.1 SQL – DDL und DML
 - Einführung
 - Datendefinition (DDL)
 - Datenbankmodifikation (DML)
 - Zugriffsüberwachung
- 4.2 SQL-Anfragen 1
- 4.3 SQL-Anfragen 2
- 4.4 Programmiersprachen-Anbindung

Übersicht SQL

- DDL (Data Definition Language)
 - Datendefinitionssprache zur Beschreibung von
 - Relationen
 - Attribute
 - Wertebereiche
 - Schlüssel
 - Constraints
- DML (Data Manipulation Language)
 - Datenmanipulationssprache
 - Mengenorientierte Operationen
- DQL (Data Query Language)
 - Datenabfragesprache
 - Mengenorientierte Operationen

Physischer Entwurf

- Entwurfsschritt physischer Entwurf
 - "Einrichten" des Datenbankschemas mit Hilfe der Datendefinitionssprache des gewählten Systems
 - Vergabe von Zugriffsrechten
 - Definition von Indexstrukturen für effizienten Datenzugriff
 - Gruppierung von Blöcken zu Cluster
 - Festlegung Datei-Formate

Hinweise zur Sprache SQL

- SQL ist nicht case-sensitiv
 - Keine Unterscheidung zwischen Groß- und Kleinschreibung
 - Verwendung von Namenskonventionen
 - `select name from kunde;`
 - `SELECT NAME FROM KUNDE;`
 - `sELeCT namE frOM kunDE;`
- Zeichenketten in Hochkommata sind case-sensitiv
 - `'Konstanz' ≠ 'KONSTANZ'`
- Attributwerte in Oracle
 - Zahlen: 100 bzw. 9.55
 - Zeichenketten: 'Name'
 - Datum: '10-MAR-15'

Datendefinition (DDL)

- Erzeugen einer neuen Datenbank
 - `CREATE DATABASE databasename;`
- Erzeugen neuer Tabellen (Relationen)
 - `CREATE TABLE tabellename
(spaltenname datentyp [NOT NULL] [DEFAULT deftyp]
, spaltenname datentyp [, ...])`
- Festlegung von Primär- und Fremdschlüsseln
 - `PRIMARY KEY (spaltenliste)`
 - `FOREIGN KEY (spaltenliste)
REFERENCES tabellename (spaltenliste)`
- Modifizieren von Tabellen
 - `ALTER TABLE tabellename;`
- Löschen von Tabellen
 - `DROP TABLE tabellename;`

Datendefinition (DDL)

Beispiel

```
CREATE TABLE Studium
( studienfachNr      INT,
  studienfach        VARCHAR(20),
  anzahlSemester    INT,
  abschluss         VARCHAR(20),
  CONSTRAINT Studium_pk PRIMARY KEY(studienfachNr) );
```

```
CREATE TABLE Student
( matrikelNr        INT,
  name              VARCHAR(20),
  studienfachNr     INT,
  semester          INT,
  studienort         VARCHAR(20),
  CONSTRAINT Student_pk PRIMARY KEY(matrikelNr),
  CONSTRAINT Student_fk FOREIGN KEY (studienfachNr)
    REFERENCES Studium(studienfachNr)
    ON DELETE SET NULL );
```

Datentypen in SQL

CHAR (n)	Zeichenkette der Länge n
VARCHAR (n) in Oracle: VARCHAR2 (n)	Zeichenkette variabler Länge mit maximaler Länge n
INTEGER oder INT	Ganzzahl
NUMERIC (i, n) in Oracle: NUMBER (i, n)	Dezimalzahl mit i Insgesamt-Stellen und n Nachkommastellen, Darstellung exakter numerischer Werte
FLOAT	Binäre Gleitkommazahl: Darstellung annähernder numerischer Werte
DATE	Datum
CLOB, BLOB, NCLOB	große Objekte

- Es gibt in Oracle keinen Datentyp BOOLEAN
 - Ersatz:

```
flag char(1) check (flag in ( 'Y', 'N' ))
```

Beispiel Datendefinition

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Erzeuge eine neue Tabelle Abt für Abteilung.

Beispiel Datendefinition

Abt = ({anr, name, ort})

Erzeuge eine neue Tabelle Abt für Abteilung.

Beispiel Datendefinition

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Erzeuge eine neue Tabelle Pers für Personal.

Beispiel Datendefinition

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Erzeuge eine neue Tabelle Pers für Personal.

Schema-Modifikation

- Löschen und Ändern von Spalten
 - `ALTER TABLE table_name ADD column_name datatype;`
 - `ALTER TABLE table_name DROP column_name;`
 - `ALTER TABLE table_name MODIFY column_name datatype;`
- Nachträgliches Hinzufügen oder Entfernen von Constraints
 - Reihenfolge der Tabellendefinition beliebig
 - Rekursive Fremdschlüssel-Beziehungen möglich

```
ALTER TABLE PERS ADD (
CONSTRAINT fk_abt2 FOREIGN KEY(anr) REFERENCES Abt(anr)
);

ALTER TABLE PERS DROP CONSTRAINT fk_abt2;
```

Datenintegrität

- Statische Integritätsbedingung (Zustandsbedingungen)
 - Bedingung, die von jedem Zustand der Datenbank erfüllt werden muss
 - Beispiele:
 - Kilometerzahl eines Autos ≥ 0
 - Professoren haben Besoldungsgruppe C2, C3, C4, W2, W3, W4
 - Definition in SQL: CHECK
- Dynamische Integritätsbedingung (Übergangsbedingungen)
 - Beispiele:
 - Kilometerzahl eines Autos kann nur größer werden
 - Professoren können nur befördert, nicht degradiert werden
 - Definition in SQL: Trigger

Datenintegrität

- Spaltenbedingungen (column constraints)
 - Spezialfall einer Tabellenbedingung
 - Definition in Spalten- oder Tabellendefinition
 - Wertebereichseinschränkungen (domain constraints)
 - Bsp: Gehalt > 20.000
- Tabellenbedingungen (table constraints)
 - Integritätsbedingungen über der betreffenden Tabelle
 - Definition in Tabellendefinition
 - Bsp: Gehalt > 30.000 bei 20-jähriger Firmenzugehörigkeit
- General Constraint
 - Integritätsbedingungen über beliebige Spalten beliebiger Tabellen
 - Syntax: CREATE ASSERTION
 - In SQL-Standard vorgesehen, nicht in allen DBMS realisiert
 - Bsp: Programmierer arbeiten in Erlangen

Spalten- und Tabellenbedingungen

- PRIMARY KEY
 - Definition einer oder mehrerer Spalten als Primärschlüssel
- UNIQUE
 - Verhindern, dass in einer Spalte doppelte Werte eingegeben werden
- DEFAULT
 - Definition eines Default-Werts
- NOT NULL
 - Kein NULL-Wert erlaubt
- CHECK
 - Definition einer Bedingung, der jeder Datensatz genügen muss
 - Bereichsangaben CHECK ($a \geq 0$ AND $a < 10$)
 - Konkrete Werte CHECK (anrede IN ('Herr', 'Frau'))
 - Werterelationen CHECK (kundennr <= 4711)

Check-Bedingung

- Inline-Definition
 - Innerhalb der Attribut-Definition

```
CREATE TABLE Pers (
    ...
    gehalt integer NOT NULL CHECK (gehalt > 0),
    ...
)
```

Out-of-line Definition

- Ausserhalb Attributdefinition als Teil der Tabellendefinition

```
CREATE TABLE Pers (
    ...
    gehalt integer NOT NULL,
    ...
    CONSTRAINT PersGehalt CHECK (gehalt > 0),
    ...
);
```

Inline vs. Out-of-line Definition

- **Inline-Definition**
 - Mit oder ohne Constraintname möglich
 - NOT NULL kann nur inline definiert werden
- Vorteile bei Definition eines Constraintnamen
 - Verständlichere Fehlermeldungen
 - Einfacheres Löschen oder Modifizieren des Constraints

```
CREATE TABLE Pers(  
    ...  
    gehalt integer NOT NULL CHECK (gehalt > 0),  
    jahrg  integer CONSTRAINT persjahrg CHECK(jahrg > 1900),  
    ...  
) ;
```

Spezifikation von Constraints

- Beispiele für CHECK

```
CREATE TABLE Pers (
    pnr                INT NOT NULL,
    name               VARCHAR2(20) NOT NULL,
    jahrg              INT NOT NULL,
    eindat             DATE NOT NULL,
    gehalt              INT NOT NULL,
    beruf              VARCHAR2(20),
    anr                CHAR(3) NOT NULL,
    vnr                INT,
    CONSTRAINT pers_pk PRIMARY KEY(pnr),
    CONSTRAINT pers_fk FOREIGN KEY (anr) REFERENCES abt(anr),
    CONSTRAINT persjahrgang CHECK (jahrg > 1900
                                  AND jahrg < 2100),
    CONSTRAINT PersGehalt CHECK (gehalt > 0)
);
```

Überprüfung von Regular Expressions

- Regular Expressions zur Definition von Integritätsconstraints
 - Verwendung von REGEXP_LIKE in Oracle
 - Beispiel: HTWG-Telefonnummern sollen im folgenden Format definiert sein:
XXXXX-XXX-XXX
 - Beispielnummer: 07531-206-630

```
CREATE TABLE Person
(
    name          VARCHAR2(30),
    telnummer    VARCHAR2(30),
    CONSTRAINT person_telnummer_format
        CHECK ( REGEXP_LIKE
            ( telnummer, '^\\d{5}-\\d{3}-\\d{3}$' ) )
);
```

Semantische Datenintegrität - Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Es existieren nur die Orte Nürnberg, Erlangen und Konstanz.

Semantische Datenintegrität - Beispiel

Pers = ({pnr, name, jahrg, eindat, gehalt, beruf, anr, vnr})

Abt = ({anr, name, ort})

Das Minimalgehalt beträgt 20.000

Semantische Datenintegrität

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Das Minimalgehalt in Abteilung K55 beträgt 80.000

Semantische Datenintegrität

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

Das Minimalgehalt in Abteilung K55 beträgt 80.000

Fremdschlüsselbedingungen

(referentielle Integrität)

- Parent-Tabelle
 - Tabelle, die den Primärschlüssel enthält
- Dependent (Child) -Tabelle
 - Tabelle, die den Fremdschlüssel enthält
- Fremdschlüsselbedingung (referentielle Integrität)
 - Zu jedem von NULL verschiedenen Fremdschlüsselwert der Dependent-Tabelle existiert ein entsprechender Schlüsselwert der Parent-Tabelle
 - Es existieren somit keine "dangling references"

Pers								
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr	
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123	
123	Mueller	1980	01.09.00	88.000	Programmierer	K51		
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123	
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829	
503	Junghans	1997		80.000	Programmierer	K51	123	
...	

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Fremdschlüsselbedingungen

- Lösch- und Änderungsregeln
 - Aktionen zur Gewährleistung der referentiellen Integrität
 - Durchführung bei DELETE- und UPDATE
 - Syntax: CONSTRAINT pers-fk FOREIGN KEY (anr)
 REFERENCES abt(anr) ON DELETE SET NULL

ON DELETE RESTRICT	Verhindern von Löschungen (Default bei Oracle, keine Angaben notwendig)
ON DELETE CASCADE	Automatisches Löschen weiterer Sätze
ON DELETE SET NULL	Automatisches auf NULL setzen
ON DELETE SET DEFAULT	Automatisches auf Default setzen

Date Check Constraints

- Check-Befehl mit aktuellem Datum nicht möglich
 - Beispiel: keine Buchung darf in der Vergangenheit angelegt werden
 - Vergleich mit aktuellem Daten in Oracle nicht möglich
 - Constraint buchungsstart <= buchungsende ist möglich

```
CREATE TABLE Buchung (
    buchungsnr          VARCHAR2(30) PRIMARY KEY,
    buchungsstart        DATE NOT NULL,
    buchungsende         DATE NOT NULL,
    ...
    CONSTRAINT Buchung_in_Zukunft
        CHECK ( buchungsstart > SYSDATE )      // Fehler
) ;
```

Date Check Constraints

- Begründung
 - Begründung: SYSDATE ist indeterministische Funktion, deren Rückgabewert sich ständig ändert
 - Check-Output kann sich damit auch ständig ändern
 - Constraint soll aber eigentlich nur zum Speicher-Zeitpunkt gelten
 - Lösung: Trigger, siehe späteres Kapitel

```
CREATE TABLE Buchung (
    buchungsnr          VARCHAR2(30) PRIMARY KEY,
    buchungsstart        DATE NOT NULL,
    buchungsende         DATE NOT NULL,
    ...
    CONSTRAINT Buchung_in_Zukunft
        CHECK ( buchungsstart > SYSDATE ) // Fehler
) ;
```

4.1 SQL – DDL und DML

- 4.1 SQL – DDL und DML
 - Einführung
 - Datendefinition (DDL)
 - Datenbankmodifikation (DML)
 - Zugriffsüberwachung
- 4.2 SQL-Anfragen 1
- 4.3 SQL-Anfragen 2
- 4.4 Programmiersprachen-Anbindung

Datenbankmodifikation (DML)

- Einfügen von Tupeln (Insertion)
 - `INSERT INTO tabellename VALUES (...);`
 - `INSERT INTO tabellename SELECT-Anweisung;`
- Löschen von Tupeln
 - `DELETE FROM tabellename [WHERE bedingung];`
- Verändern von Tupeln (Update)
 - `UPDATE tabellename
SET spaltenname = wert
[, spaltenname = wert] [, ...]
WHERE bedingung;`

Datenbank-Updates

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = (anr, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Füge einen neuen Angestellten mit PNR = 007 und Namen Mayer in Abteilung K55 ein.

Datenbank-Updates

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Erhöhe das Gehalt von Herrn Abel auf 45.000 €.

Datenbank-Updates

Beispiel

Pers = ({pnr}, name, jahrg, eindat, gehalt, beruf, anr, vnr)

Abt = ({anr}, name, ort)

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

Abt		
anr	name	ort
K51	Entwicklung	Erlangen
K53	Buchhaltung	Nürnberg
K55	Personal	Nürnberg
...

Lösche den Angestellten mit PNR = 007.

Vergabe von Schlüsselwerten in Oracle

- Verwendung von Sequenzen
 - Automatische Vergabe von eindeutigen Zahlen

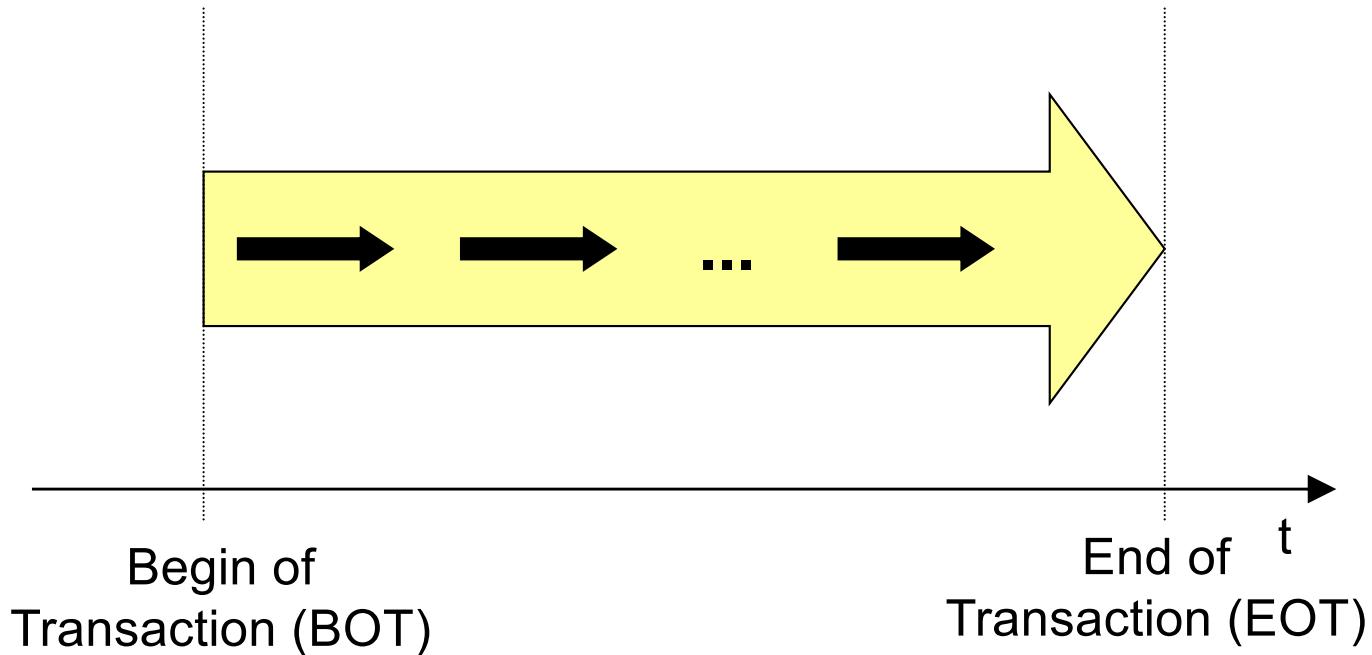
```
CREATE SEQUENCE persID INCREMENT BY 1 START WITH 100;  
INSERT INTO Pers(pnr, name, jahrg)  
VALUES (persID.NextVal, 'Jens Maier', 1960);
```

- Identity Columns
 - Neu in Oracle19c

```
CREATE TABLE T1 (c1 INT GENERATED BY DEFAULT  
                c2 VARCHAR(10));  
INSERT INTO T1(c2) VALUES ('Wert 1');
```

Definition Transaktion

Eine Transaktion ist eine ununterbrechbare Folge von DB-Operationen, die eine DB von einem konsistenten Zustand in einen (nicht notwendigerweise verschiedenen) konsistenten Zustand überführt.

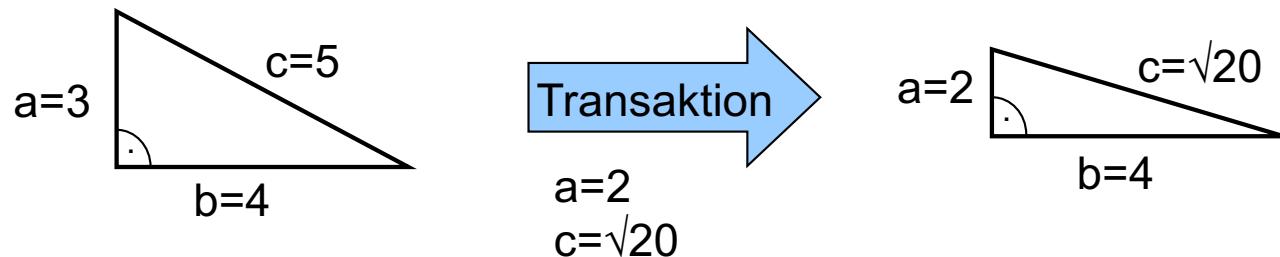


Datenbank-Transaktionen

- Commit
 - Alle Änderungen einer Transaktion werden permanent
 - Rollback ist anschließend nicht mehr möglich
 - Syntax: COMMIT; (äquivalent: COMMIT WORK;)
- Rollback
 - Einfügen, Aktualisieren und Löschen von Daten kann rückgängig gemacht werden
 - Wichtig bei Fehlern
 - Syntax: ROLLBACK;
- DDL-Statements bilden immer eine Transaktion und müssen nicht mit COMMIT abgeschlossen werden
 - CREATE TABLE...
 - DROP TABLE...

Arten von Konsistenz

- Unverzögerte Integritätsbedingungen
 - Überwachung direkt nach DB-Änderung, z.B. Wertebereichsgrenzen
 - Bsp: Mitarbeitergehalt > 0
- Verzögerte Integritätsbedingungen
 - Überwachung am Transaktionsende
 - Inkonsistente Zustände innerhalb der Transaktion erlaubt
 - Rechtwinkliges Dreieck: $a^2 + b^2 = c^2$



Überprüfung von Constraints in Oracle

```
CREATE TABLE Pers
( pnr          INTEGER PRIMARY KEY,
  name         VARCHAR2(20),
  jahrg        INTEGER,
  eindatum     DATE,
  gehalt       INTEGER,
  beruf        VARCHAR2(20),
  anr          CHAR(3),
  vnr          INTEGER,
  CONSTRAINT pers_fk
    FOREIGN KEY (anr) REFERENCES abt(anr) DEFERRABLE);

SET AUTOCOMMIT OFF;

SET CONSTRAINTS ALL DEFERRED;

INSERT INTO Pers
VALUES (444, 'test', 1980, '10-JAN-1990', 50000,
'Kaufmann', 'K66', NULL);

INSERT INTO abt VALUES ('K66', 'Entwicklung', 'Konstanz');

COMMIT;
```

Enabling and Disabling Constraints

- Constraints können auf deaktiviert werden
 - Beispiel: Constraints zur Überprüfung des Gehalts deaktivieren
 - Es können Gehälter mit Wert 0 gespeichert werden
 - Durch Aktivierung mit „ENABLE“ wird Constraint wieder aktiviert

```
ALTER TABLE Pers  
DISABLE CONSTRAINT persgehalt;
```

```
ALTER TABLE Pers  
ENABLE CONSTRAINT persgehalt;
```

4.1 SQL – DDL und DML

- 4.1 SQL – DDL und DML
 - Einführung
 - Datendefinition (DDL)
 - Datenbankmodifikation (DML)
 - Zugriffsüberwachung
- 4.2 SQL-Anfragen 1
- 4.3 SQL-Anfragen 2
- 4.4 Programmiersprachen-Anbindung

Sicherheit in Datenbanken

- Information ist ein wertvolles Gut
 - Vertraulichkeit (z.B. Gehalt)
 - Löschen aller Daten kann das Ende einer Firma bedeuten
 - Verhindern von Manipulationen (z.B. Gehalt)
- Unterschiedliches Schutzbedürfnis
 - Hochschule
 - Betrieb
 - Militärische Anlagen

Zugriffsschutz in SQL

- Zugriffsschutz
 - Kontrolliert Zugriffe von Subjekten auf Objekte
- Benutzerbestimmbare Zugriffskontrolle
 - Eigentümer-Prinzip: Jedes Objekt hat einen Eigentümer
 - Eigentümer ist zuständig für Schutz
 - Interne Kennung eines DB-Benutzer, z.B. User-ID
 - Rolle: Gruppe zusammenhängender Privilegien
- Identifikation und Authentisierung
 - Identifikation von Benutzern vor Zugang zu DBMS (Benutzername)
 - Authentifizierung meist durch Passwort
- Autorisierung und Zugriffskontrolle
 - Prinzip des kleinstmöglichen Privileges
 - Menge von Regeln, die die erlaubten Arten des Zugriffs bestimmt

Zugriffsüberwachung (Data Control Facility)

- Zugriffsberechtigungen
 - ALL, ALTER, DELETE, INDEX, INSERT, SELECT
- GRANT privilege-type
ON { table-Name | view-Name }
TO grantees
- Entziehen von Zugriffsrechten mit REVOKE
- Teilweise unterschiedliche Konzepte in versch. DBMS
- Privilege-types in Oracle
 - ALL – erlaubt alle Operationen
 - DELETE, INSERT, SELECT, UPDATE

Grant und Revoke

- Zugriffsrechte vergeben
 - GRANT INSERT, DELETE, SELECT, UPDATE
ON pers TO dbsys01;
 - GRANT UPDATE(name)
ON pers TO dbsys01;
 - GRANT SELECT ON sequencename TO dbsys01;
- Zugriffsrechte vergeben und Erlaubnis Rechte weiter zu geben
 - GRANT INSERT, DELETE, UPDATE, SELECT
ON pers TO dbsys01
WITH GRANT OPTION;
- Zugriffsrechte entziehen
 - REVOKE INSERT, DELETE, UPDATE
ON pers FROM dbsys01;

Zugriff auf andere Benutzertabellen

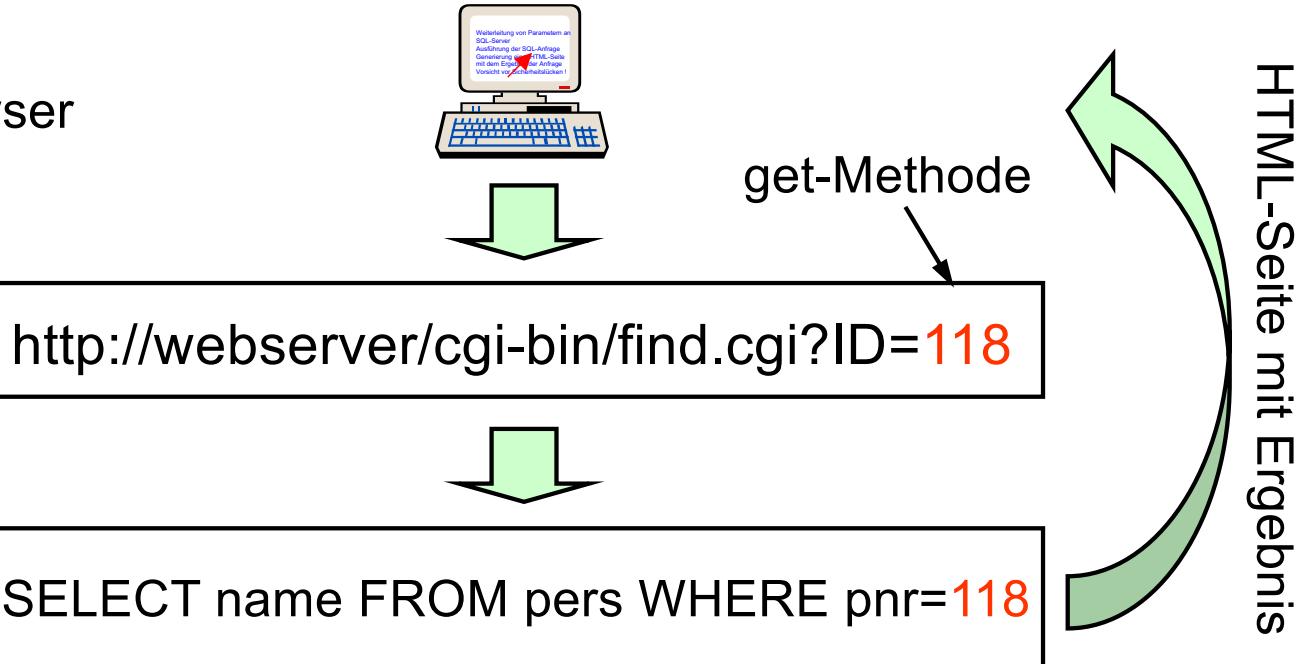
- Schema in Oracle
 - Datenbank kann mehrere Schemas haben
 - In jedem Schema sind alle Objekte, die ein bestimmter Nutzer der Datenbank erstellt hat
- Angabe des Schemanamens beim Zugriff auf Tabellen fremder Benutzer
 - ```
SELECT name, gehalt
FROM dbsys99.pers
WHERE jahrg > 1980;
```
- Alternative
  - ```
ALTER SESSION SET current_schema = dbsys27;
```

SQL-Injection in Webseiten

Benutzereingabe im Browser

Aufruf Website

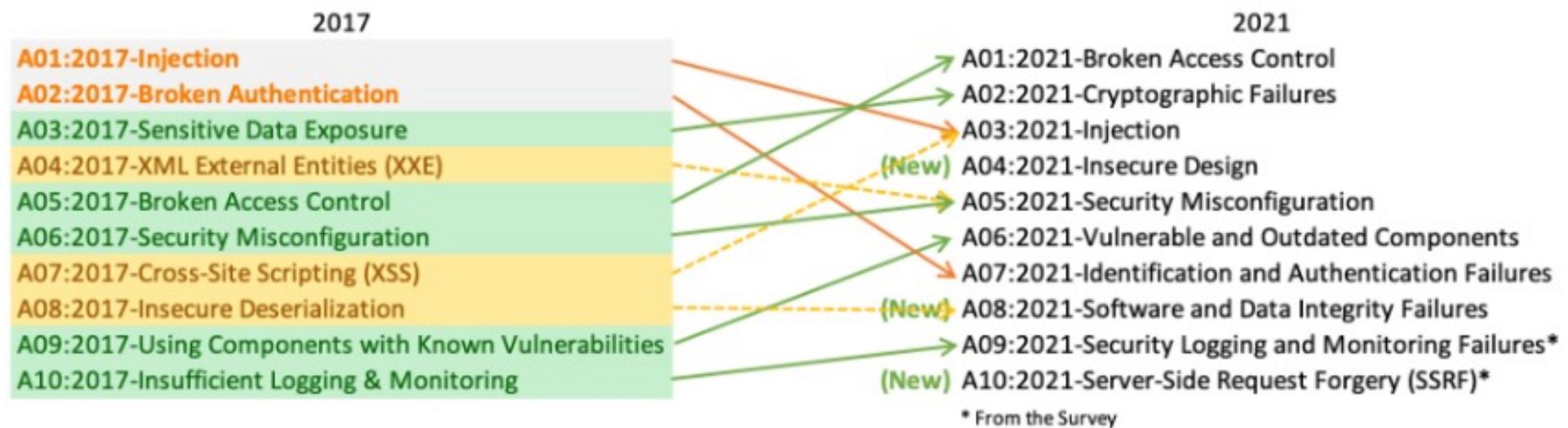
SQL-Anfrage im Server



- SQL-Anfragen in Webseiten
 - Weiterleitung von Parametern an SQL-Server
 - Ausführung der SQL-Anfrage
 - Generierung einer HTML-Seite mit dem Ergebnis der Anfrage
 - Vorsicht vor Sicherheitslücken, z.B. durch SQL-Injection !

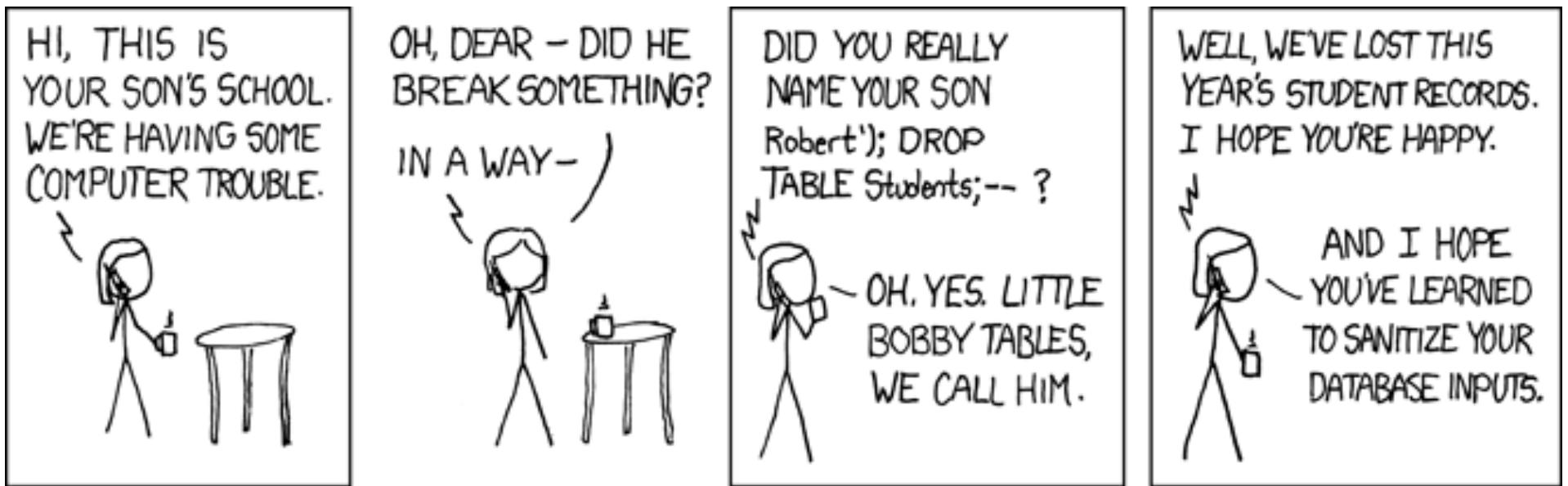
SQL-Injection

- Open Web Application Security Project (OWASP)
 - SQL-Injection ist eine der am meisten ausgenutzten Sicherheitslücken bei Webanwendungen
 - In Top Ten noch vor Authentifikation
 - Übernahme von ganzen Servern durch Betriebssystemschnittstellen



Quelle: <https://owasp.org/www-project-top-ten/>

Don't trust user input



Quelle: B. Karwin: SQL Antipatterns

Gegenmaßnahmen gegen SQL-Injections

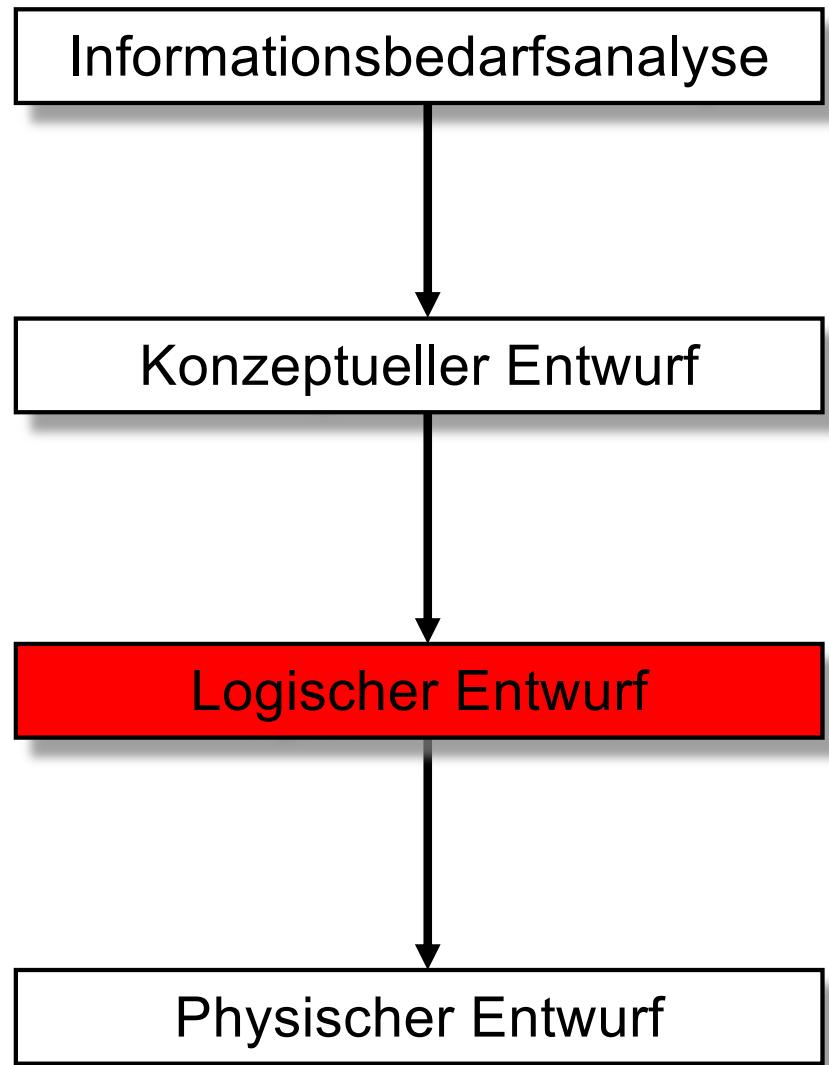
- Authentifikation, Privilegien zuweisen
 - Prinzip des kleinstmöglichen Privilegs
- Prüfung der Eingabedaten
 - Prüfung von Länge, Sonderzeichen, z.B. durch reguläre Ausdrücke
- Parametrisierte Kommandos, z.B. Prepared Statement in JDBC
 - Daten als Parameter an einen bereits kompilierten Befehl
 - Siehe letztes Kapitel

3. Logischer Datenbankentwurf

- Definition Relation
- Relationenschemata
- Transformation von ER-Modellen
- Normalisierungen
- Relationale Algebra

Datenbankentwurf

Entwurfsschritte



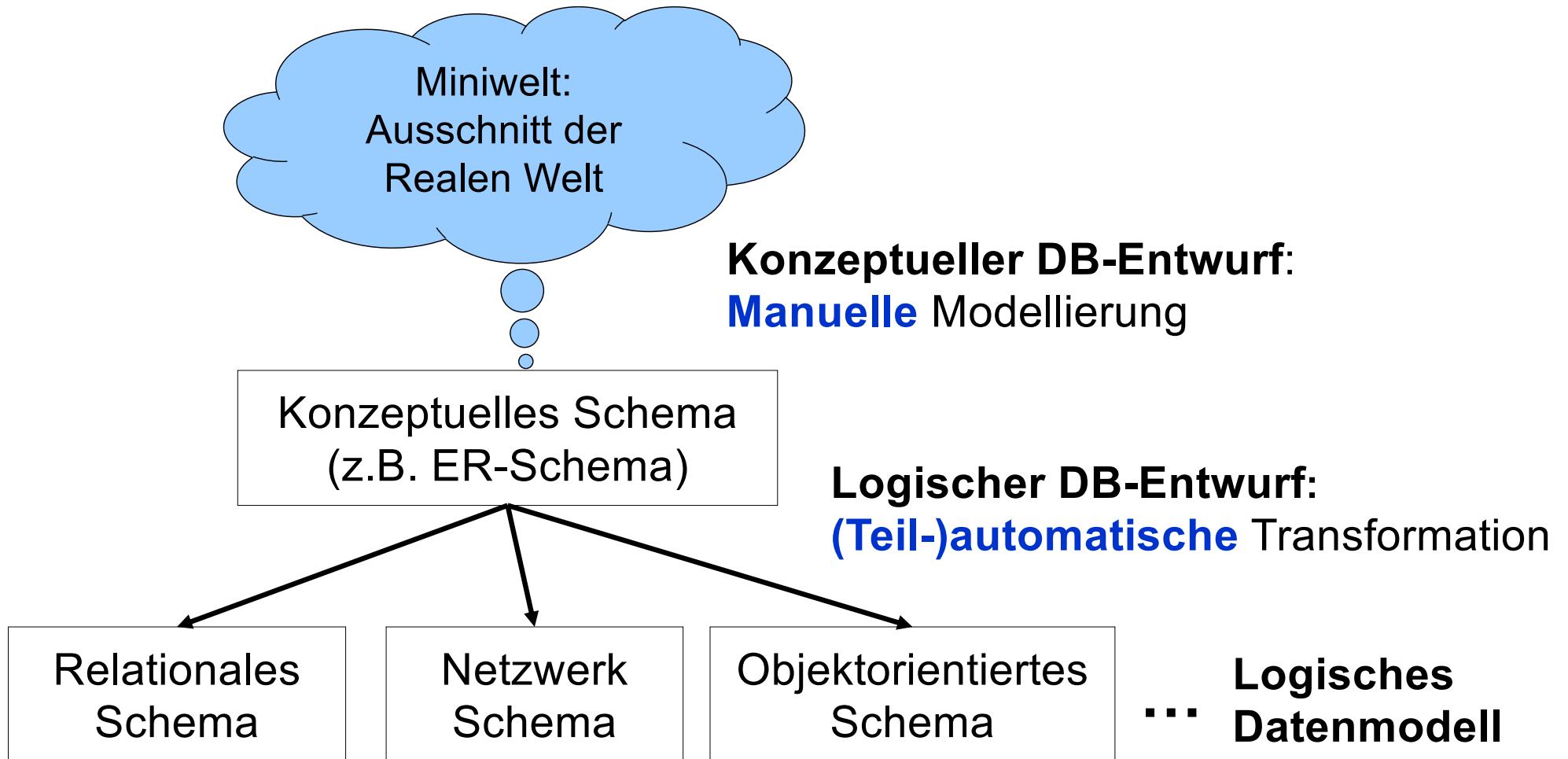
Sammlung aller für eine Miniwelt bedeutsamen Gegenstände, Eigenschaften, Beziehungen und Operationen

Präzise Beschreibung einer Miniwelt durch relationale oder objektorientierte Modelle

Abbildung auf ein rechnergestützt interpretierbares Schema, z.B. relationales Schema

Abbildung des logischen Datenbankschemas in eine effiziente physische Datenbasisstruktur

Logischer Datenbank-Entwurf



Das Relationenmodell

- Das Entity-Relationship-Modell von CHEN beschreibt die Miniwelt in einer sehr abstrakten Form
- Ziel: Beschreibung in "computerverständlicher" Form
- Bereitstellung einer Notation (Syntax) und eindeutigen Bedeutung (Semantik)
- Gebräuchliche Datenbankmodelle
 - Netzwerk-Datenbankmodell
 - Hierarchisches Datenbankmodell
 - Relationales Datenbankmodell
 - Objektorientiertes Datenbankmodell
 - XML-Datenmodell
 - Graph, JSON, etc.

Gliederung

- Strukturteil
 - Beschreibung von Objekttypen (Entity-Typen, Beziehungstypen) der Anwendungswelt
- Operationenteil
 - Bereitstellung von Operationen zur Anfrage oder Manipulation der diesen Objekttypen gehörenden Instanzen (Daten)

Definition Relation

A_1, A_2, \dots, A_n seien beliebige Mengen

1. Das kartesische Produkt $A_1 \times A_2 \times \dots \times A_n$ der Mengen A_i , $i=1, 2, \dots, n$ ist die Menge
$$A_1 \times A_2 \times \dots \times A_n := \{(a_1, a_2, \dots, a_n) \mid a_i \in A_i \text{ für } i = 1, 2, \dots, n\}$$
2. Eine Teilmenge $r \subseteq A_1 \times A_2 \times \dots \times A_n$ heißt (n -stellige) Relation über den Mengen (Attributen) A_1, A_2, \dots, A_n . N ist der Grad der Relation. Wir schreiben:
$$r(A_1, A_2, \dots, A_n)$$
3. Ein Element $t := (t_1, t_2, \dots, t_n) \in r$ wird als n -Tupel der Relation bezeichnet

Relationenschemata

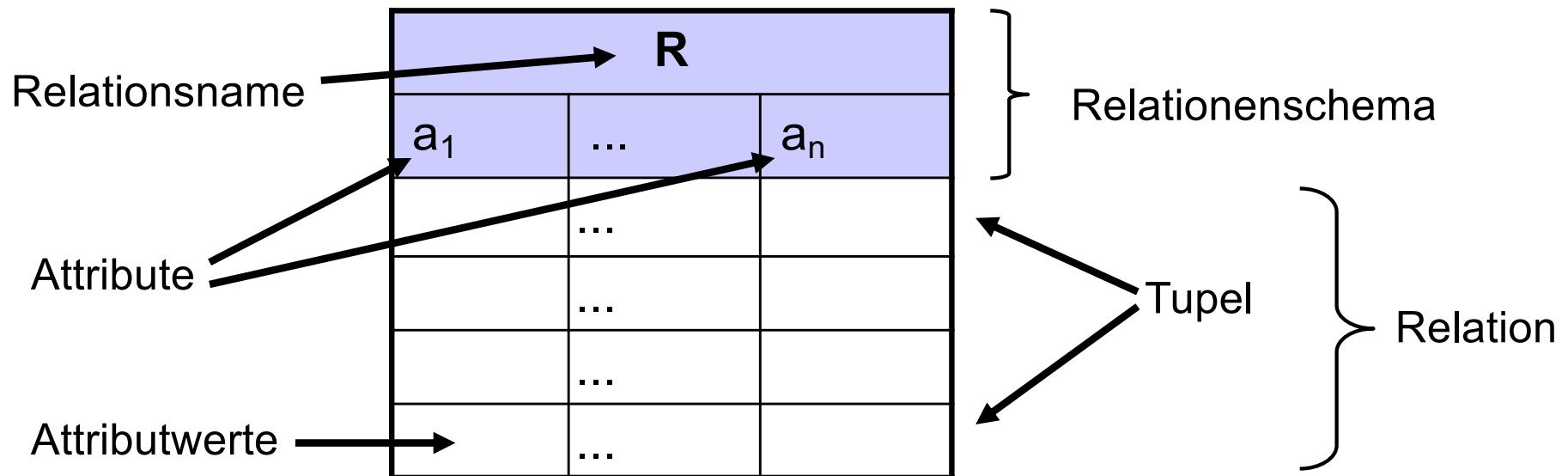
- Ein Relationenschema (Relationstyp) $R = (V, \Sigma)$ besteht aus
 - einem Namen R
 - einer Menge V von Attributen, $V = \{ a_1, a_2, \dots, a_n \}$
 - einer Menge Σ von Integritätsbedingungen (Constraints)
- Attributen werden Wertebereiche (Domains) zugeordnet
 - In der Praxis Standard-Datentypen wie z.B. INTEGER, STRING, DATE, ...
 - Beispiel: $\text{dom}(\text{NAME}) = \text{STRING}$
 - $\text{dom}(V) = \text{dom}(a_1) \times \text{dom}(a_2) \times \dots \times \text{dom}(a_n)$

Instanzen von Relationenschemata

- Eine **Relation** ist eine Instanz des zugehörigen Relationenschemas
 $R: (V, \Sigma) \Leftrightarrow$
 - 1) r ist Relation vom Format V d.h. $r \subseteq \text{dom}(V)$
 - 2) r genügt allen Integritätsbedingungen von Σ
- Sei $\mathfrak{R} = \{R_1, \dots, R_k\}$, $\Sigma_{\mathfrak{R}}$ eine Menge von interrelationalen Integritätsbedingungen. Ein **Datenbankschema** wird definiert durch
 - $D = (\mathfrak{R}, \Sigma_{\mathfrak{R}})$
- Eine **relationale Datenbank** $d = \{r_1, \dots, r_k\}$ ist eine Menge von Relationen r_i vom Typ R_i $1 \leq i \leq k$, die $\Sigma_{\mathfrak{R}}$ erfüllen
 - Eine relationale Datenbank ist also eine Zusammenfassung mehrerer Relationen mit Integritätsbedingungen

Tabellarische Darstellung von Relationen

- Spaltenüberschriften: Attribute des Relationsschemas
- Tupel: Zeile in Tabelle
- Relation: Einträge in Tabelle
- Einträge in Tabellen gehören zu den entsprechenden Domains



Tabellarische Darstellung von Relationen

Schlüsselattribute

Student = ({matrikelnr, name, wohnort}, {matrikelnr → name, wohnort ist vom Typ String})

Vereinfacht:

Student = ({matrikelnr, name, wohnort})

Mit Darstellung der Wertebereichen:

Student = ({matrikelnr: Integer, name: String, wohnort: String})

Student		
<u>matrikelnr</u>	name	wohnort
132004	Müller	Singen
131208	Zimmer	Lindau
131001	Abel	
131013	Jung	Konstanz
132740	Moser	Singen

Eigenschaften einer Relation / Tabelle

- Eindeutiger Name
- Reihenfolge der Tupel (Zeilen) ist beliebig
- Attributnamen sind eindeutig innerhalb einer Relation
- Die Tupel der Relation (Zeilen der Tabelle) sind paarweise verschieden
- Für das Einbringen eines Tupels in eine Relation ist mindestens der Primärschlüssel vorzugeben
- Nicht-Schlüsselattribute können durch „Nullwerte“ belegt werden
 - Bedeutung: „Wert ist nicht existent“

Primärschlüssel

- Primärschlüssel
 - Attribut (Attributmenge), die ein Tupel eindeutig identifiziert
 - Ein Primärschlüssel existiert nur einmal in einer Relation
 - Bei der Tupel-Suche reicht die Suche nach Primärschlüssels aus
 - Darstellung: Name unterstrichen
- Schlüsselkandidat
 - Es können mehrere potentielle Schlüssel vorhanden sein ("Schlüsselkandidaten")
 - Auszeichnung eines Schlüsselkandidaten als Primärschlüssel
 - Sekundärschlüssel: nicht als Primärschlüssel ausgezeichnet

Studierender			
matrikelnr	personalausweisnr	name	wohnort
132004	1252345432	Müller	Singen
131001	5432534234	Abel	

Primärschlüssel

- Ein Primärschlüssel kann aus mehreren Attributen bestehen

Benotung		
matrikelnr	klausur	note
135745	Datenbanksysteme	2,3
135745	Systemmodellierung	4,0
135663	Datenbanksysteme	1,7

- Einführung "künstlicher" Primärschlüssel (Pseudokey)
 - Falls kein Schlüsselkandidat existiert
 - Falls Schlüsselkandidaten aus zu vielen Attributen besteht

Kunde			
kunde-ID	name	vorname	wohnort
11200	Kunz	Stefan	Konstanz
11210	Maier	Andreas	Konstanz

Primärschlüssel

- Wahl des Primärschlüssels
 - Konstanz: Primärschlüssel sollten sich nicht ändern
 - Wertepflicht: Primärschlüssel muss ein Mußattribut sein
 - Minimalität: Bei zusammengesetzten Schlüsseln sollte Primärschlüssel so gewählt werden, dass kein Attribut entfernt werden kann, ohne dass Identifikationsvermögen verloren geht
 - Design-Entscheidung: Pseudokey / zusammengesetzter Key
- Tabellen mit gleichem Primärschlüssel können zusammengefasst werden

Fremdschlüssel

- Fremdschlüssel
 - Attribute zur Identifikation von Tupel aus anderen Relationen
 - Modellierung einer Zuordnung (Beziehung)
 - Darstellung: Name gestrichelt unterstrichen

Kunde			
kundennr	name	vorname	wohnort
11200	Kunz	Stefan	Konstanz
11210	Maier	Andreas	Konstanz

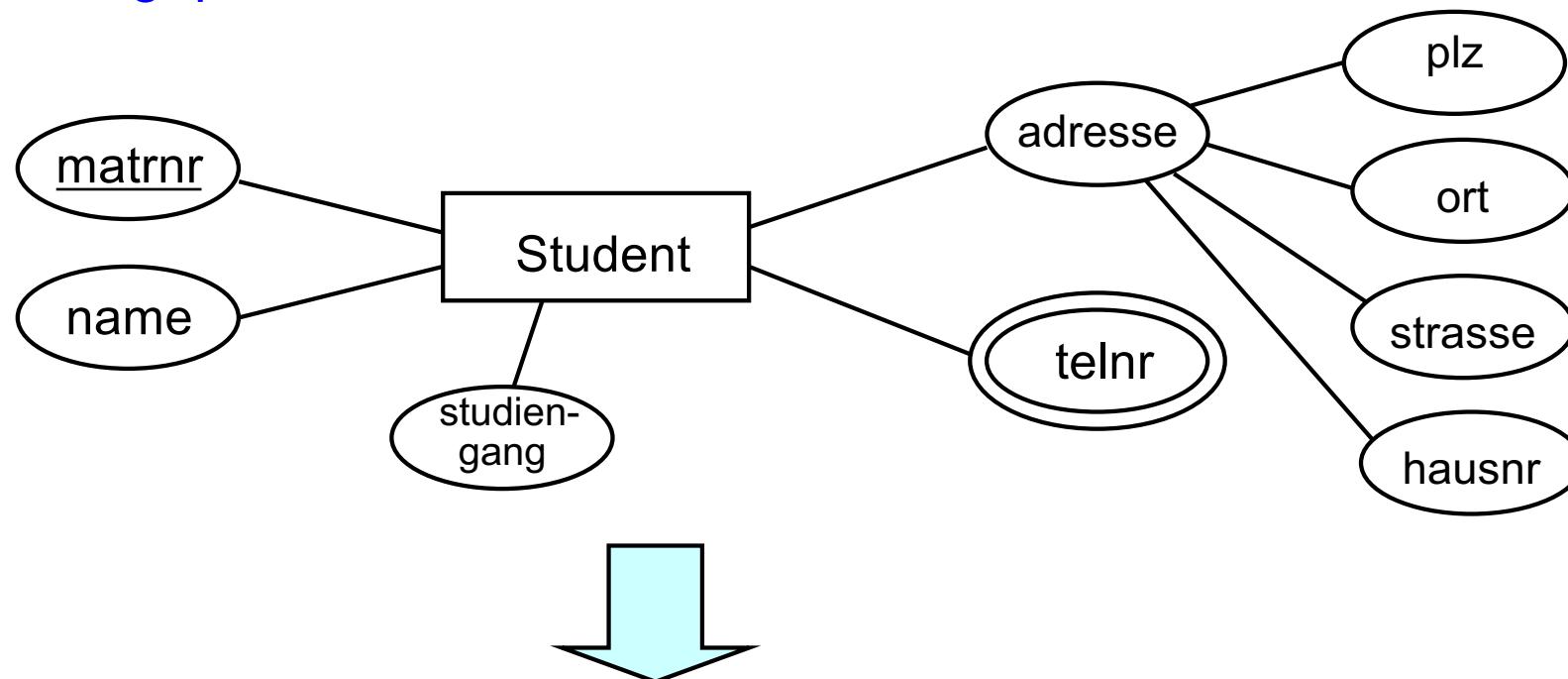
Artikel		
artikelnr	bezeichnung	preis
224	Fernseher	600
116	DVD-Player	200

Bestellung			
bestellnr	kundennr	artikelnr	datum
224533	11200	224	21.03.2005
226522	11210	116	17.02.2005

Transformation von ER-Modellen

Entity-Typen

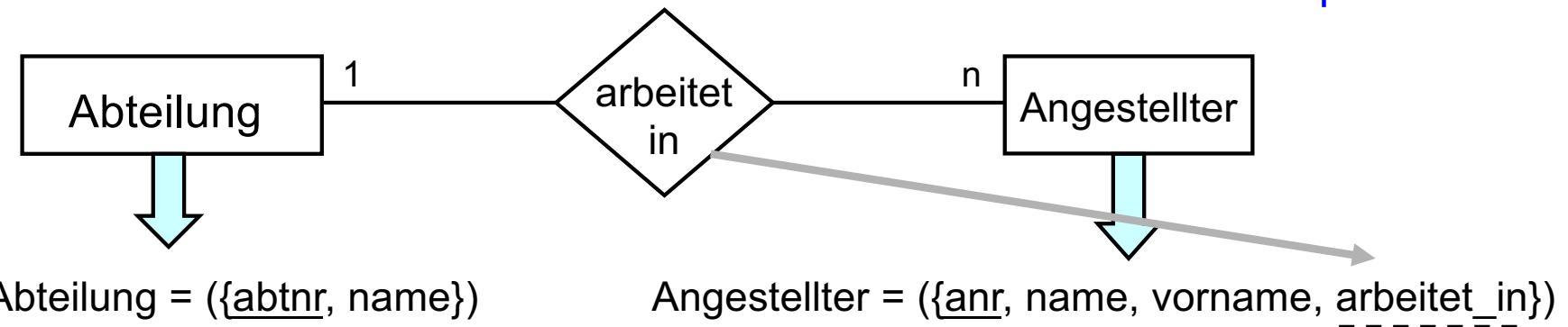
- Abbildung Entity-Typen auf Relationen
 - Abbildung Entity-Attribute auf Attribute der Relation
 - Mehrwertige und zusammengesetzte Attribute müssen noch angepasst werden !



$\text{Student} = (\{\underline{\text{matrnr}}, \text{name}, \text{studiengang}, (\text{plz}, \text{stadt}, \text{strasse}, \text{hnر}), \{\text{telnr}\}\})$

Transformation von 1:n-Relationships

- Abbildung 1:n-Relationship
 - Anhängen der Attribute an die Relation, die dem Entity-Typ mit der mit „n“ bezeichneten Kante entspricht
- Fremdschlüssel
 - Attributmenge, die in einer anderen Relation Primärschlüssel ist
 - Name: Name des Primärschlüssels oder Name der Relationship



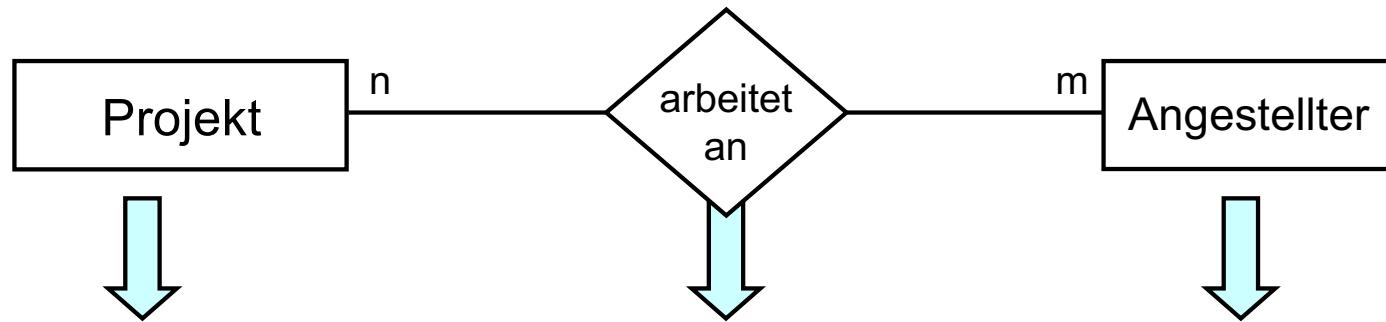
Beispiel:

Abteilung	
abtnr	name
448	HW
122	SW
200	Personal

Angestellter			
anr	name	vorname	arbeitet_in
2004	Müller	Hans	448
1208	Zimmer	Jochen	122
1001	Abel	Kai	122

Transformation von n:m-Relationships

- Realisierung einer eigenen Relation
 - Schlüssel sind die Schlüssel der Relationen der beteiligten Entity-Typen
 - Attribute des Beziehungstyps als zusätzliche Attribute



Projekt = (pnr, name)

arbeitet_an = (pnr, anr)

Angestellter = (anr, name, vorname)

Beispiel:

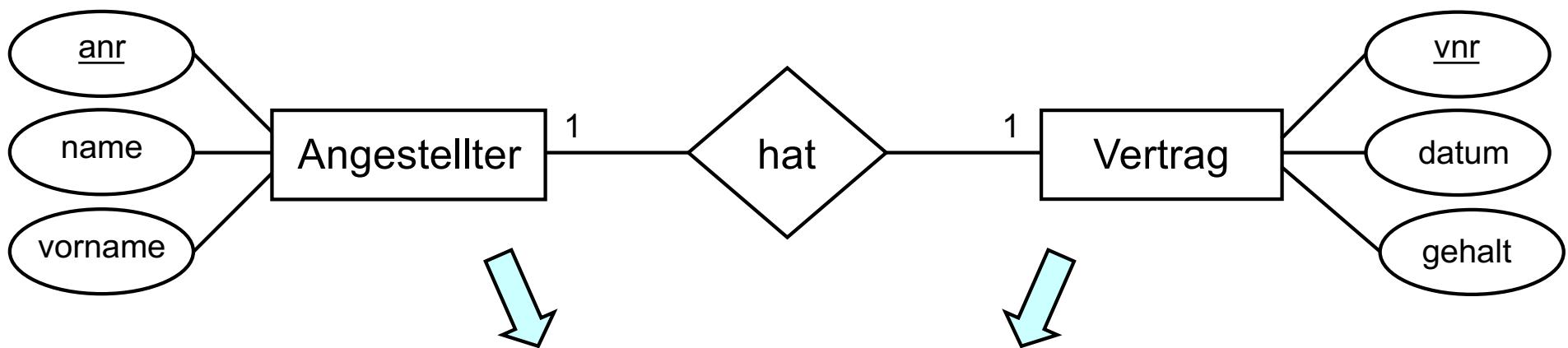
Projekt	
pnr	name
56	PS1
77	QEM
12	PN

arbeitet an	
pnr	anr
56	2004
77	2004
77	1001

Angestellter		
anr	name	vorname
2004	Müller	Hans
1208	Zimmer	Jochen
1001	Abel	Kai

Transformation von 1:1-Relationships

- Möglichkeit 1
 - Zusammenfassen zu einer Relation

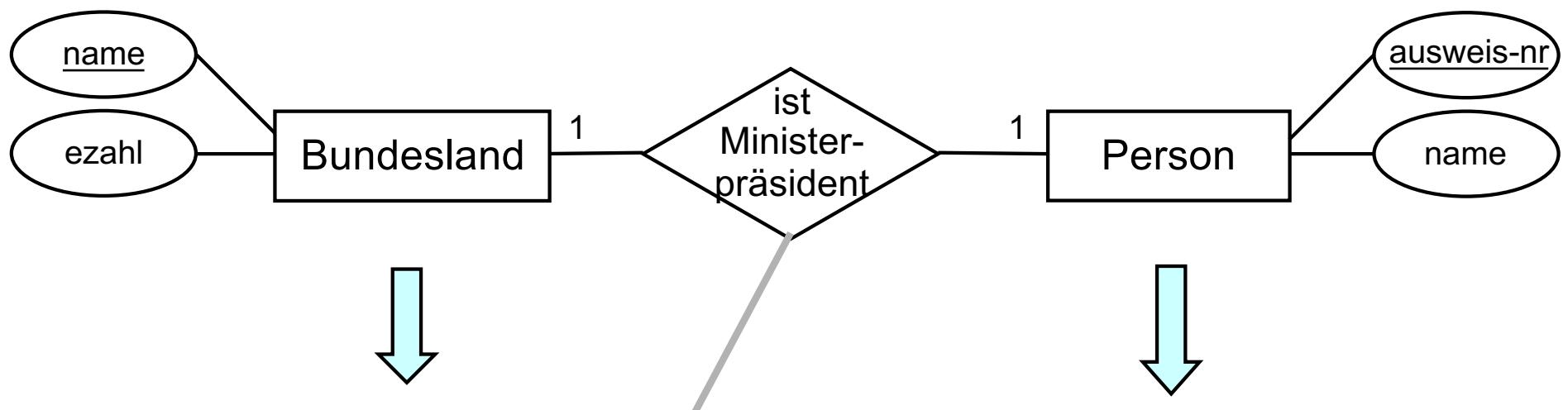


Beispiel:

Angestellter					
<u>anr</u>	name	vorname	vnr	datum	gehalt
2004	Müller	Hans	v646	1.10.1999	3500
1208	Zimmer	Jochen	v83	1.1.2002	4000
1001	Abel	Kai	v143	1.3.1990	5500

Transformation von 1:1-Relationships

- Möglichkeit 2
 - Übernahme des Primärschlüssels

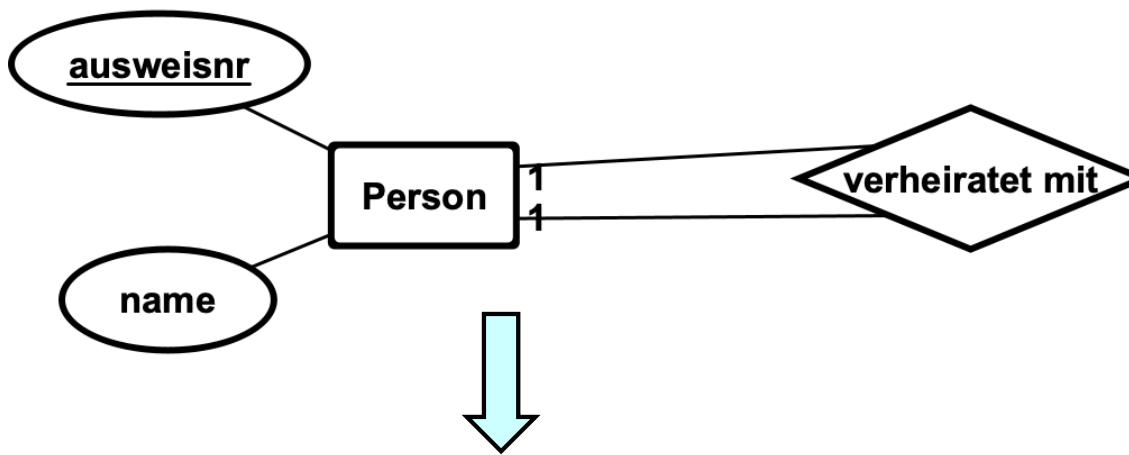


Bundesland		
<u>name</u>	<u>ezahl</u>	<u>ausweis-nr</u>
BW	10.000.000	20045566
Bayern	12.000.000	12087744

Person	
<u>ausweis-nr</u>	<u>name</u>
20045566	Kretschmann
12087744	Söder

Transformation von 1:1-Relationships

- Möglichkeit 1
 - Schlechte Umsetzung !!!



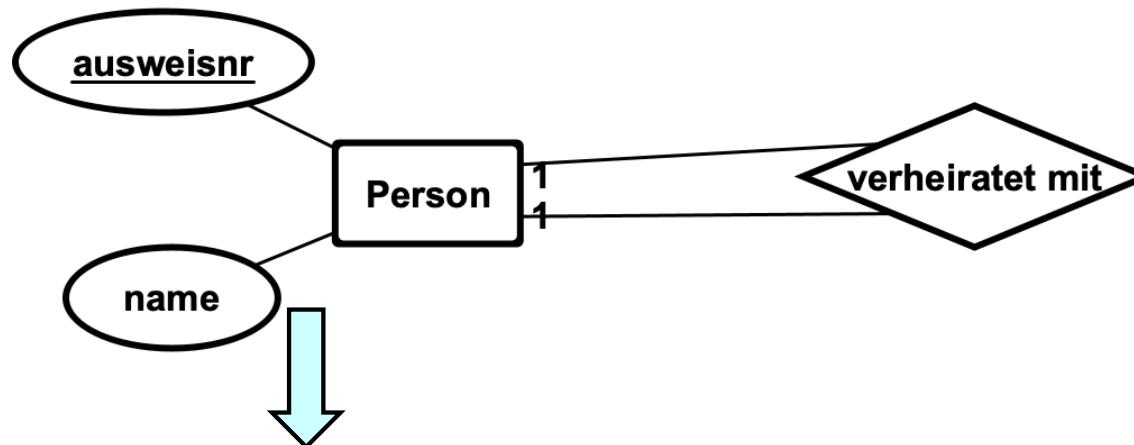
Person = ({ausweisnr1, name1, ausweisnr2, name2})

Person			
ausweisnr1	name1	ausweisnr2	name2
3333212	Müller		
		4444222	Maier
5555313	Kunz	7777636	Kunz

Probleme?

Transformation von 1:1-Relationships

- Möglichkeit 2
 - Realisierung als eigene Relation

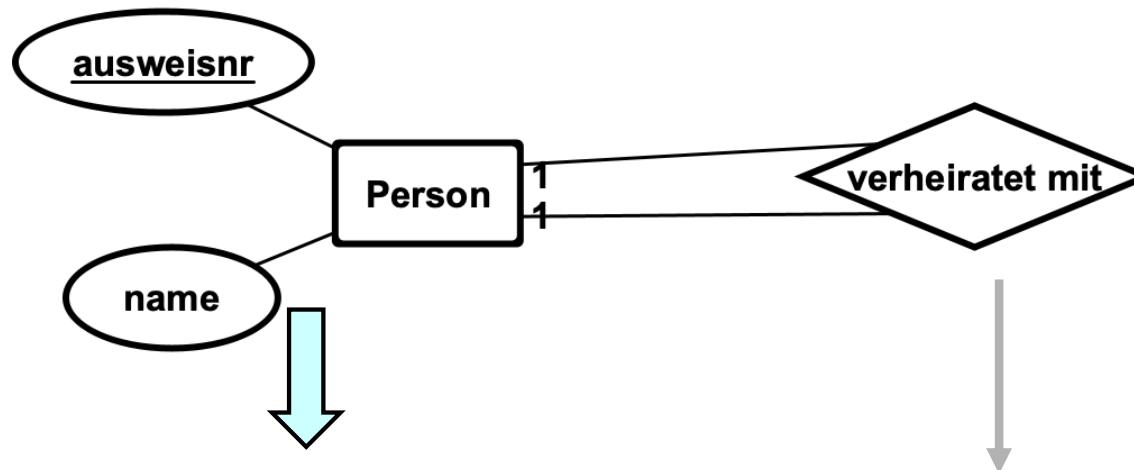


Person = ({ausweisnr, name, verheiratetMit})

Person		
ausweisnr	name	verheiratetMit
3333212	Müller	
5555313	Kunz	7777636
7777636	Kunz	???

Transformation von 1:1-Relationships

- Möglichkeit 3 – Vermeidung von NULL-Werten
 - Realisierung als eigene Relation



Person = ({ausweisnr, name})

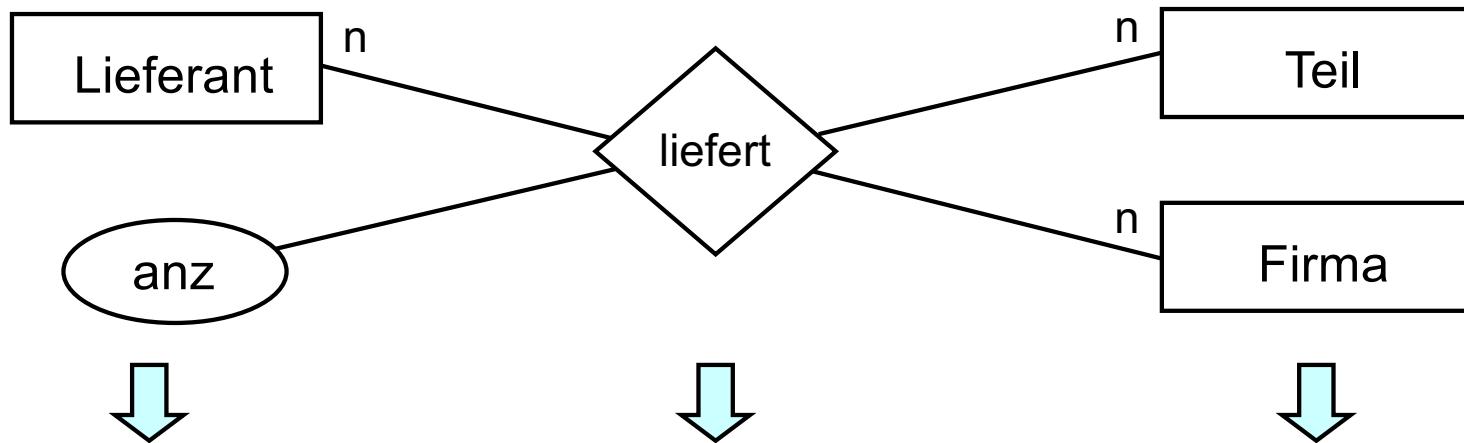
Person	
<u>ausweisnr</u>	name
3333212	Müller
4444222	Maier
5555313	Kunz

verheiratet = ({ausweisnr1, ausweisnr2})

verheiratet	
<u>ausweisnr1</u>	<u>ausweisnr2</u>
3333212	5345432
4444222	5555313

Transformation von mehrstellige Relationships

- Abbildung einer mehrstelligen Relationship analog zu n:m-Relationship



Lieferant = ({Inr, name})

liefert = ({Inr, tnr, fnr, anz})

Firma = ({fnr, name})

Teil = ({tnr, name})

Beispiel:

Lieferant	
<u>Inr</u>	name
346	XY GmbH
624	KL AG

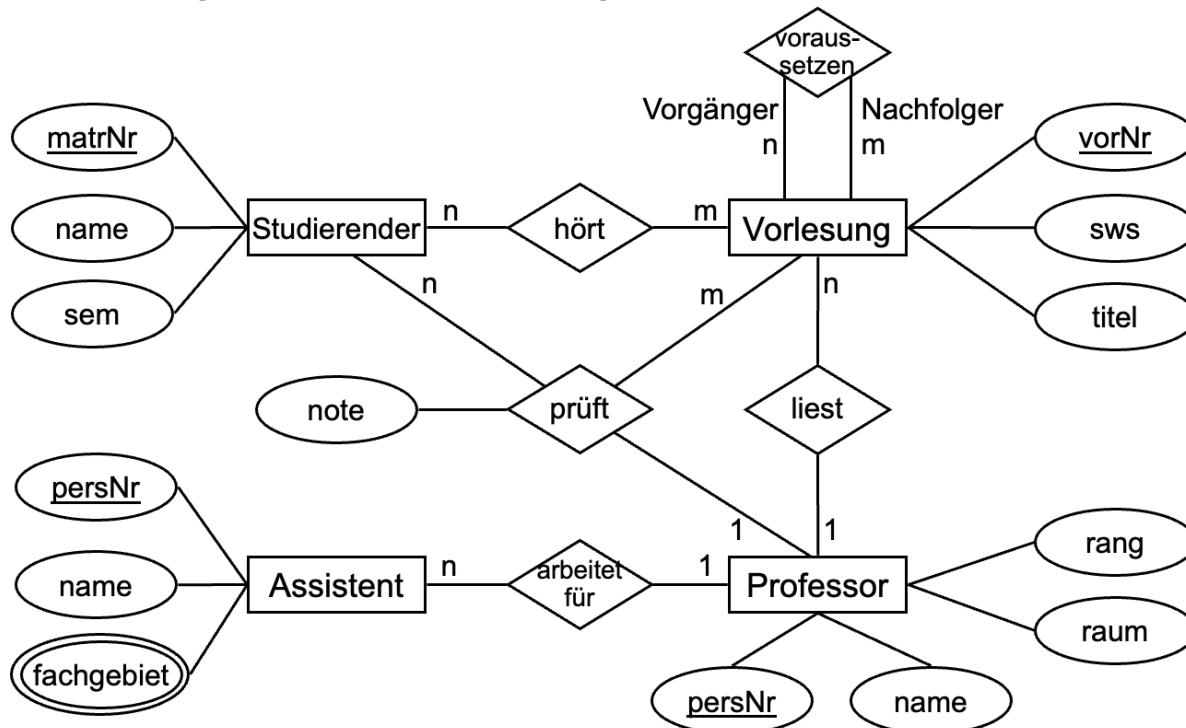
Teil	
<u>tnr</u>	name
4211	Getriebe
6344	Kupplung

Firma	
<u>fnr</u>	name
442	AB GmbH
613	DE AG

liefert			
<u>Inr</u>	<u>tnr</u>	<u>fnr</u>	<u>anz</u>
346	4211	442	1000
624	6344	613	500

Transformation von mehrstellige Relationships

- Abbildung einer dreistelligen Relationship



Student = {matrNr, name, sem}

Vorlesung = {vorNr, sws, titel}

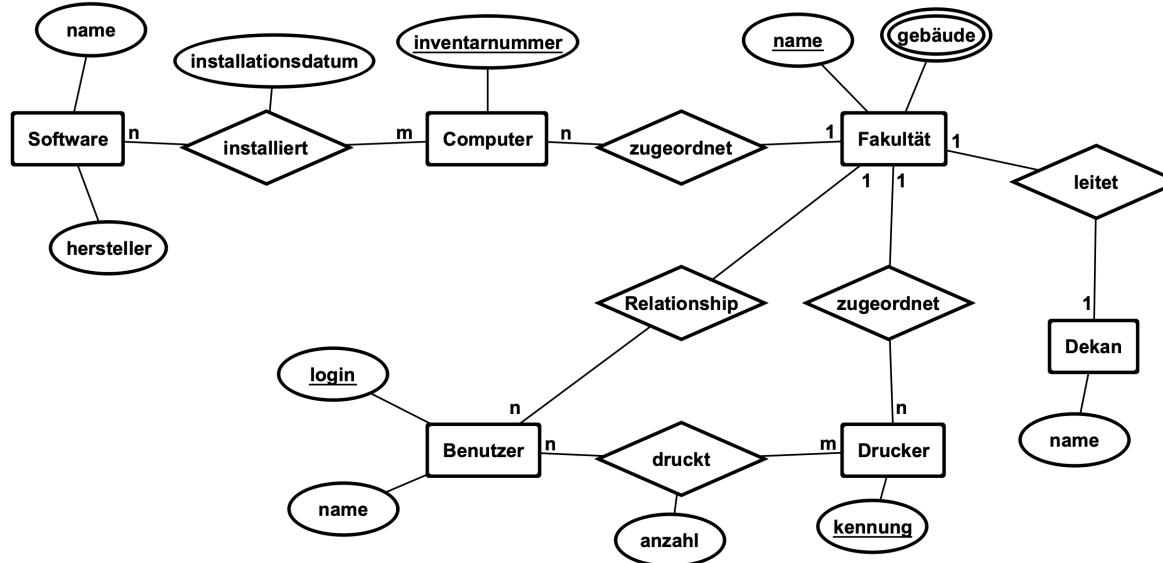
Professoren = {persNr, name, raum, rang, profPersNr}

Assistent = {persNr, name}

prüft = {matrNr, vorNr, persNr, note}

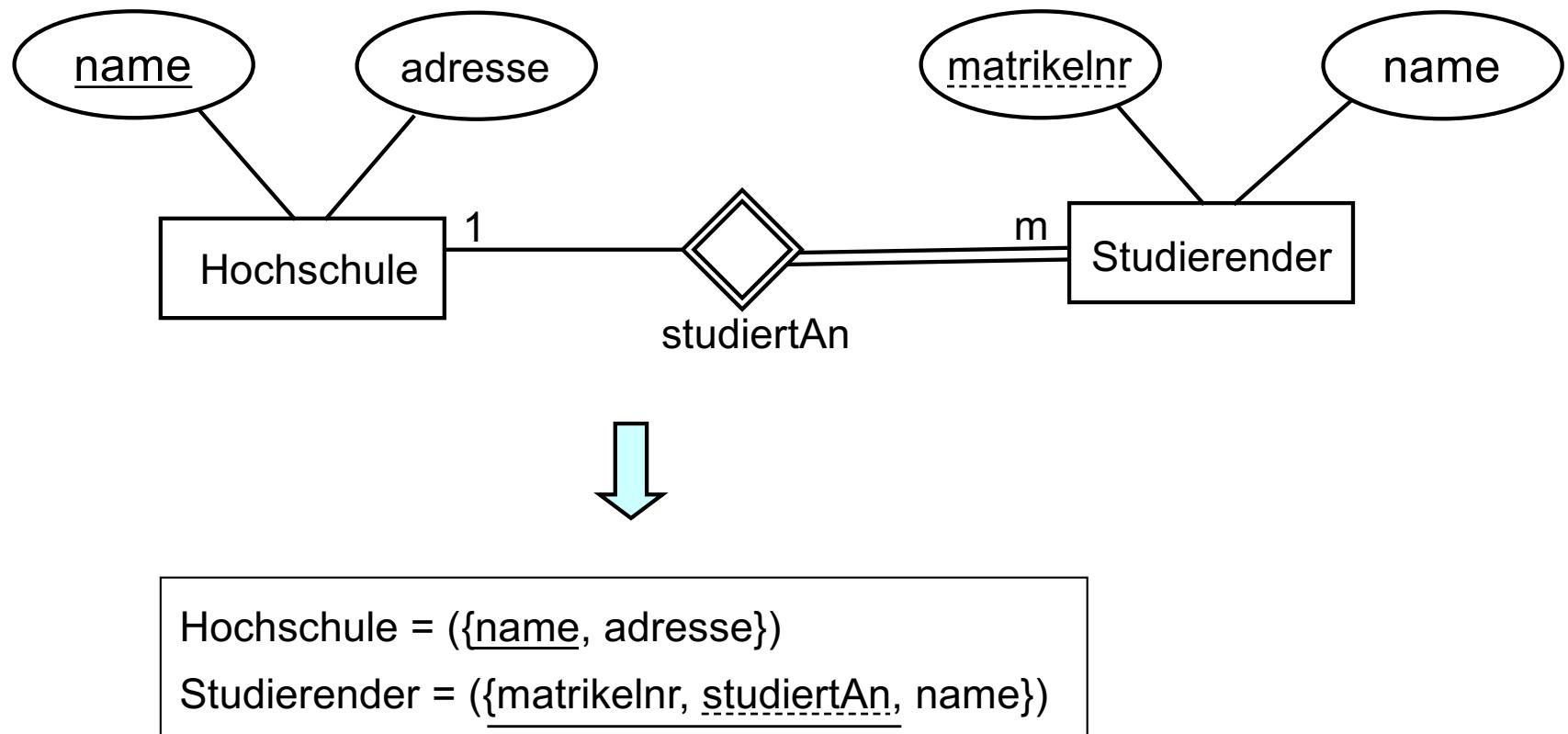
Quelle: Kemper: Datenbanksysteme

Beispiel Abbildung Relationenmodell

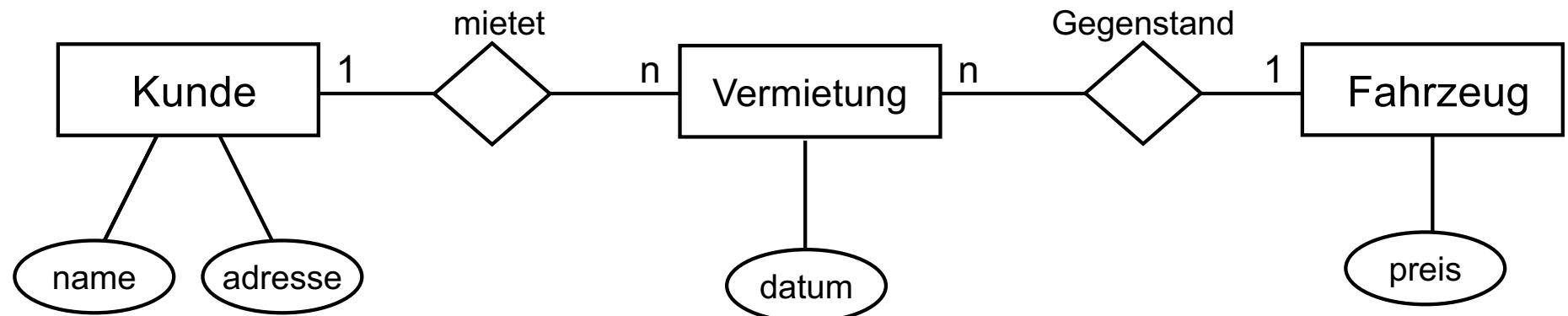
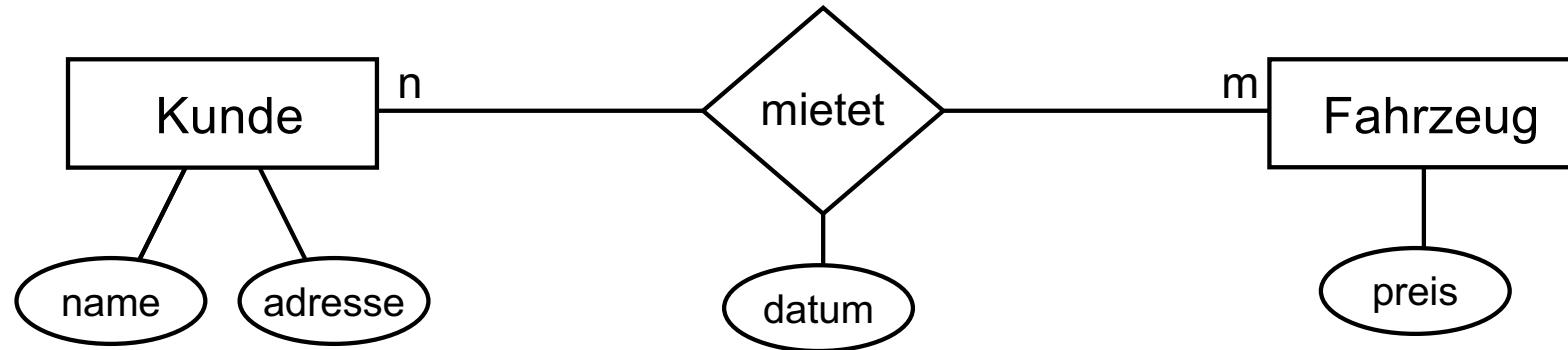


Transformation schwacher Entities

- Partieller Schlüssel bildet zusammen mit Fremdschlüssel den Primärschlüssel der Relation

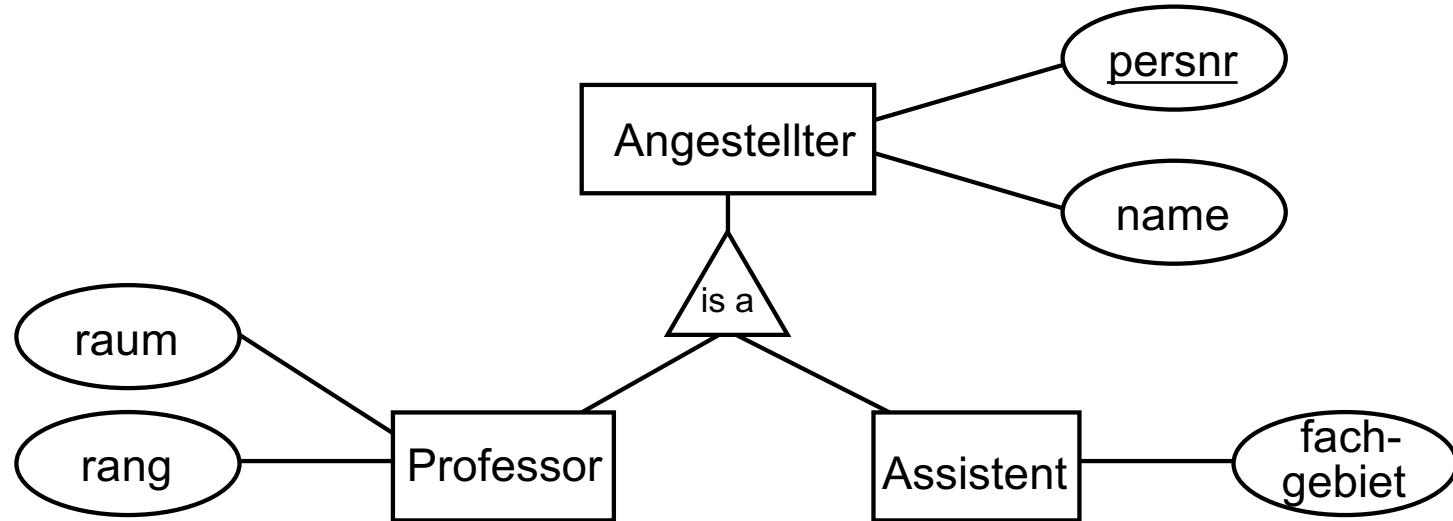


Unterschiedliche Abbildung ?



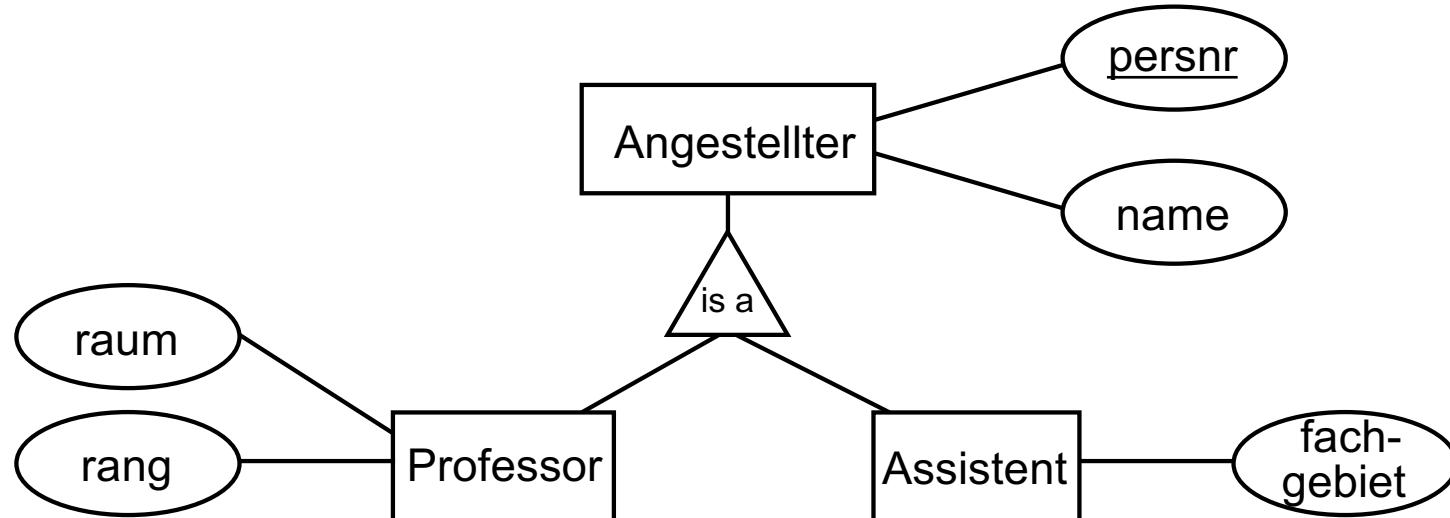
Transformation von Spezialisierung

- Keine direkte Abbildung in relationales Modell möglich



Transformation von Spezialisierung

- Keine direkte Abbildung in relationales Modell möglich



1. Table per Subclass / Vertical

```
Angestellter = ({persnr, name})
```

```
Professor = ({persnr, raum, rang})
```

```
Assistent = ({persnr, fachgebiet})
```

2: Table per Class Hierarchy / Flat

```
Angestellter = ({persnr, name, raum, rang,
                 fachgebiet, is_a})
```

3. Table per Concrete Class / Horizontal

```
Professor = ({persnr, name, raum, rang})
```

```
Assistent = ({persnr, name, fachgebiet})
```

Zusammenfassung ER → Relationenmodell

ER-Modell	Relationenmodell
Entity-Typ	Relation
1:1 – Relationship	Zu einer Relation zusammenfassen
1:n – Relationship	Fremdschlüssel zu eindeutiger Relation
n:m – Relationship	Relation aus Relationship mit zwei Fremdschlüssel
n-äre Relation	Relation aus Relationship mit n Fremdschlüssel
Einfaches Attribut	Attribut
Zusammengesetztes Attribut	Menge von einfachen Attributen
Mehrwertiges Attribut	Relation mit Fremdschlüssel
Schlüsselattribut	Primärschlüssel (oder Sekundärschlüssel)

Motivation Normalformen

Artikel					
artikelnr	bezeichnung	preis	lagernr	lagerort	lagerstrasse
211	Radkappe	25	20	Konstanz	Seestrasse
333	Pumpe	100	15	Stuttgart	Hauptstrasse
655	Federbein	250	15	Stuttgart	Hauptstrasse
225	Kugellager	190	15	Stuttgart	Hauptstrasse

Motivation Normalformen

Artikel					
artikelnr	bezeichnung	preis	lagernr	lagerort	lagerstrasse
211	Radkappe	25	20	Konstanz	Seestrasse
333	Pumpe	100	15	Stuttgart	Hauptstrasse
655	Federbein	250	15	Stuttgart	Hauptstrasse
225	Kugellager	190	15	Stuttgart	Hauptstrasse

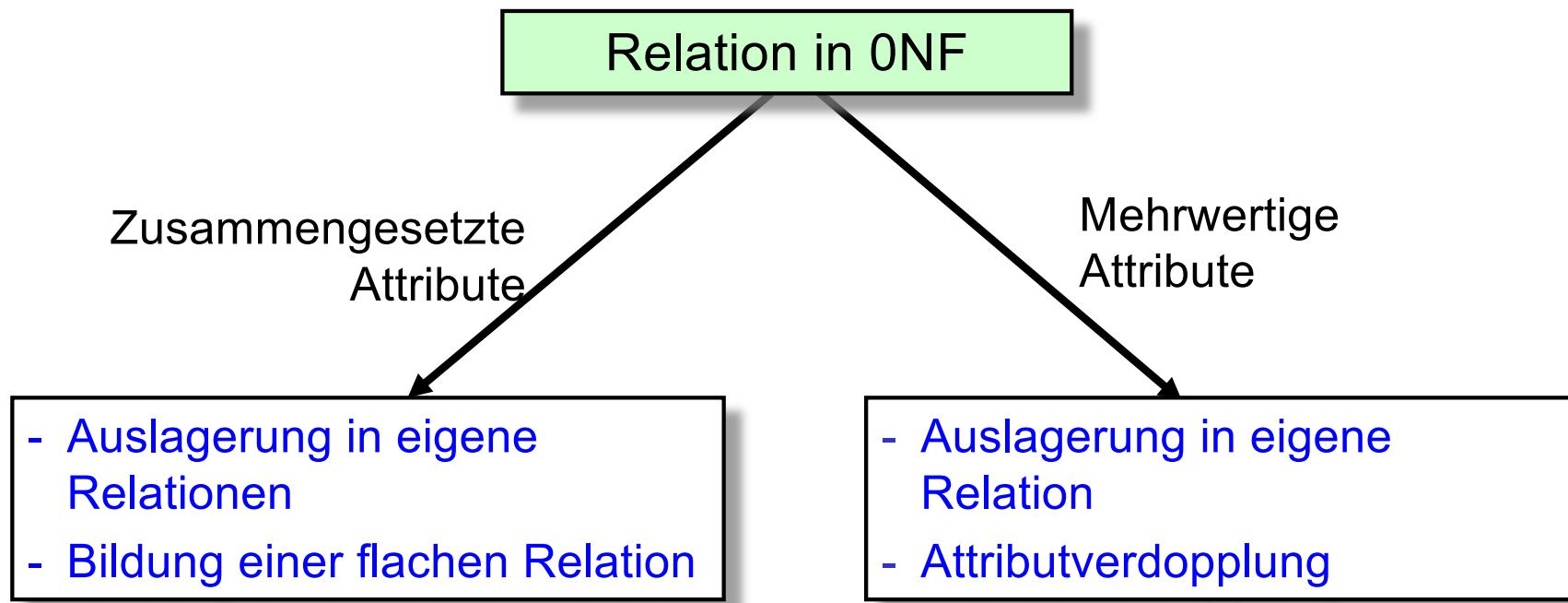
- Einfügeanomalie
 - Ein neues Lager kann nur dann aufgenommen werden, wenn bereits ein Artikel zugeordnet ist
- Änderungsanomalie
 - Wenn die Strasse des Lagers in Stuttgart geändert wird, müssen mehrere Tupel geändert werden
- Löschanomalie
 - Wenn die Radkappe aus der Artikelliste gelöscht wird, wird auch das Lager in Konstanz gelöscht

Flache Relationen

- Flache Relationen sind Relationen, die
 - Keine mehrwertigen Attribute und
 - Keine zusammengesetzte Attribute enthalten
- Beispiel für mehrwertiges Attribut
 - Student = (<{ matrnr}, name, {telnr} })
- Beispiel für zusammengesetztes Attribut
 - Person = (<{ pnr}, name, (plz, stadt, strasse, hnr) })

1. Normalform.

- Eine Relation ist in erster Normalform (1NF), wenn jeder Attributwert elementar ist
 - d.h. es existieren keine matrix-, listen-, mengenwertige Attribute
- Normalisierung:



Redundanz und Konsistenz

- Redundanz (redundancy)
 - Maß für „Überflüssigkeit“ von Informationen
 - Redundante Information: mehrfaches Vorhandensein gleicher Information
- Änderungen von redundanten Informationen führen häufig zu Inkonsistenzen

Funktionale Abhängigkeiten

- X, Y sind Attributmengen, R ein Relationsschema
- Y heißt *funktional abhängig* von X in R , geschrieben $X \rightarrow Y$, wenn es in jeder Relation zu R keine zwei Tupel gibt, die in ihrem Wert unter X , aber nicht in ihrem Wert unter Y übereinstimmen
- Y heißt *voll funktional abhängig* von X in R , geschrieben $X \xrightarrow{*} Y$, wenn X minimal ist, d.h. wenn es keine Teilmenge aus X gibt, die bereits Y funktional bestimmt
- Triviale funktionale Abhängigkeiten: $X \rightarrow Y$ mit $Y \subseteq X$

2. Normalform

- Eine Relation ist in zweiter Normalform (2NF), wenn sie
 - in 1NF ist und
 - jedes Nichtschlüsselattribut von jedem Schlüsselkandidaten voll funktional abhängig ist
- Informale Beschreibung
 - Ein Relationenschema verletzt die 2NF, wenn Informationen über mehr als ein einziges Konzept modelliert werden
- Normalisierung durch
 - Auslagerung nicht abhängiger Schlüsselattribute in eigene Tabellen

arbeitetIn					
projektnr	personalnr	projektname	name	vorname	seit
P01	2004	Carrera	Müller	Hans	01.10.2000
P02	1208	Coyote	Zimmer	Jochen	01.01.1999
P02	1001	Coyote	Abel	Kai	01.09.1995
P02	2004	Coyote	Müller	Hans	01.12.2000

3. Normalform

- Eine Relation ist in dritter Normalform (3NF), wenn sie
 - in 2NF ist und
 - kein Nichtschlüsselattribut transitiv von einem Schlüsselattribut abhängt
- Anschaulicher: In der Menge der Nichtschlüsselattribute darf es keine nicht trivialen funktionale Abhängigkeiten geben

Artikel					
artikelnr	bezeichnung	preis	lagernr	lagerort	lagerstrasse
211	Radkappe	25	20	Konstanz	Seestrasse
333	Pumpe	100	15	Stuttgart	Hauptstrasse
655	Federbein	250	15	Stuttgart	Hauptstrasse
225	Kugellager	190	15	Stuttgart	Hauptstrasse



Boyce-Codd Normalform

- Eine Relation ist in Boyce-Codd Normalform (BCNF), wenn sie
 - in 3NF ist und
 - keine transitiven Abhangigkeiten der Schlsselattribute existieren
- Anschaulicher: In der Menge der Schlsselattribute darf es keine funktionale Abhangigkeiten geben
- Beispiel Relation Adresse:
 - Adresse = ({plz, ort, bundesland, hauptstadt})
 - Schlsselkandidaten: plz, {ort, bundesland}, {ort, hauptstadt}
 - Transitive Abhangigkeiten: plz \rightarrow bundesland \rightarrow hauptstadt

*A table is based on
the key,
the whole key,
and nothing but the key (so help me Codd)*

(William Kent, "A Simple Guide to Five Normal Forms in Relational Database Theory")

Normalformen

Beispiel

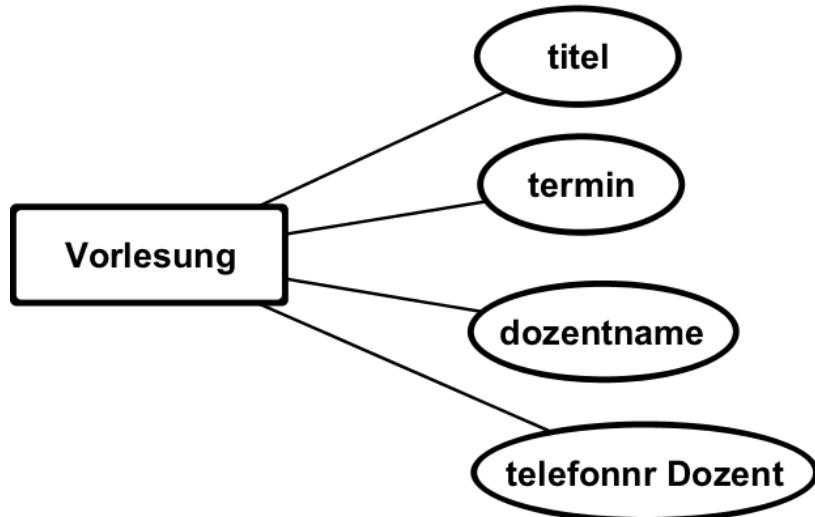
Gegeben ist ein Realweltausschnitt „Reisebüro“ mit folgender Relation:

Reisebuchung = ({buchungs-nr}, reisebuero-nr, reisebuero-name, kunden-nr,
kunden-nachname, kunden-vorname, kunden-wohnort, {kunden-tel-nr},
buchungs-datum, reise-nr, reise-beschreibung})

- In welcher Normalform befindet sich diese Relation? Begründen Sie kurz Ihre Antwort.
- Überführen Sie die Relation in die BCN Normalform. Legen Sie dabei auch die Primärschlüssel sinnvoll durch Unterstreichen der jeweiligen Attribute fest.

Kurzwiederholung ER-Modell

- Durch eine korrekte ER-Modellierung Abbildung ins Relationenmodell erhält man gleich Relationen in BCNF
- Relationen welcher Normalform werden aus folgendem Modell gebildet?



Relationale Algebra

Übersicht

- Selektion σ
 - Auswahl von Tupel aus einer Relation
- Projektion π
 - Auswahl auf ausgewählte Attribute einer Relation
- Verbund \bowtie
 - Verbindung mehrerer Relationen über gemeinsame Attribute

Selektion und Projektion

- Selektion σ : Auswahl von Tupel aus einer Relation
 - $\sigma_{\text{name} = \text{„Müller“}}(\text{Student})$

Student		
matrikelnr	name	wohnort
132004	Müller	Singen

- Projektion π : Auswahl von Spalten (Attribute) aus einer Relation
 - $\pi_{\text{name}, \text{wohnort}}(\text{Student})$

name	wohnort
Müller	Singen
Zimmer	Lindau

Beispiel zu relationale Operationen

Student = ({ matrikelnr, name, sg })

Studium = ({ sg, abschluss})

Student		
<u>matrikelnr</u>	name	<u>sg</u>
134711	Müller	AIN
138801	Kunz	WIN
131294	Maier	MSI

Studium	
<u>sg</u>	abschluss
AIN	Bachelor of Science
WIN	Bachelor of Science
MSI	Master of Science

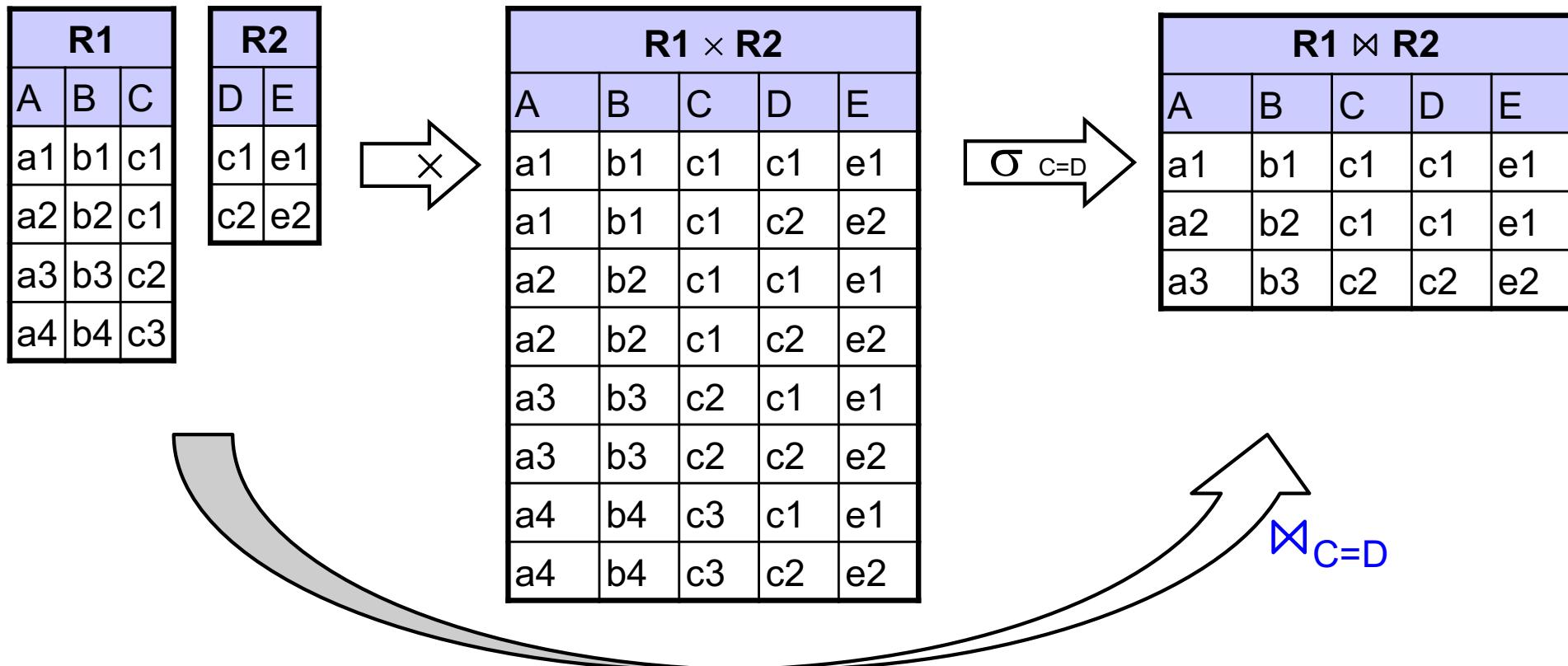
Welche Matrikelnummer hat der Student mit Namen Müller?

Join-Operation

- Join (Verbund)
 - Kartesisches Produkt zw. Relationen, eingeschränkt durch gemeinsame Attribute
 - Inverse ist Projektion
 - Symbol \bowtie
- Definition Theta-Join (Θ -Join, Theta-Verbund)
 - $R \bowtie_{A\Theta B} S = \sigma_{A\Theta B} (R \times S), \quad \Theta = \{ <, =, >, \leq, \neq, \geq \}$
- Andere Schreibweise
 - $R (A \Theta B) S$

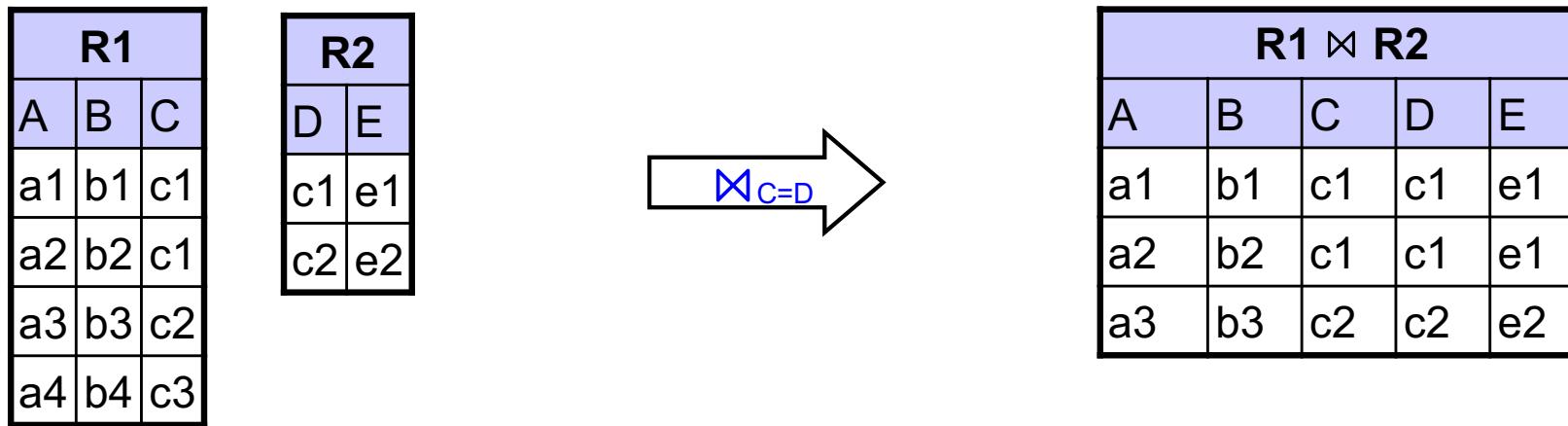
Equi-Join (Gleich-Verbund)

- Join = Kartesisches Produkt zw. Relationen, eingeschränkt durch Gleichheit zwischen Attributen
 - $R1 \bowtie_{C=D} R2 = \sigma_{C=D} (R1 \times R2)$



Eigenschaften Join

- Eigenschaften
 - Assoziativ: $(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$
 - Kommutativ: $A \bowtie B = B \bowtie A$
- Equi-Join (Gleich-Verbund)
 - Kartesisches Produkt zw. Relationen, eingeschränkt durch Gleichheit zwischen Attributen

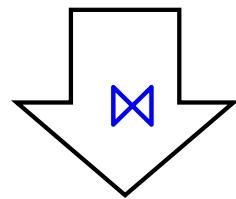


Beispiel Equi-Join

Student	
matrikelNr	name
260111	Hans Müller
260112	Kai Maier
260113	Egon Schneider

besucht	
matrikelNr	vorlNr
260111	246
260111	241

Vorlesung	
vorlNr	name
246	Datenbanksysteme
241	Systemmodellierung

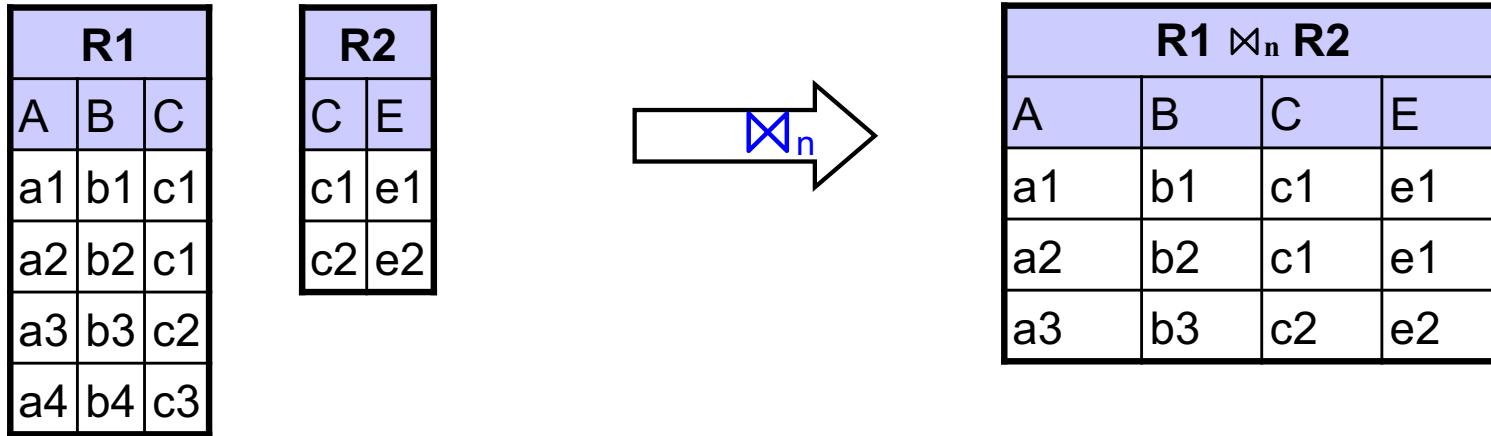


Student.Matrikel-Nr = besucht.Matrikel-Nr
besucht.Vorl-Nr = Vorlesung.Vorl-Nr

Student \bowtie besucht \bowtie Vorlesung					
Student		besucht		Vorlesung	
matrikelNr	name	matrikelNr	vorlNr	vorlNr	name
260111	Hans Müller	260111	246	246	Datenbanksysteme
260111	Hans Müller	260111	241	241	Systemmodellierung

Natural Join (natürlicher Verbund)

- Gleichsetzen aller Attribute, die gleich heißen
- Entfernung der redundanten Attribute

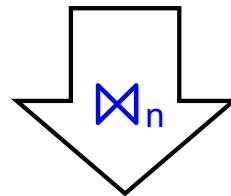


Beispiel Natural Join

Student	
<u>matrikelNr</u>	name
260111	Hans Müller
260112	Kai Maier
260113	Egon Schneider

besucht	
<u>matrikelNr</u>	<u>vorl-nr</u>
260111	246
260111	241

Vorlesung	
<u>vorlNr</u>	name
246	Datenbanksysteme
241	Systemmodellierung



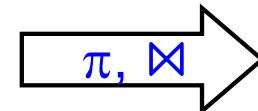
Probleme?

Student \bowtie_n besucht \bowtie_n Vorlesung			
<u>matrikelNr</u>	name	<u>vorlNr</u>	name
260111	Hans Müller	246	Datenbanksysteme
260111	Hans Müller	241	Systemmodellierung

Beispiel Join-Operation

- Beispiel: Welche Studenten machen welchen Abschluss?
 - $\pi_{\text{name}, \text{abschluss}} \bowtie_{\text{sg} = \text{sg}}$ (Student, Studium)

Student		
matrikelNr	name	sg
134711	Müller	AIN
138801	Kunz	WIN
131294	Maier	MSI



Studium	
sg	abschluss
AIN	Bachelor of Science
WIN	Bachelor of Science
MSI	Master of Science

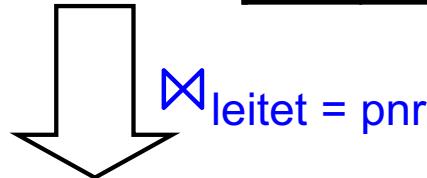
name	abschluss
Müller	Bachelor of Science
Kunz	Bachelor of Science
Maier	Master of Science

Eigenschaften Inner-Joins

Ziel: Auflistung aller Mitarbeiter und der Projekte, die sie leiten:

Mitarbeiter		
mnr	name	leitet
4711	Müller	44
8801	Schmidt	33
1288	Huber	
1294	Maier	42

Projekt	
pnr	projektnname
33	Projekt 1
44	Projekt 2
55	Projekt 3
42	Projekt 4

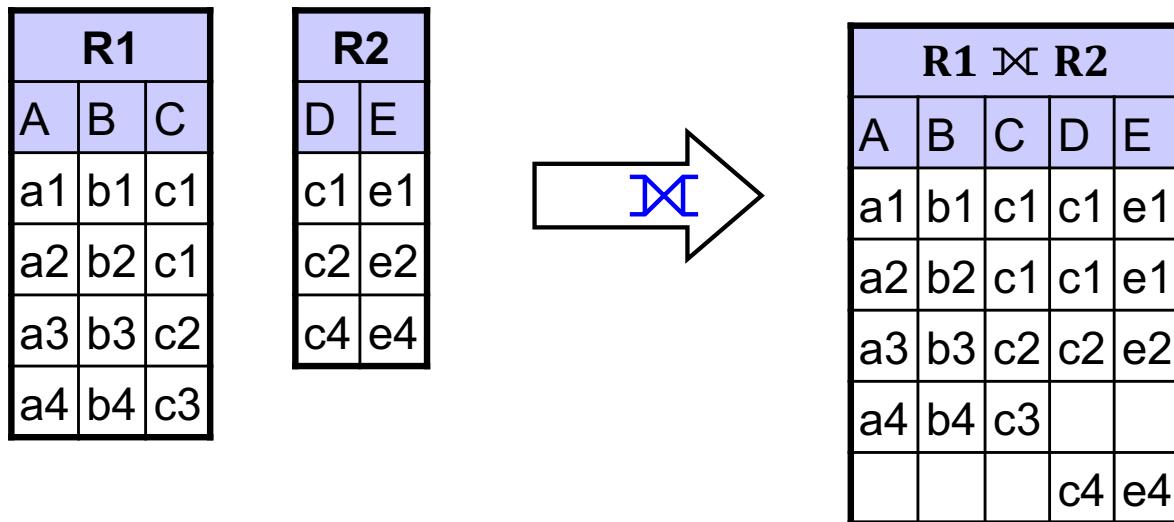


Mitarbeiter \bowtie Projekt				
mnr	name	leitet	pnr	projektnname
4711	Müller	44	44	Projekt 2
8801	Schmidt	33	33	Projekt 1
1294	Maier	42	42	Projekt 4

Mitarbeiter
Huber fehlt !

Outer Joins (1)

- Full Outer Join
 - Bei einem full outer join zwischen Relationen R1 und R2 werden auch dann Tupel aus R1 (bzw. R2) in das Ergebnis aufgenommen, für die es kein passendes Tupel in R2 (bzw. R1) gibt
 - Fehlende Attributwerte werden mit Nullwerten aufgefüllt
 - Symbol \bowtie



Outer Joins (2)

- Left Outer Join von A, B
 - Der left outer join ist ein Verbund vereinigt mit den Tupeln aus A
 - Symbol \bowtie
- Right Outer Join von A, B
 - Der right outer join ist ein Verbund vereinigt mit den Tupeln aus B
 - Symbol \ltimes

Aufgabe zu Outer Join

Gegeben seien die folgenden Tabellen L und R. Geben Sie für die folgenden Join-Verknüpfungen die Ergebnistabellen an.

- a) $L \bowtie_{L.a = R.d} R$
- b) $L \bowtie_{L.b = R.d} R$
- c) $L \bowtie_{L.a = R.c} R$

L	
a	b
8	2
15	16
40	16

R	
c	d
2	2
8	40
8	16

Beispiel zu relationale Operationen

Student = ({ matrikelnr, name, sg })

Studium = ({ sg, abschluss })

Student		
<u>matrikelNr</u>	name	<u>sg</u>
134711	Müller	AIN
138801	Kunz	WIN
131294	Maier	MSI

Studium	
<u>sg</u>	abschluss
AIN	Bachelor of Science
WIN	Bachelor of Science
MSI	Master of Science

Welchen Abschluss macht der Student Müller?

Beispiel zu relationale Operationen

Student = ({ matrikelnr, name, sg })

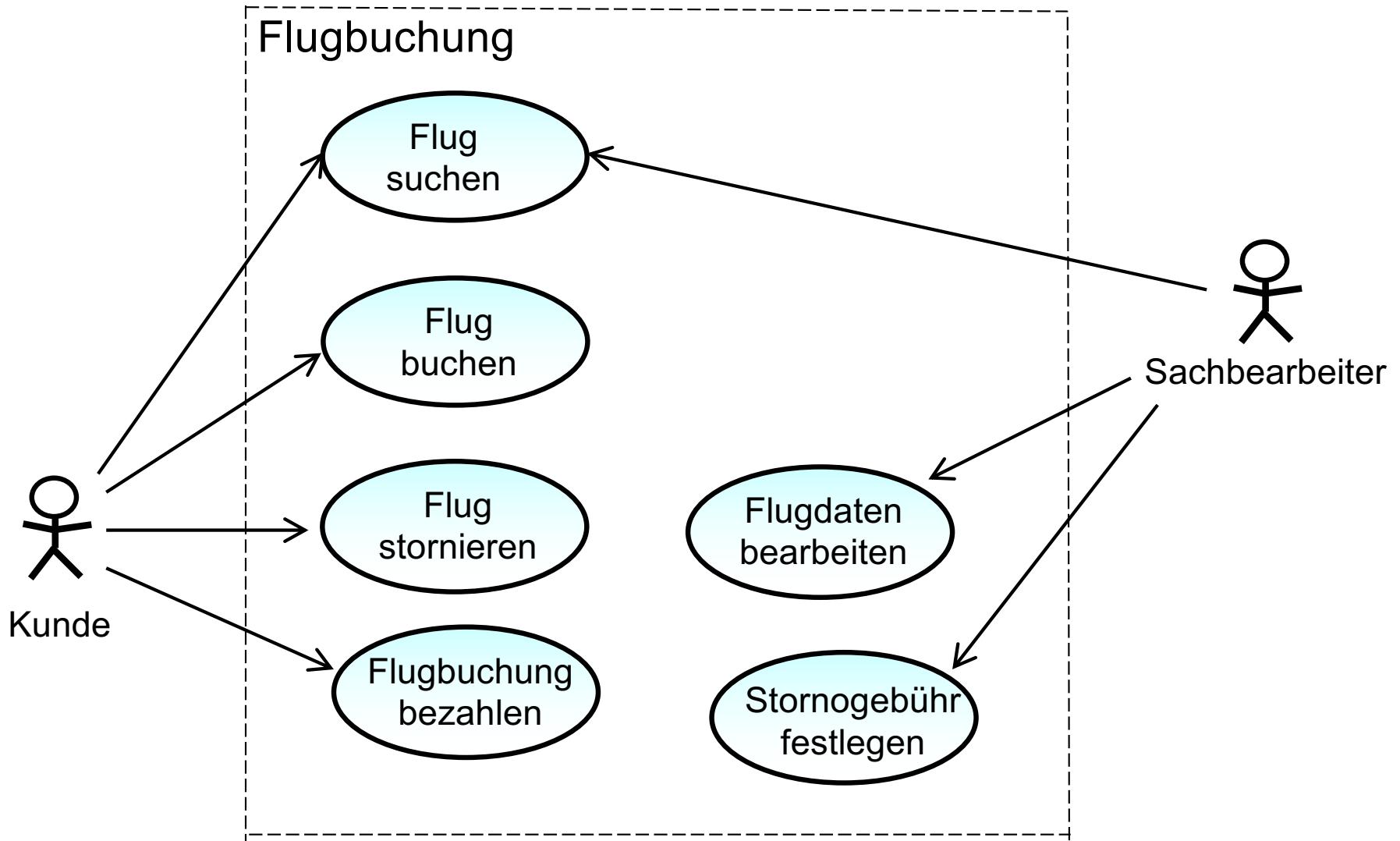
Studium = ({ sg, abschluss })

Student		
<u>matrikelNr</u>	name	<u>sg</u>
134711	Müller	AIN
138801	Kunz	WIN
131294	Maier	MSI

Studium	
<u>sg</u>	abschluss
AIN	Bachelor of Science
WIN	Bachelor of Science
MSI	Master of Science

Welche Studiengänge haben keinen einzigen Studenten?

Beispiel für Use-Case-Diagramm



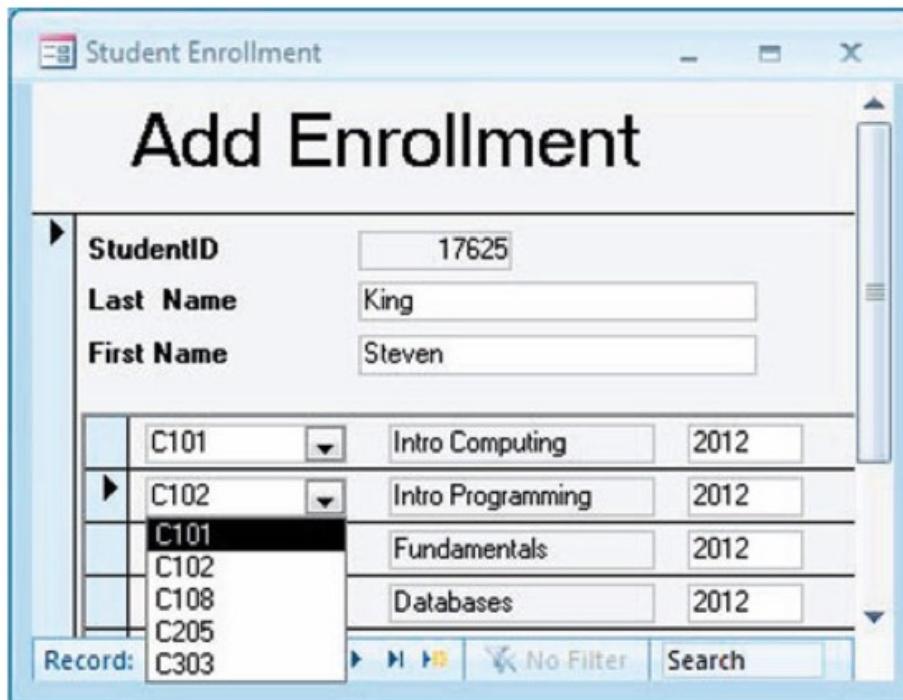
Modellierung der Zugriffsrechte

- Darstellung als Zugriffsmatrix
 - Zugriffsrechte können aus Use Case-Diagramm abgeleitet werden
 - Subjekte sind einzelne Benutzer bzw. Benutzergruppen
 - Objekte können Entitytypen, Entities oder Attribute sein
 - Prinzip des kleinstmöglichen Privilegs:
Welche Rechte benötigt ein Benutzer mindestens, um einen Use Case auszuführen?
 - Zugriffsrechte = {read, write, delete}

Subjekt \ Objekt	Flug	Flug-hafen	Flugbuchung(flugNr, datum kundenNr, von, bis)	Flugbuchung (stornogebühr)
Subjekt				
Kunde	r	r	r, w	r
Sachbearbeiter	r, w, d	r, w, d	r, w, d	r, w

User Interface

- User Interface für Datenfelder von Tabellen
 - Eingabefelder für Attribute
 - Auswahlliste (Listbox) zur Spezifikation von Relationships
 - Berücksichtigung der Kardinalität von Relationships (Combobox)



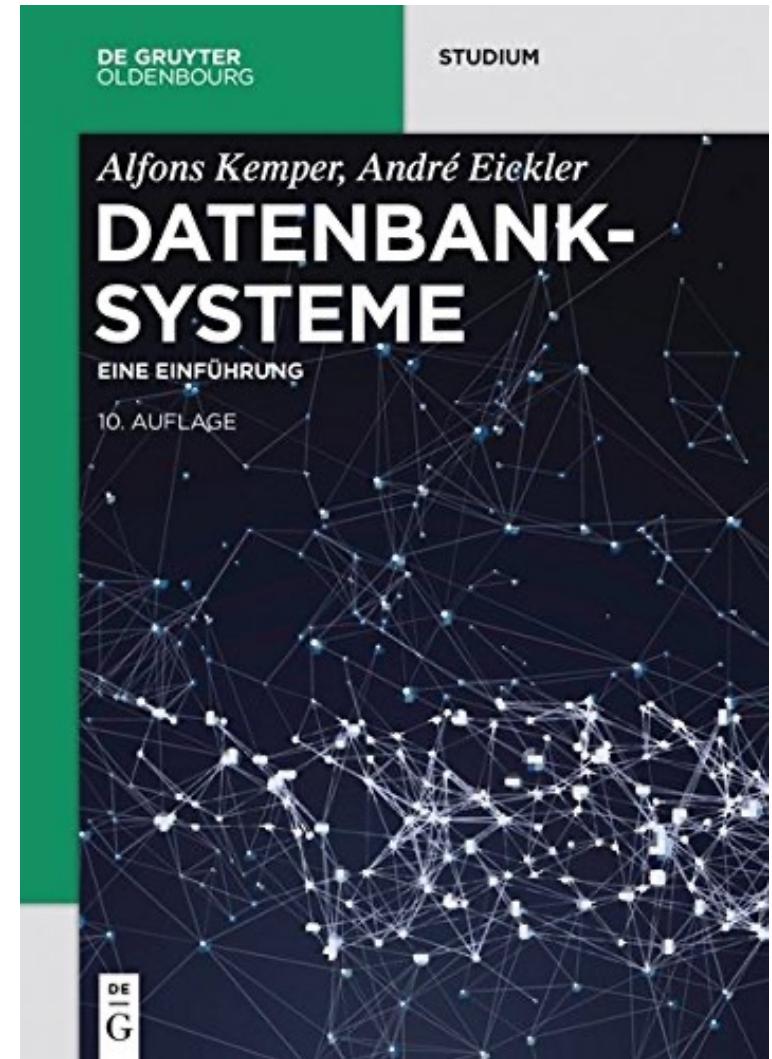
Quelle: C. Churcher: Beginning Database Design

2. Konzeptueller Datenbankentwurf

- Grundbegriffe Modellierung
- Entity-Relationship-Modell
 - Entities
 - Beziehungen
 - Kardinalität

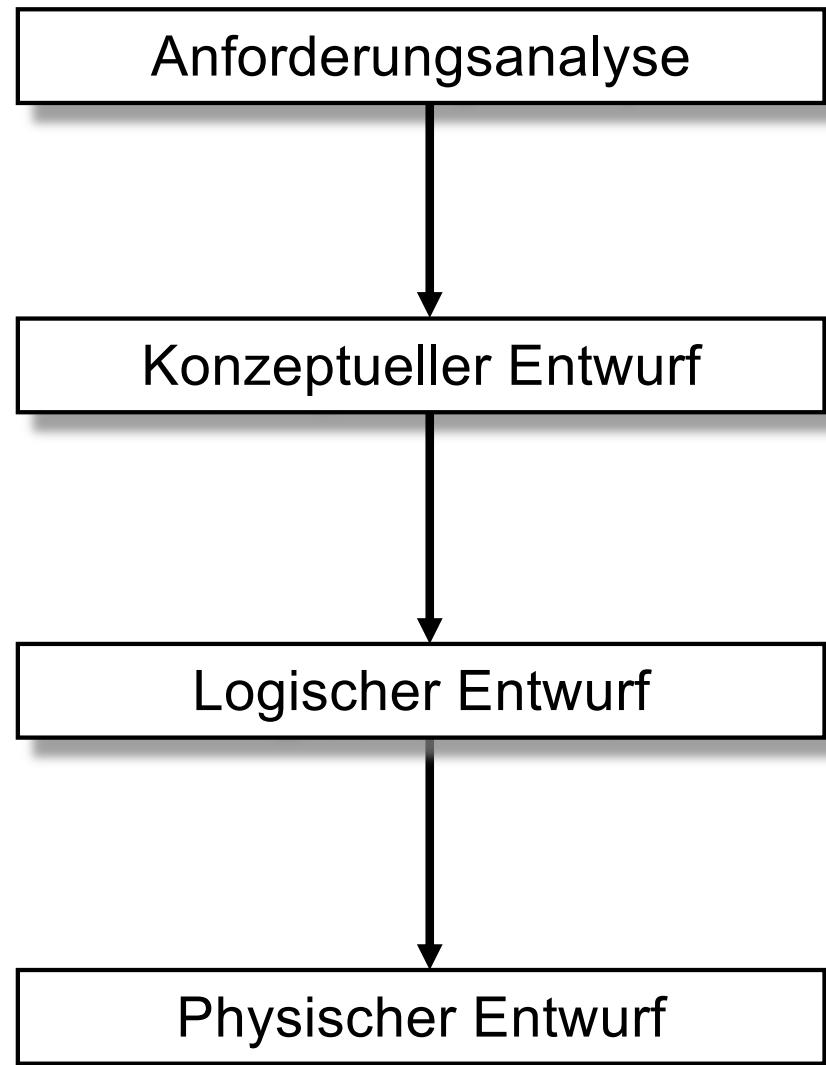
Literaturempfehlung

- A. Kemper, A. Eickler:
Datenbanksysteme – Eine Einführung,
De Gruyter Verlag, 10. Auflage, 2015
- Andreas Gadatsch: Datenmodellierung
– Einführung in die Entity-Relationship-
Modellierung und das
Relationenmodell, Springer, Vieweg, 2.
Auflage, 2019
als EBook in HTWG:Bib vorhanden
- <http://www3.in.tum.de/research/publications/books/DBMS einf/>



Datenbankentwurf

Entwurfsschritte



Sammlung aller für eine Miniwelt bedeutsamen Gegenstände, Eigenschaften, Beziehungen und Operationen

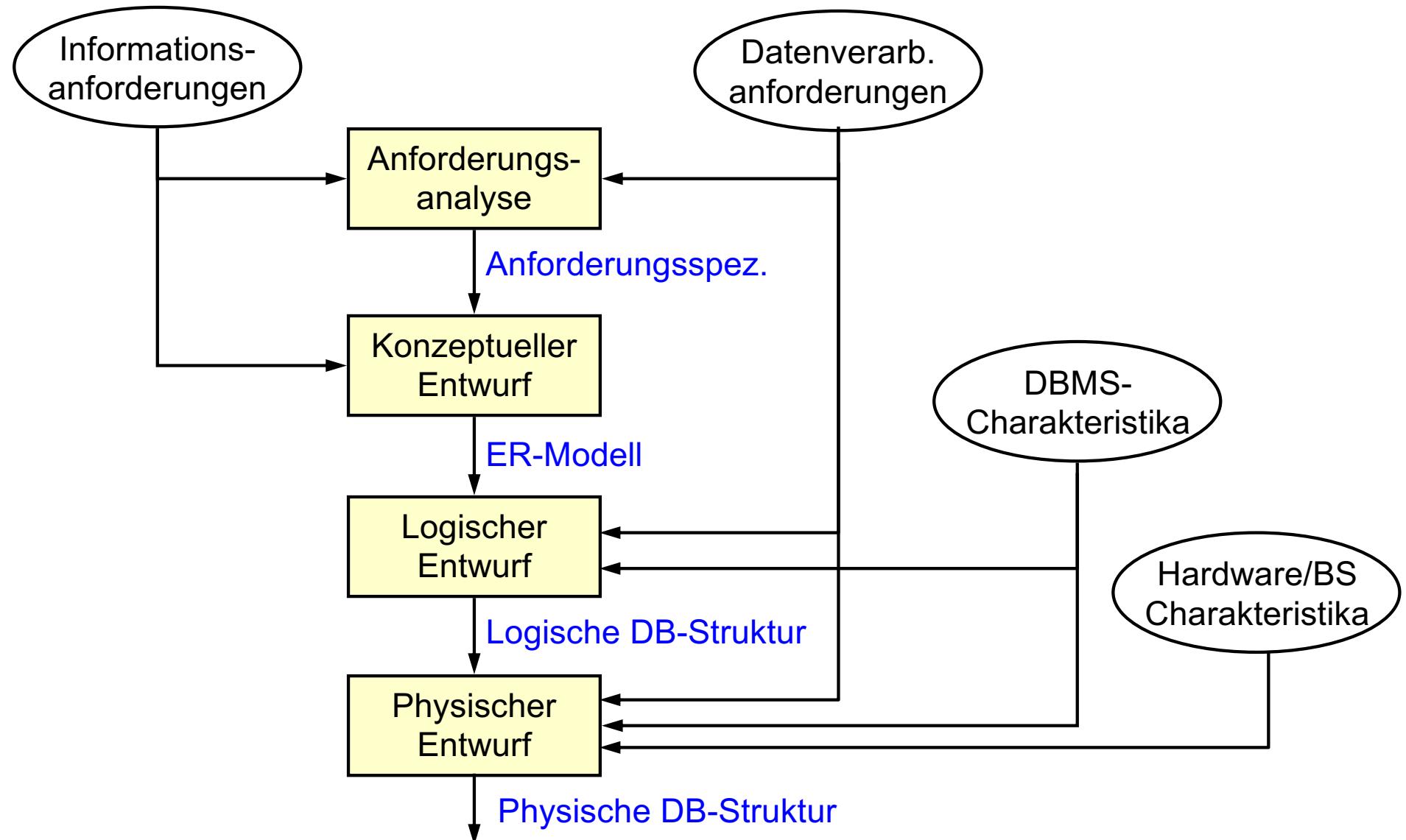
Präzise Beschreibung einer Miniwelt durch relationale oder objektorientierte Modelle

Abbildung auf ein rechnergestützt interpretierbares Schema, z.B. relationales Schema

Abbildung des logischen Datenbankschemas in eine effiziente physische Datenbasisstruktur

Datenbankentwurf

Entwurfsschritte

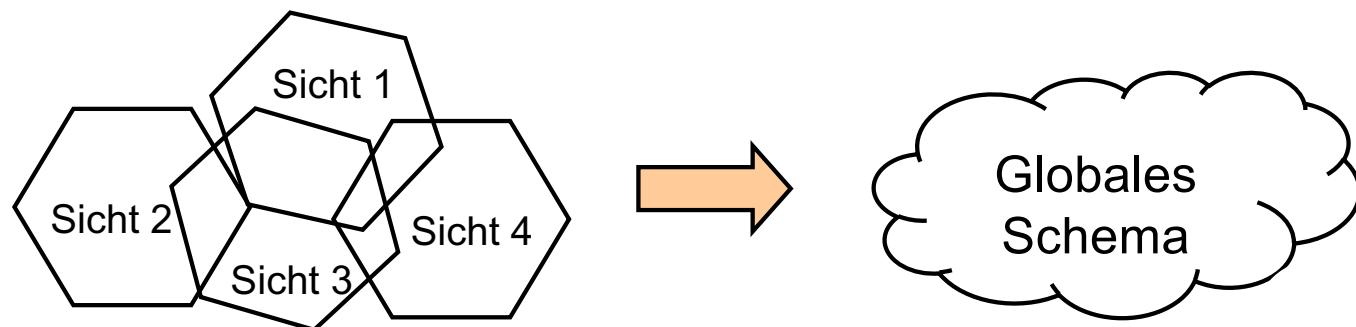


Datenbankentwurf – Ebenen

- Konzeptuelle Ebene
 - Strukturierung des projektierten Anwendungsbereichs
 - Unabhängigkeit von verwendeten Datenbanksystem
 - Verwendung einer Modellierungssprache
 - Entity-Relationship-Modellierung
 - UML
- Implementationsebene
 - Beschreibung der Datenbankanwendung in den Konzepten des verwendeten Datenbanksystems
- Physischer Entwurf
 - Erhöhung der Leistungsfähigkeit (Performance) der Datenbankanwendung
 - Tiefgehende Kenntnisse des Datenbanksystems, des zugrundeliegenden Betriebssystems und sogar der Hardware notwendig

Konsolidierung, Sichtenintegration

- Aufteilung in Anwendersichten bei größeren Anwendungen
- Nach Modellierung der einzelnen Sichten Zusammenfassung zu einem globalen Schema
- Konsolidierung, Sichtenintegration
 - Entfernung Redundanzen
 - Entfernung Widersprüche
 - Bereinigung von Synonyme
Mehrere Worte haben gleiche Bedeutung, Bsp: senkrecht – vertikal
 - Bereinigung von Homonyme
Ein Wort hat verschiedene Bedeutungen, Bsp: Bank, Kiefer



Anforderungsanalyse

- Informationsanforderungen
 - Statische Informationen
 - Z.B. Angaben über Daten, Realwelt-Objekte, deren Typen, charakterisierende Eigenschaften bzw. Attribute mit Wertebereichen
 - Integritätsbedingungen: Konsistenz
- Bearbeitungsanforderungen
 - Dynamische Aktivitäten und Prozesse, welche später auf der Datenbank ablaufen sollen
 - Anfragen, Updates, Auswertungen, etc.
 - Verfügbarkeit, Sicherheit der Daten
- Aktivitäten während dieser Entwurfsphase
 - Identifikation der wesentlichen Benutzergruppen
 - Modellierung der relevanten Daten

Methoden zur Informationsgewinnung

- Interviews
 - Persönliche Befragung der betreffenden Mitarbeiter
- Fragebogen
 - Vorgegebene Fragen an Mitarbeiter verteilen
 - Rücklaufquote und Qualität der Antworten als Problem
- Inventur
 - Analysieren von Unterlagen, z.B. Statistiken, Pläne, Bilanzen, Vordrucke
- Beobachtung
 - Beobachtung von Prozesse

Materialeintrag	Kfz - Mietvertrag
Projekt	Flitz-Auto
Quelle / erhalten von	Hr. Neuland
Empfänger, erhalten am	27. 8. 2003 806
Erklärung	Vertrag und mit alten Logo/ alter CI Firmierung richtig?

Quelle: www.oose.de

Datenverarbeitungsanforderungen

- Behandlung der Verarbeitung der Daten
- Beispiel: Zeugnisausstellung
 - Häufigkeit: halbjährlich
 - Benötigte Daten
 - Prüfungen
 - Studienordnungen
 - Studenteninformationen
 - Priorität hoch
 - Zu verarbeitende Datenmenge
 - 600 Studierende
 - 4000 Prüfungen
 - 5 Studienordnungen

Informationsstrukturanforderungen

- Bestandteile
 - Objekte
 - Attribute dieser Objekte
 - Beziehungen zwischen diesen Objekten
- Beispiel: Objektbeschreibung Angestellter

Beschreibung: Ein Angestellter ist eine Person, die einen zur Zeit gültigen Arbeitsvertrag besitzt.

Attribut Personalnummer

Beschreibung: eindeutige Nummer

Typ: char

Länge: 6

Wertebereich: 0 ... 999.999

Anzahl Wiederholungen: 0

Definiertheit: 100%

Identifizierend: ja

Attribut Gehalt

Beschreibung: monatliches Bruttogehalt

Typ: dezimal

Länge: (8.2)

Anzahl Wiederholungen: 0

Definiertheit: 90%

Identifizierend: nein

Entity Relationship-Modellierung

- P. Chen 1976
- Entity Relationship Diagramme (ERD) sind eine Beschreibungsnotation für statische Beziehungen zwischen “Objekten”
- Ursprünglich eingesetzt für den Datenbankentwurf (relationale Datenbanken)
- Erweiterungen der Original-Notation von Chen
 - Erweiterte Entity Relationship-Modelle (EER-Modelle)
- Übersicht Notation
 - Entitytypen: Abstraktion ähnlicher Gegenstände
 - Relationshiptypen: Beziehungen zwischen Gegenstandstypen
 - Attribut: Charakterisierung von Gegenständen und Beziehungen

Entity

- Definition Entity
 - Individuelles und identifizierbares Exemplar eines Objekts, einer Person, eines Begriffes, eines Ereignisses, ... der realen oder der Vorstellungswelt
- Beispiele
 - Mitarbeiter (Mitarbeiter mit Personalnummer 13334)
 - Auto BMW (KN-LP 56)
 - Angebot vom 7.11.2017 an Herrn Schmidt-Fröhlich
 - Vertrag (mit einer bestimmten Vertragsnummer)
- Ein Entity ist immer eine eindeutig identifizierbare Einheit !
 - identifizierbar durch Schlüssel
 - relevant
 - Informationen werden zum Begriff gesammelt (Attribute)

Entity-Typ

- Oberbegriff für eine Menge von Entities, die gleiche Attribute (nicht Attributwerte) besitzen
- Darstellungsform
 - Darstellung von Entity-Typen als Rechtecke
 - Eindeutiger Name: Substantiv, singular
- Beispiele
 - Maier, Hans, 01.02.1965, Konstanz
 - Müller, Hugo, 12.06.1997, Singen
 - ...
 - Zugehöriger Entity-Typ: Student

Student

Beziehungstypen

- Beziehungstypen
 - Zusammenfassung gleichartiger, d.h. hinsichtlich ihrer Art und der beteiligten Entitytypen übereinstimmende Beziehungen
 - Name: Verb

- Notation



- Entitätstyp (Menge von Objekten)
 - Entity type



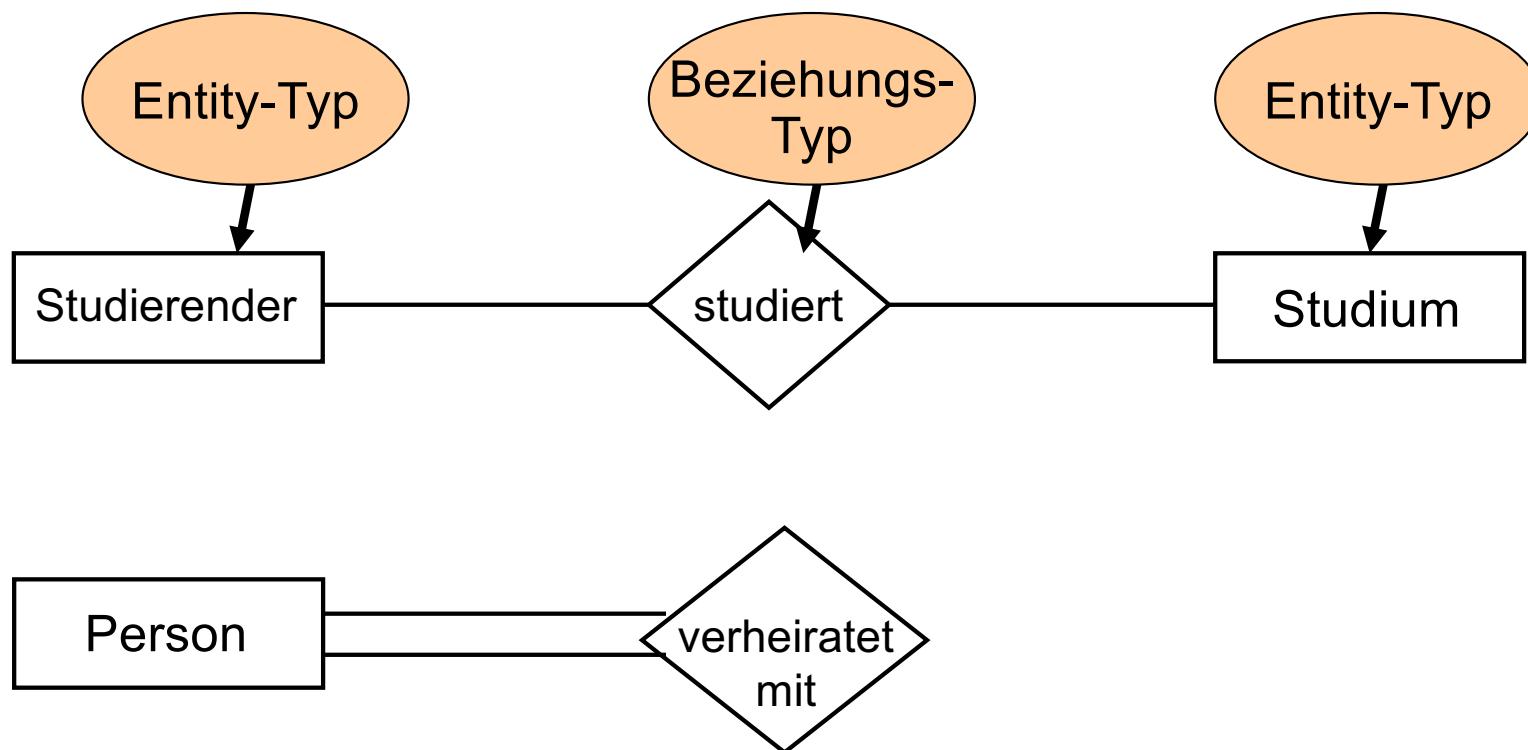
- Beziehungstyp (Assoziation zwischen Entitätstypen)
 - Relationship

m n

- Verbindung zwischen Entitätstypen und Beziehungstypen mit zugehörigen Kardinalitäten

Beziehungstypen

- Beispiele

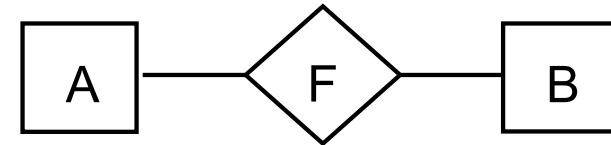


Grad von Beziehungstypen

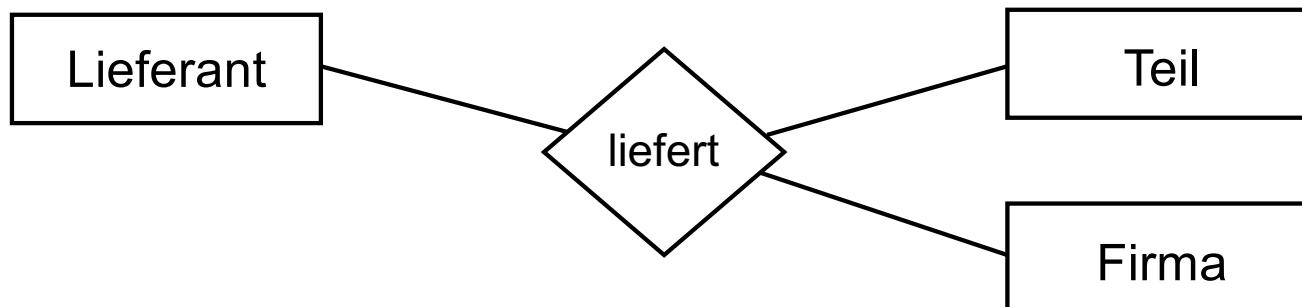
- Grad eines Beziehungstyps F:

$\text{grad}(F) := \text{Anzahl der an } F \text{ beteiligten Entity-Typen}$
„n-stellige Beziehung“

$$F = ((E_1, E_2, \dots, E_n), Y) \quad \Rightarrow \quad \text{grad}(F) = n$$



- Beispiel einer dreistelligen Beziehung
 - Häufig bessere Modellierung durch Koppelobjekte



Kardinalität von Beziehungstypen

- Kardinalität von Beziehungstypen

$\text{kard}(F, A) :=$ Anzahl der Beziehungen $f \in F$, an denen ein Entity $a \in A$ beteiligt sein kann

- Mehrere Notationen für Kardinalitäten

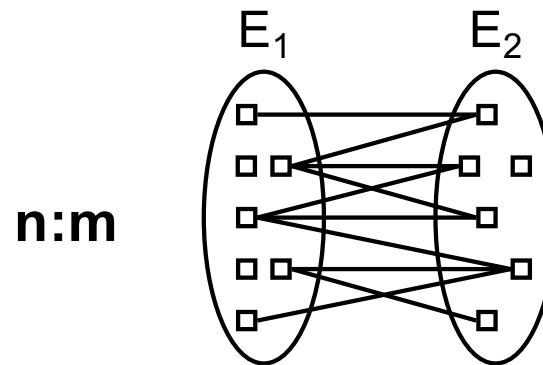
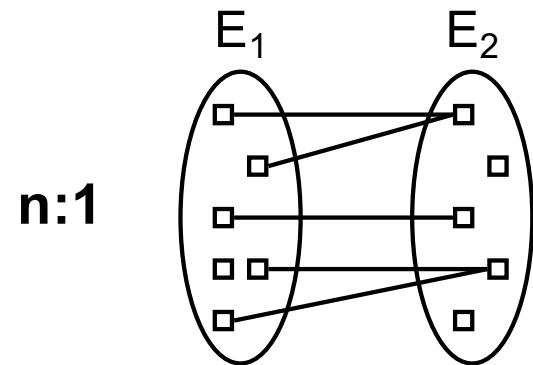
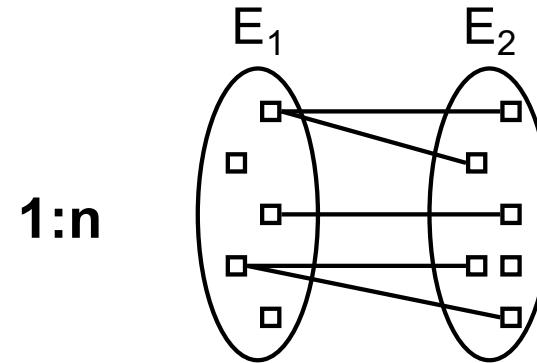
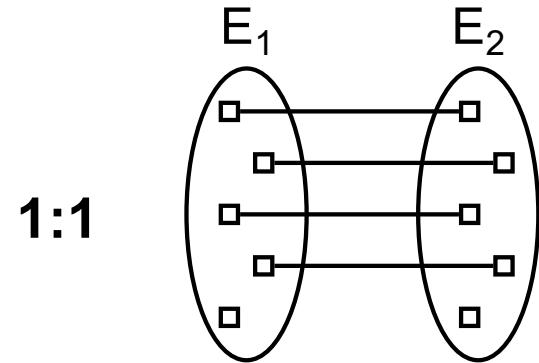
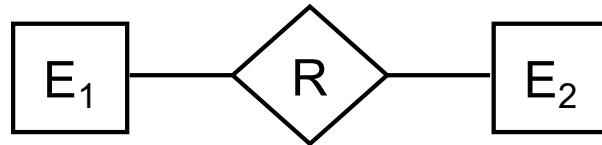
- 1:n-Notation
- Min-Max-Notation
- etc.

- 1:n-Notation

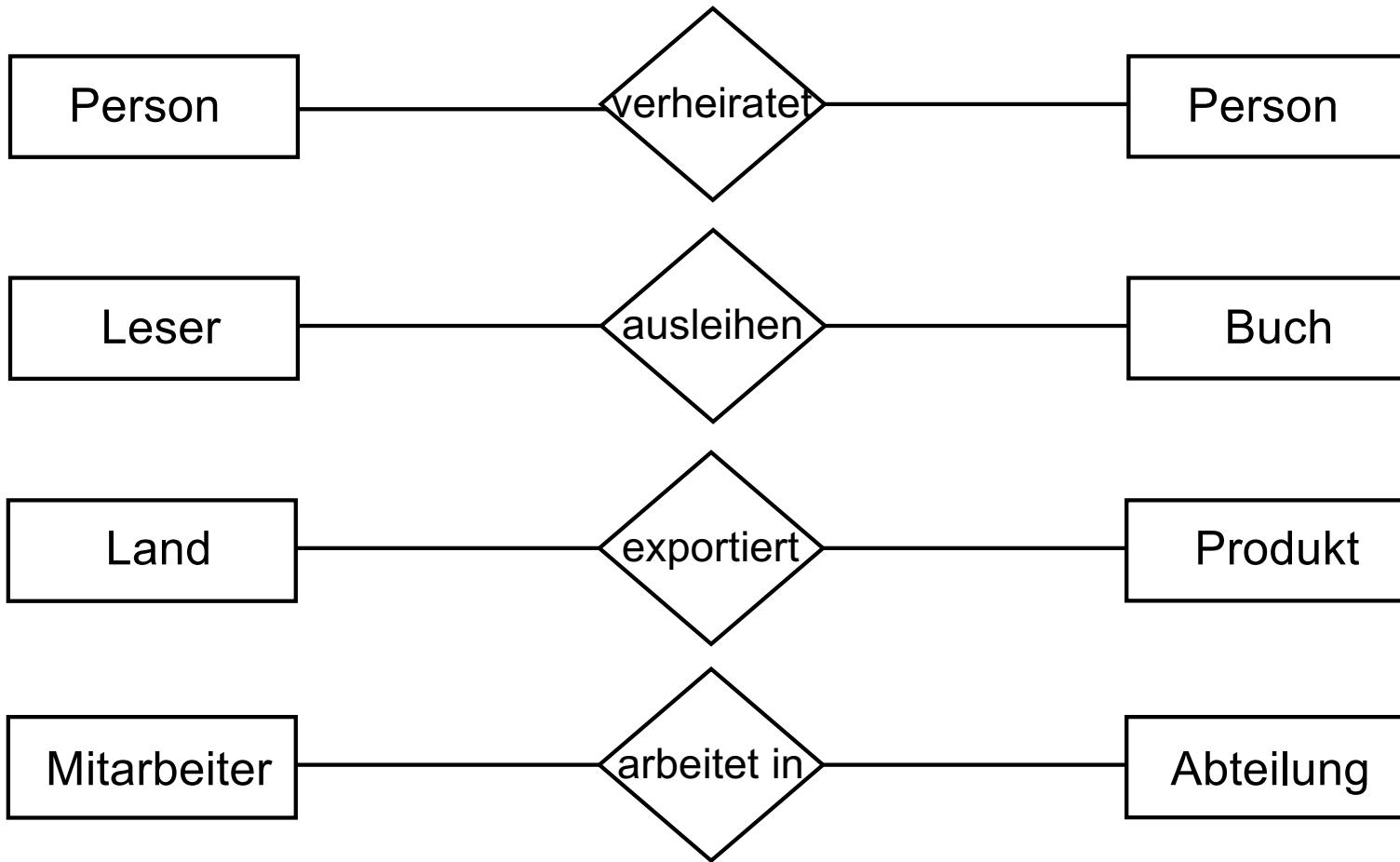
- Kardinalitäten: 1:1, 1:n, n:m

Grafische Veranschaulichung der Kardinalität

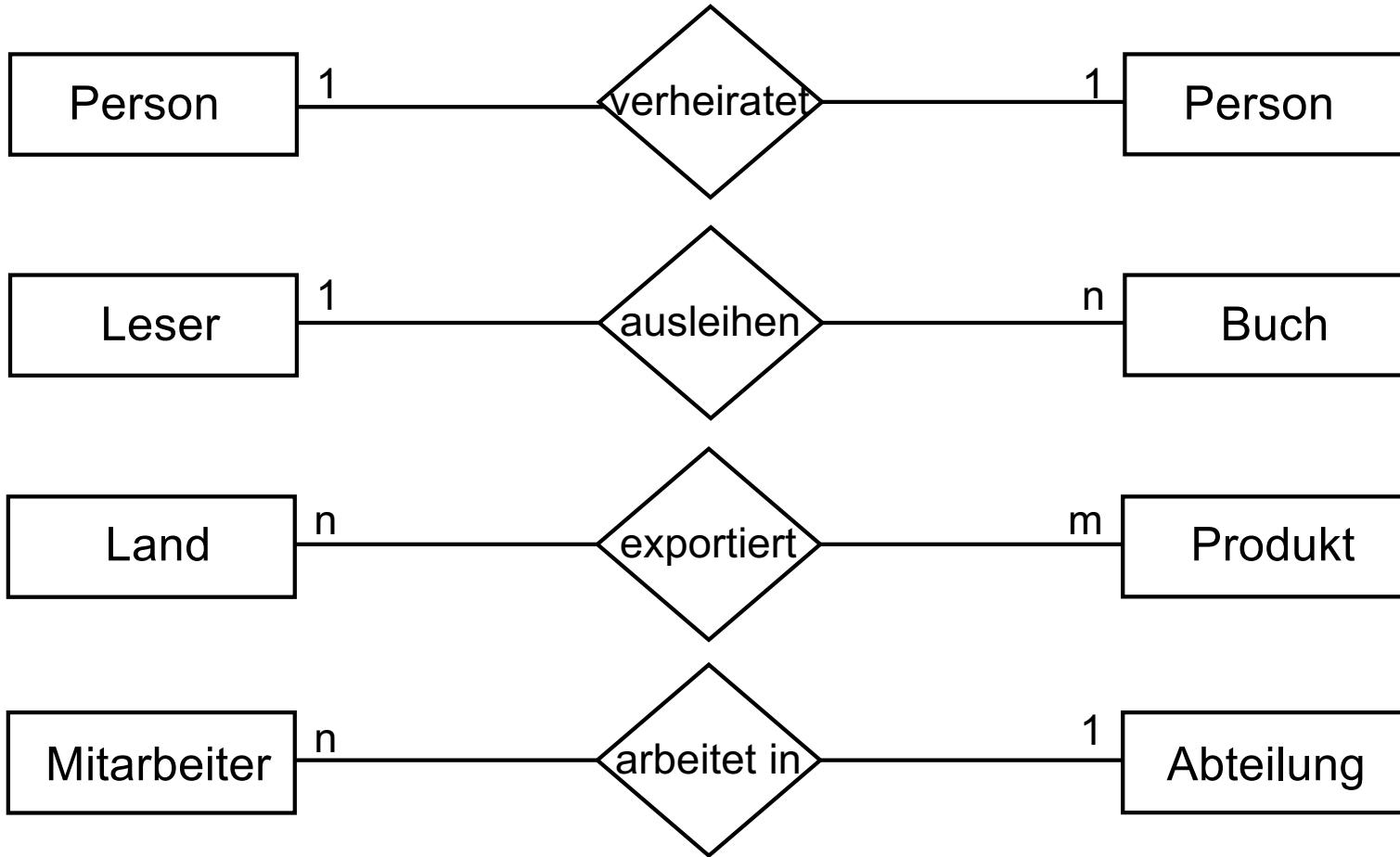
1:n Notation



Beispiele für Kardinalitäten



Beispiele für Kardinalitäten



Kardinalität von Beziehungstypen

- Min-Max-Notation

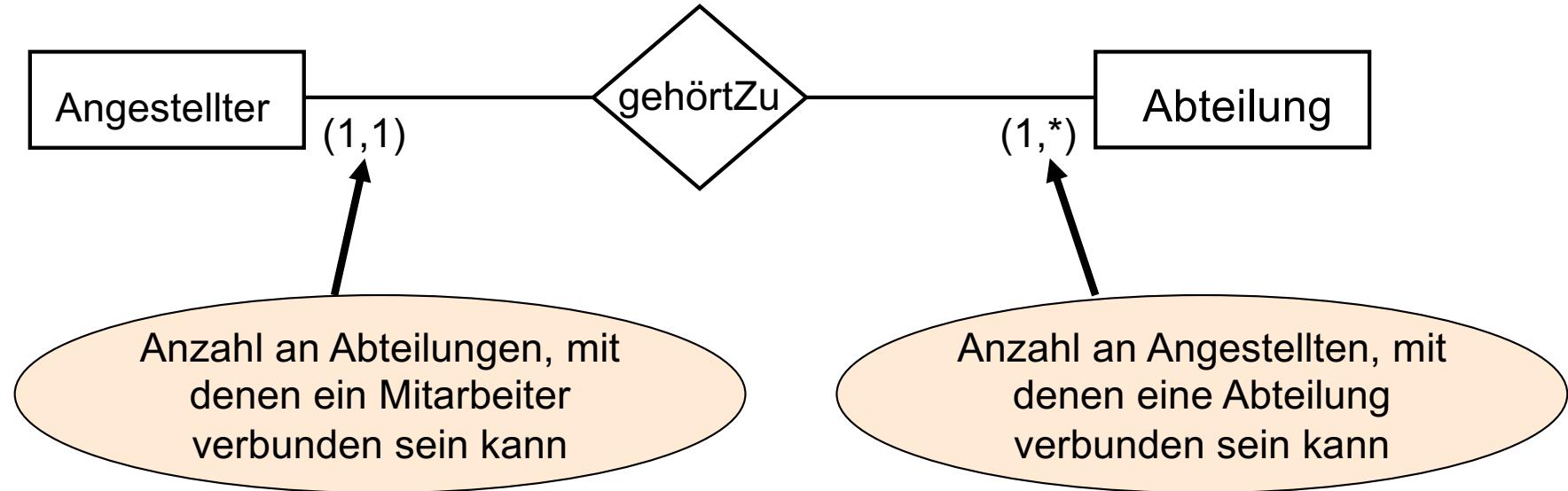
$\text{kard}(F,A) = (\alpha, \beta)$ α : Mindestanzahl der konkret vorhandenen Beziehungen

β : Höchstanzahl der konkret vorhandenen Beziehungen

 *: Symbol für beliebig viele Beziehungen

Kardinalität von Beziehungstypen

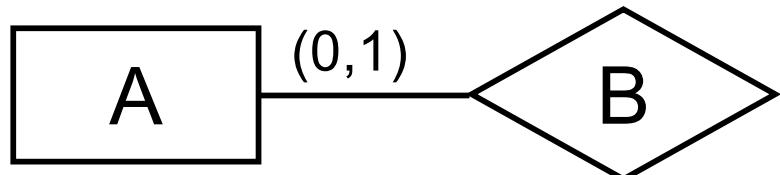
Min-Max-Notation



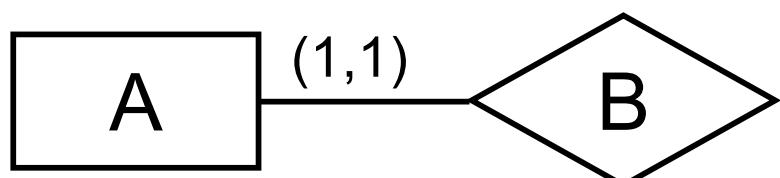
- $\text{kard}(\text{gehörtZu}, \text{Angestellter}) = (1, 1)$
bedeutet, daß jeder Angestellte zu genau einer Abteilung gehört
- $\text{kard}(\text{gehörtZu}, \text{Abteilung}) = (1, *)$
bedeutet, daß zu einer Abteilung mindestens ein Angestellter gehört

Kardinalitäten

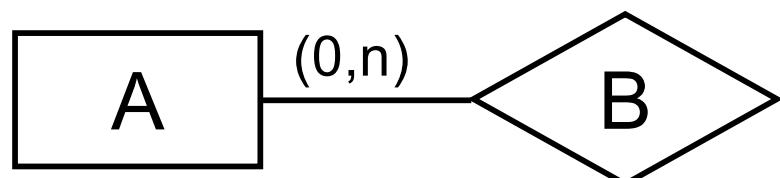
Min-Max-Notation



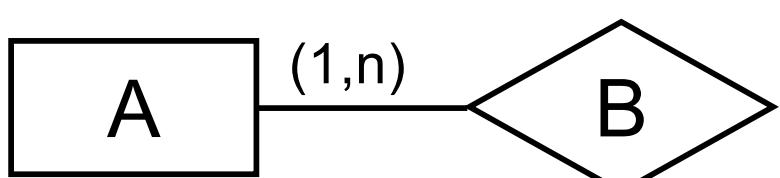
- Kann-Beziehung, einfach



- Muss-Beziehung, einfach

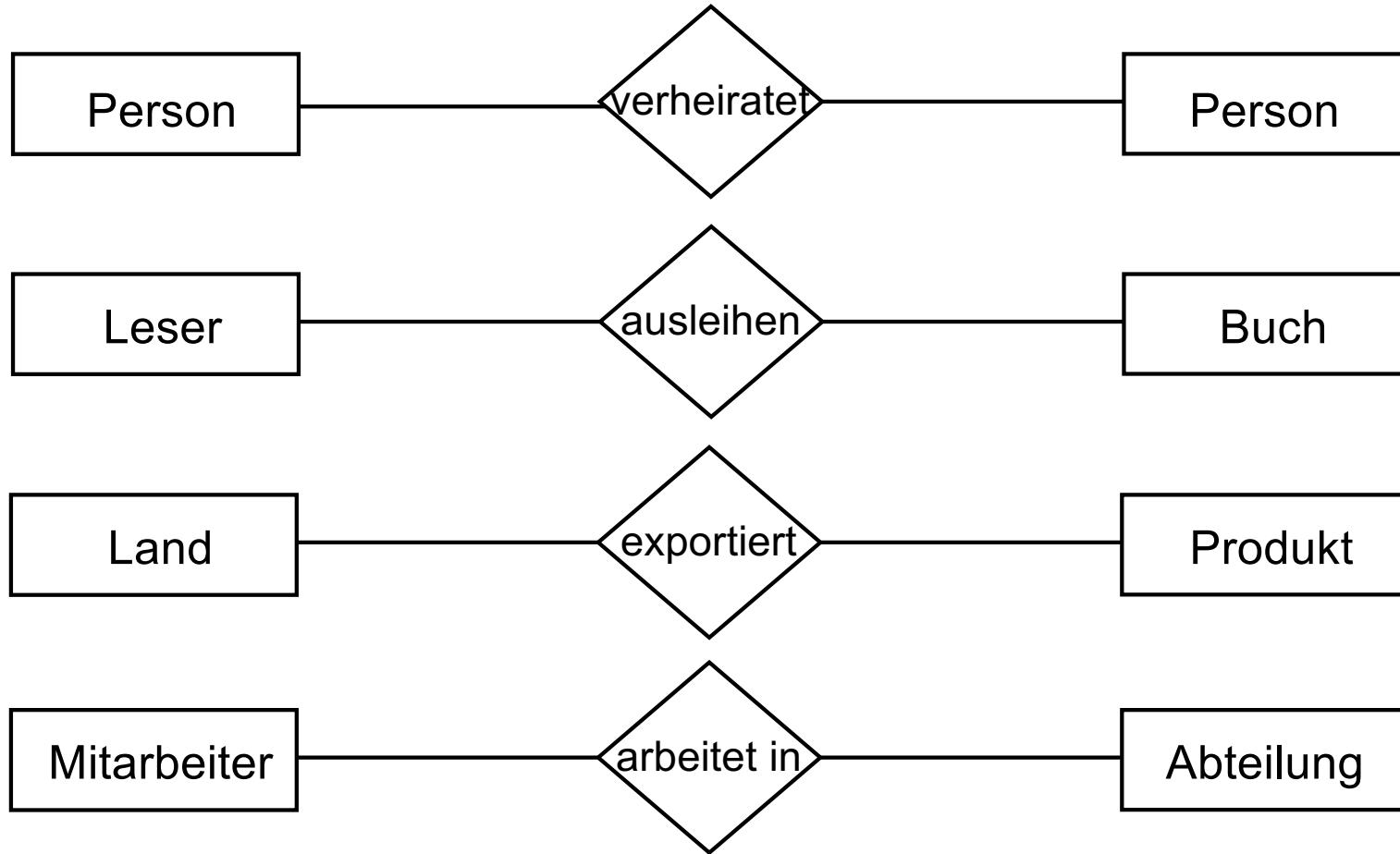


- Kann-Beziehung, mehrfach

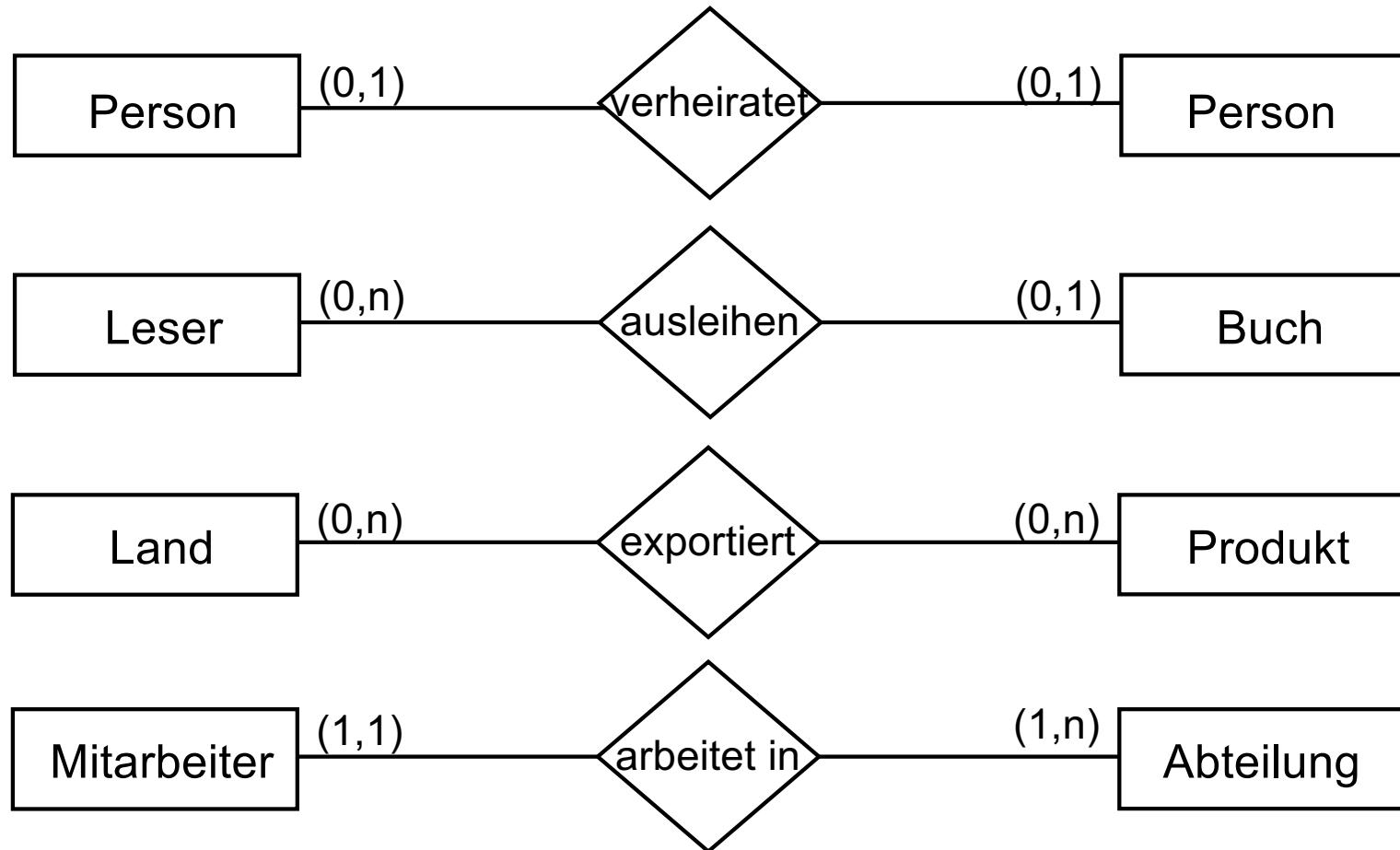


- Muss-Beziehung, mehrfach

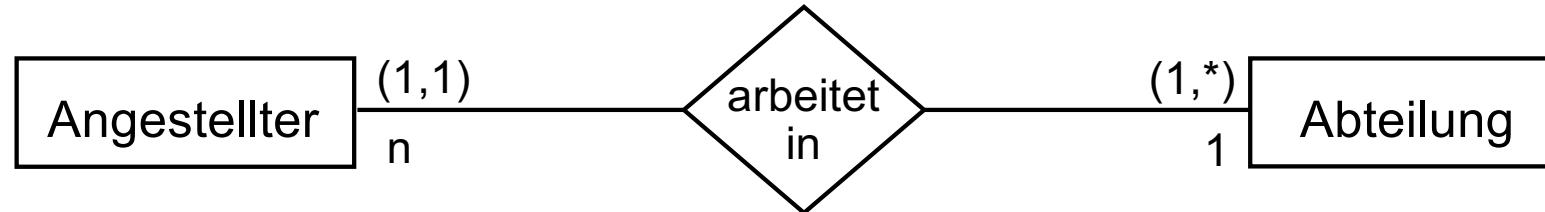
Kardinalität einer Beziehung



Kardinalität einer Beziehung



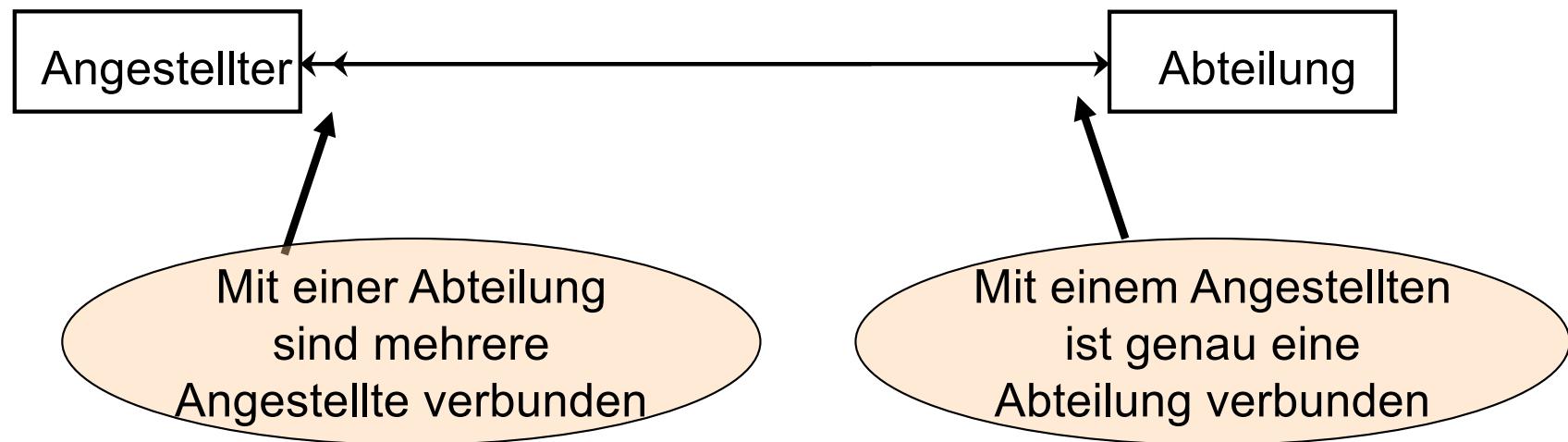
Darstellungsmöglichkeiten für Kardinalität



- Beschränkung auf zweistellige Beziehungen
- Verwendete Kardinalitäten: (m:n), (1:n), (m:1), (1:1)
- Zusammenhang zu Min-Max-Notation

E_1		E_2
(0,1) (1,1)	1:1	(0,1) (1,1)
(0,*) (1,*)	1:n	(0,1) (1,1)
(0,*) (1,*)	n:m	(0,*) (1,*)

Weitere Darstellungsmöglichkeiten für Kardinalität



Entity Eigenschaften

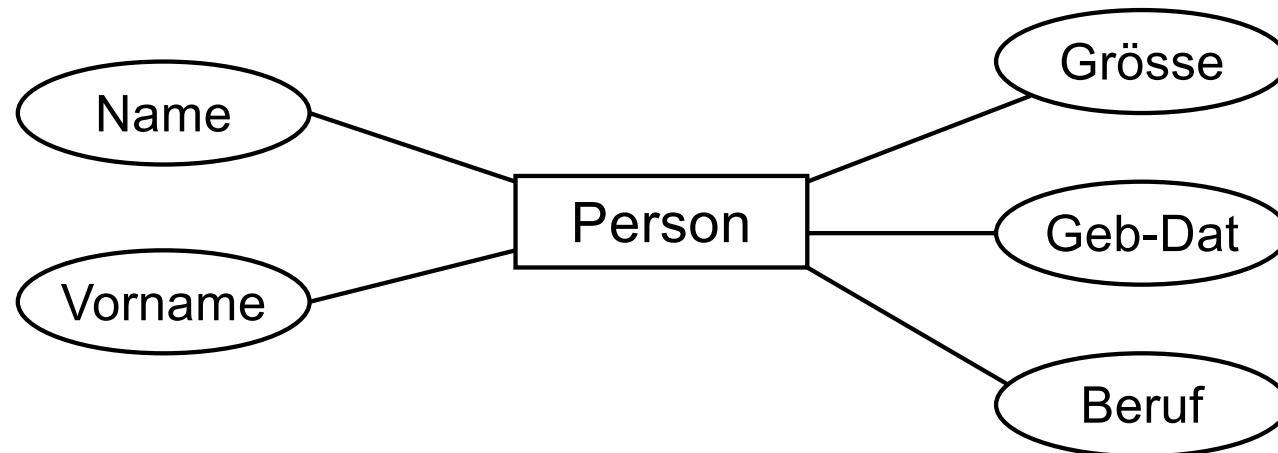
- Eine Eigenschaft wird Entities zugeordnet und ermöglicht damit deren
 - Charakterisierung
 - Eindeutige Identifizierung
- Eine Eigenschaft hat einen Namen und einen Wert
 - NAME → Müller
 - VORNAME → Hans
 - GEB.DAT → 01.02.1965
 - GRÖSSE → 180
 - BERUF → Schreiner

Attribut

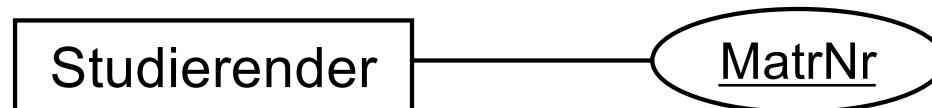
- Attribute tragen die relevanten Informationen und Eigenschaften von Entity- und Beziehungsmengen
- Attributnamen sollten etwas über den Inhalt aussagen
 - „Datum“ ist als Attributname nicht geeignet
 - Besser: Buchungsdatum, Stornierdatum, etc.
- Entität Student
 - Matrikelnummer
 - Name
 - Telefon
 - Geburtsdatum
- Klassifizierung
 - Schlüssel
 - Mehrwertige Attribute

Modellierung von Attributen

- Darstellung als Ellipsen

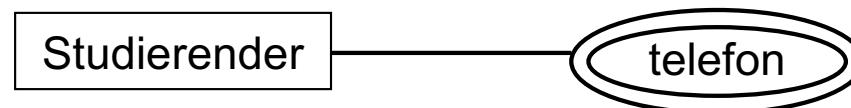


- Schlüsselattribut
 - Ein **Schlüsselattribut** identifiziert ein Tupel eines Entities eindeutig
 - Darstellung durch Unterstreichung des Attributnamens

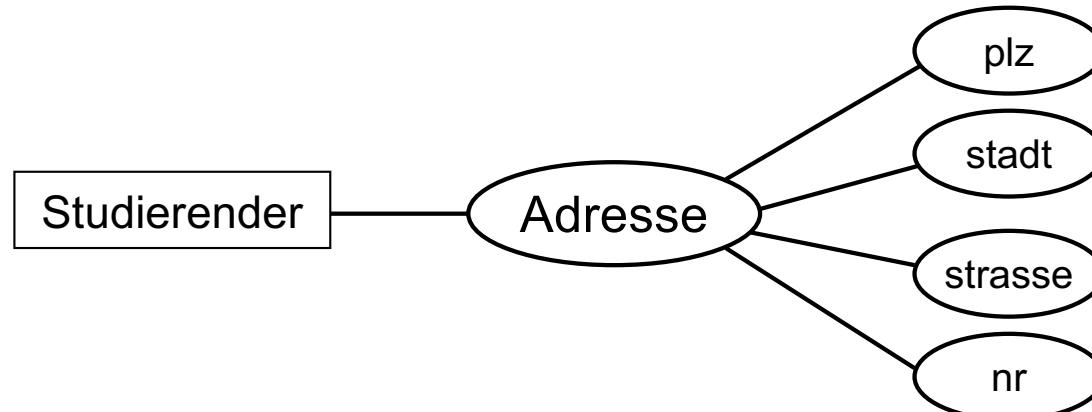


Attribute

- Mehrwertige Attribute
 - Ablegen von mehreren Attributwerten
 - Darstellung durch Doppelkreis

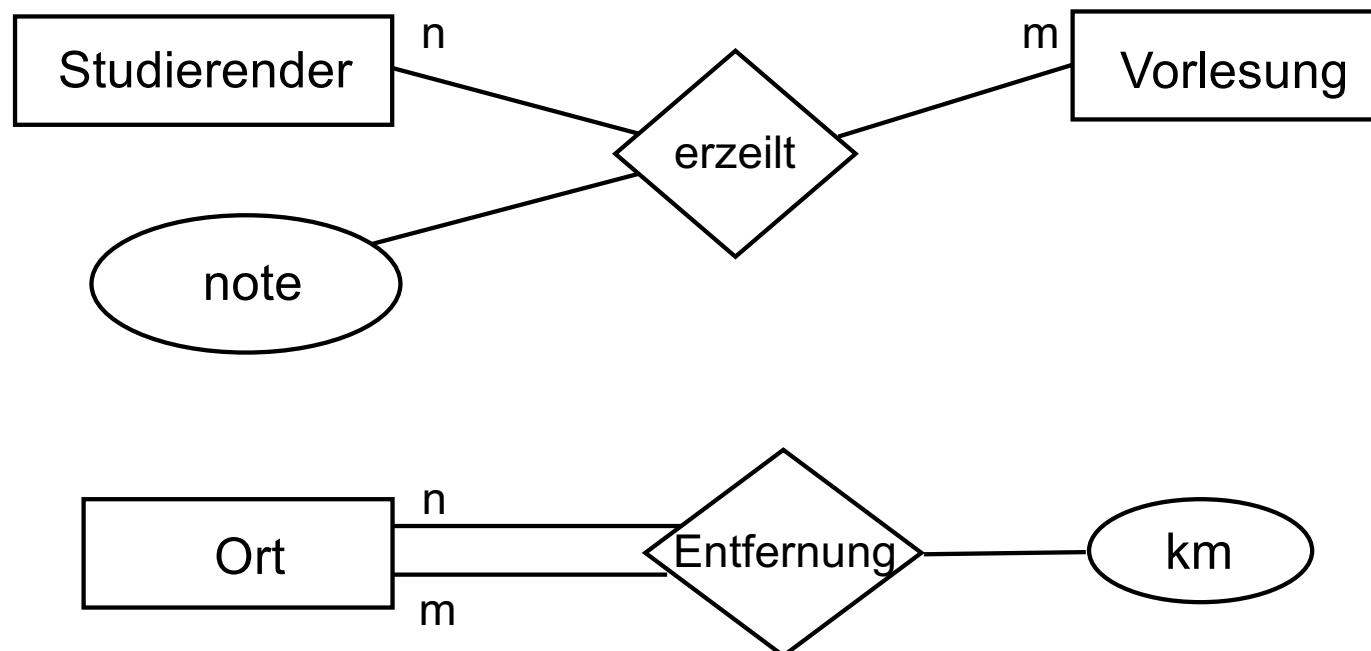


- Zusammengesetzte Attribute
 - Ein Attribut setzt sich aus mehreren anderen zusammen



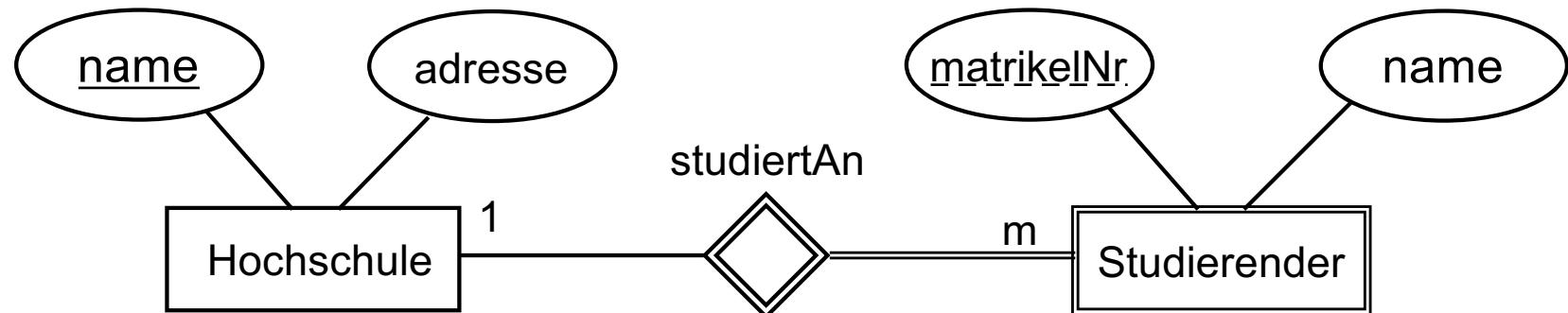
Attribute von Beziehungen

- Zuordnung von Attributen zu Beziehungen
 - Detaillierte Beschreibung der Beziehung
 - Ergänzende Eigenschaften der Beziehung
 - Nur sinnvoll bei n:m-Beziehungen

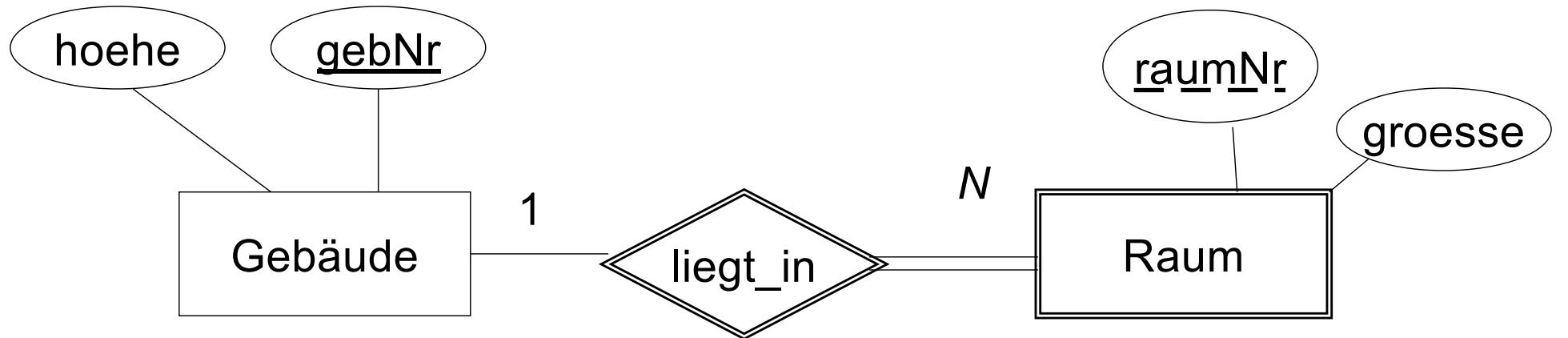


Schwache Entitytypen

- Schwache (existenzabhängige) Entities
 - Entities, die nicht autonom existieren, sondern nur in Verbindung mit einem anderen Entity
 - Nur in Kombination mit übergeordneten Entity eindeutig identifizierbar
 - Darstellung durch doppelte Linie
 - Partieller Schlüssel: Unterstreichung mit gestrichelter Linie
 - Identifizierende Relationship: Darstellung durch doppelte Linie



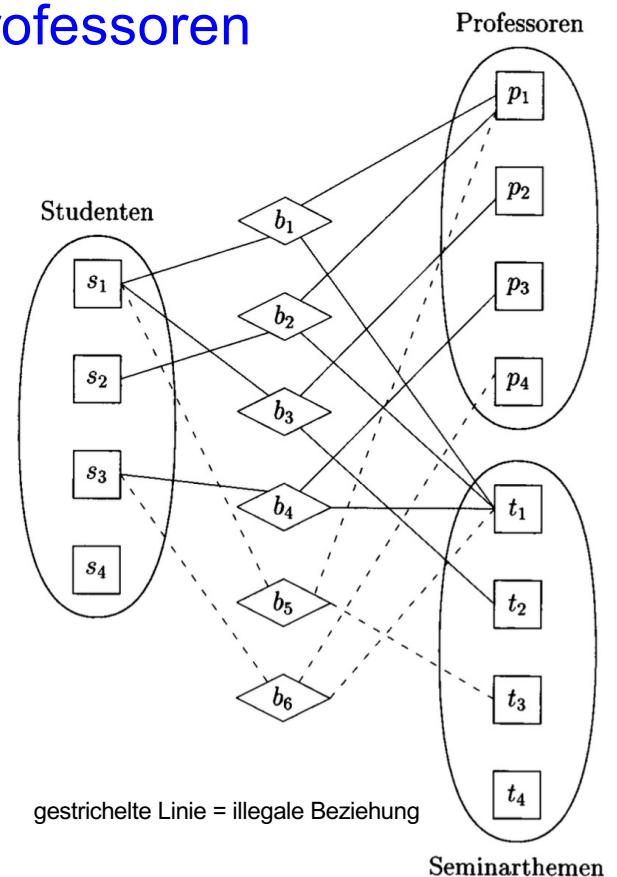
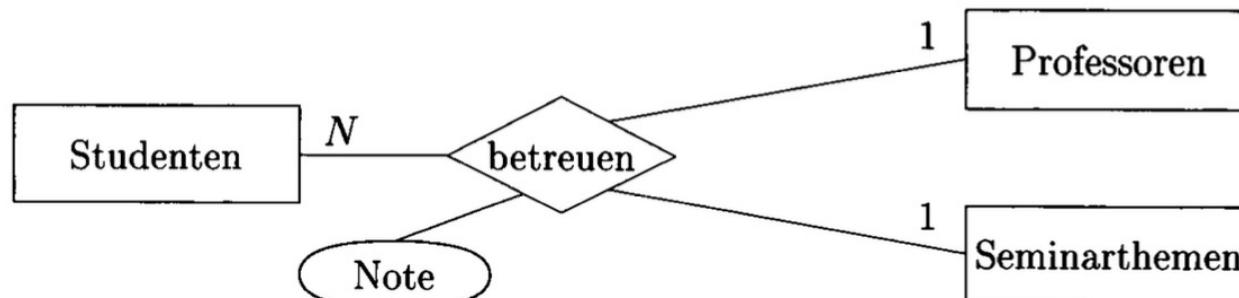
Beispiel schwache Entities



Quelle: Kemper: Datenbanksysteme

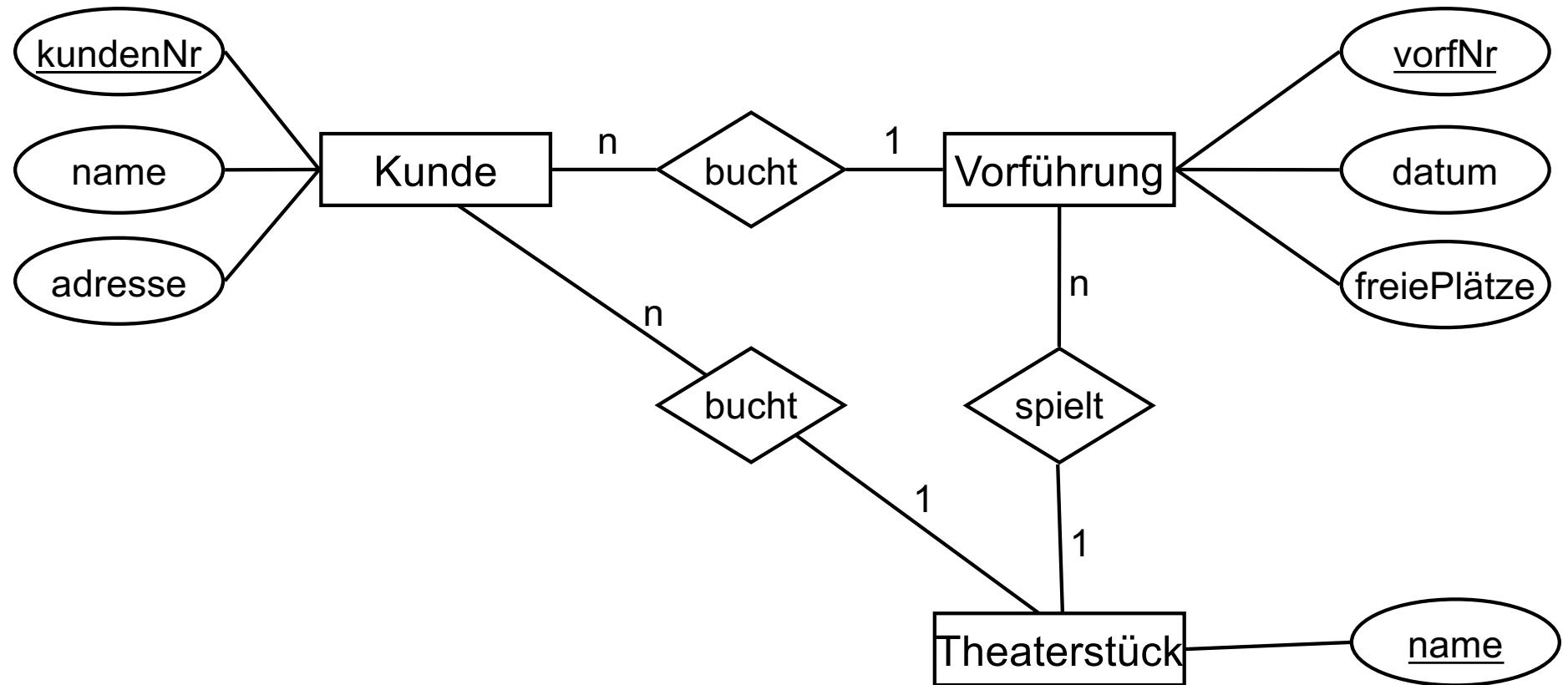
Dreistellige Beziehungen

- Beispiel für Kardinalitäten
 - *betreuen*: Professoren × Studenten → Seminarthemen
 - *betreuen*: Seminarthemen × Studenten → Professoren



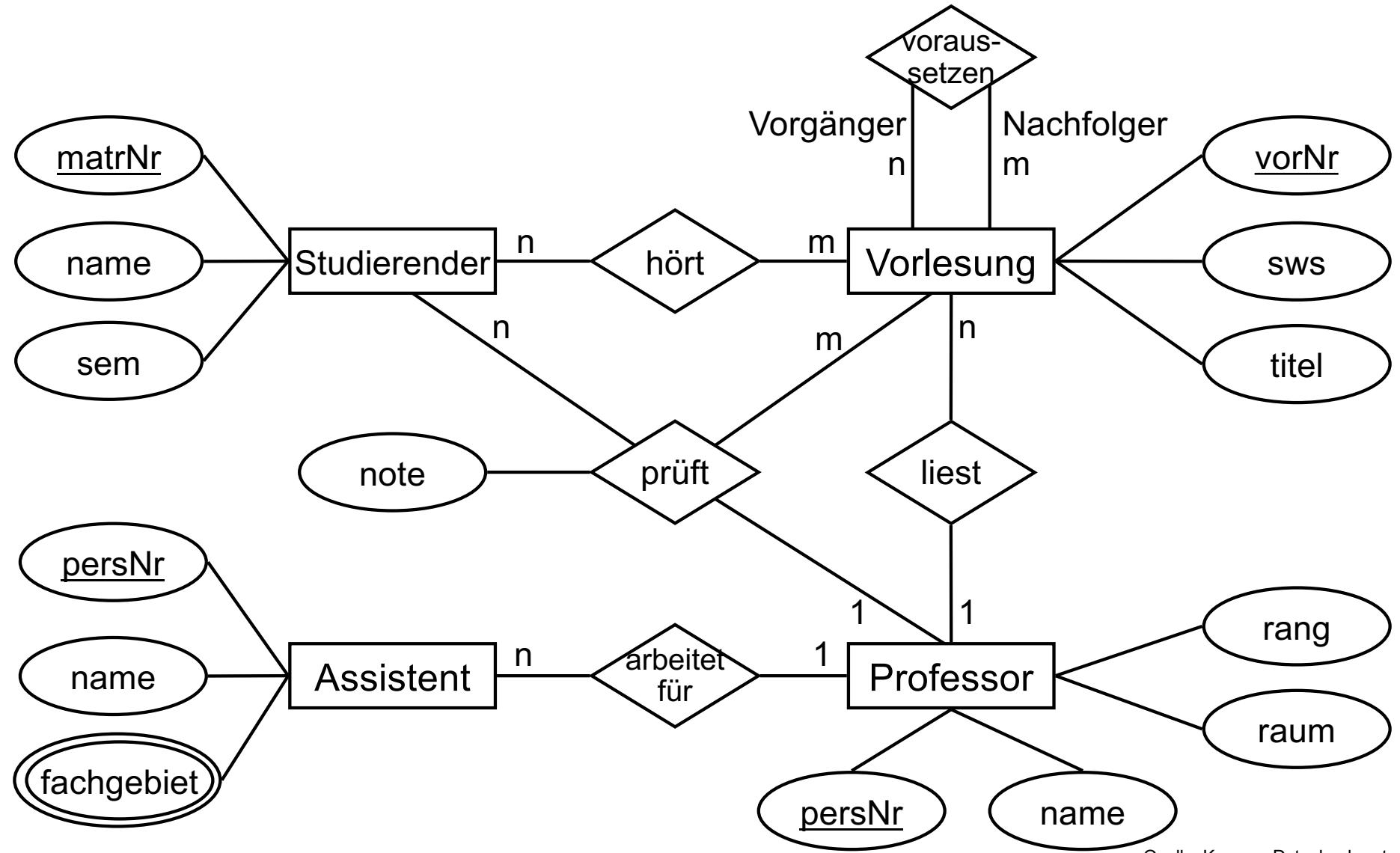
Quelle: Kemper: Datenbanksysteme

Ein korrektes Beispiel?



ER-Modellierung

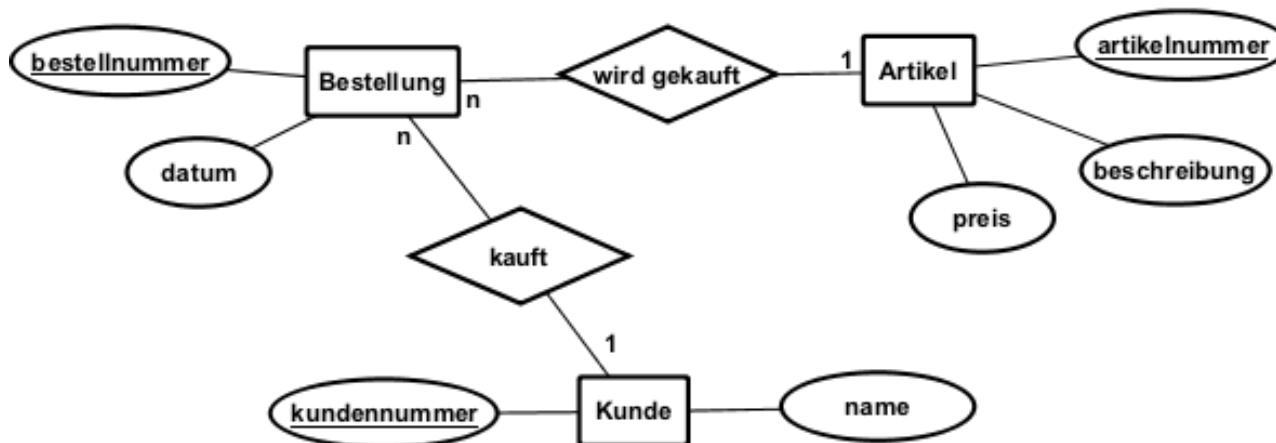
Ein korrektes Beispiel ???



Quelle: Kemper: Datenbanksysteme

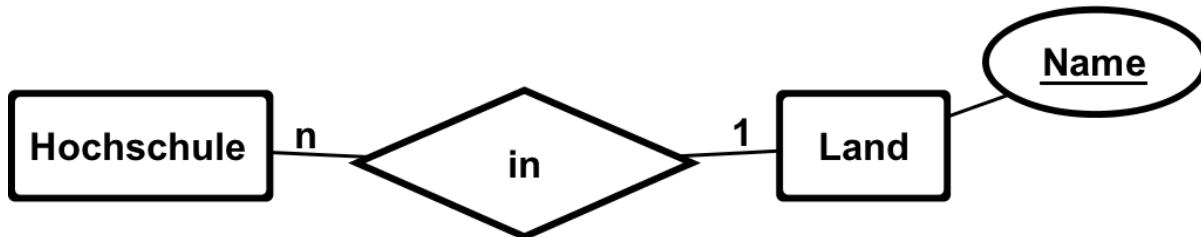
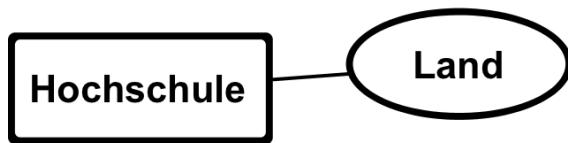
ER-Modellierung

- Attribute sollten zu ihrem zugehörigen Entitytyp modelliert werden



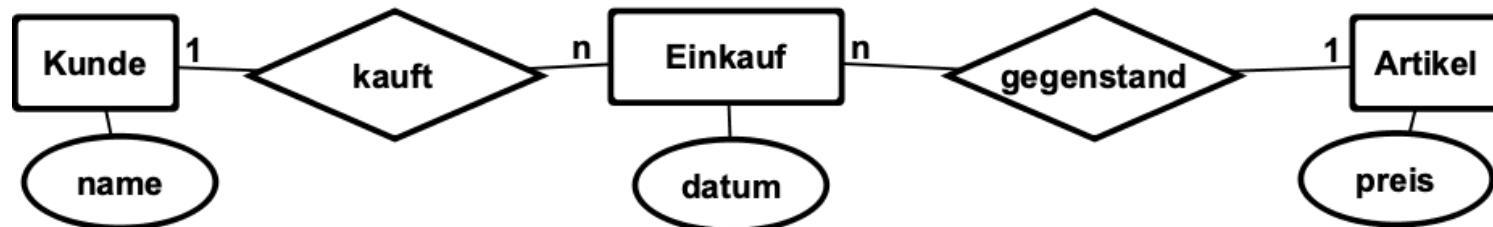
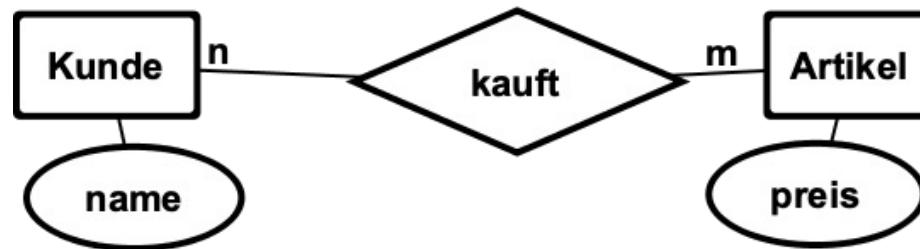
Beispiel ER-Modell

- Welche Modellierung ist besser?



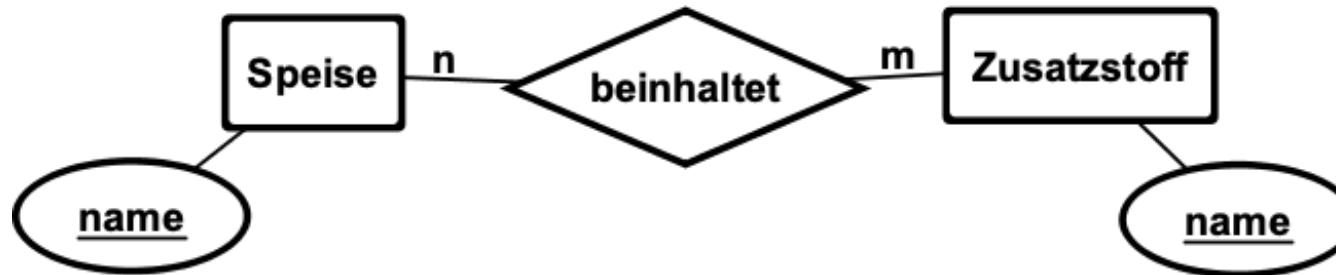
n-m Relationship vs. zwei 1-n Relationships

- Unterschied
 - Im ersten Modell gibt es höchstens **eine** Beziehung zwischen einem Kunden und einem Artikel
 - Im zweiten Modell kann es **mehrere** Käufe zwischen einem Kunden und einem Artikel geben

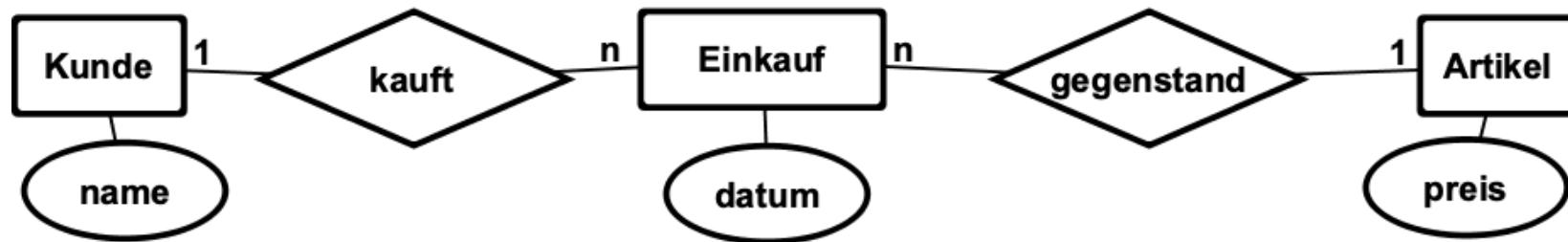


n-m Relationship vs. zwei 1-n Relationships

- Korrekte Modellierung mit n-m Relation



- Korrekte Modellerung mit Relationsobjekt



ER-Modellierung

Häufige Fehlerquellen in Prüfungen

- Attribute modellieren, nicht Attributwerte
- Keine Modellierung künstlicher Attribute
- Entities ohne Attribute sind meist sinnlos
- Beziehungen nicht als Attribute modellieren
 - Wenn „Land“ als Entity modelliert wurde, sollte z.B. Nationalität als Relationship zu Land modelliert werden
- „Rekursive“ Beziehungen
 - Vorlesungen haben eine eindeutige Vorlesungsnummer und einen Titel. Eine Vorlesung kann Voraussetzung für andere Vorlesungen sein, während andere Vorlesungen Voraussetzung für diese Vorlesung sein können

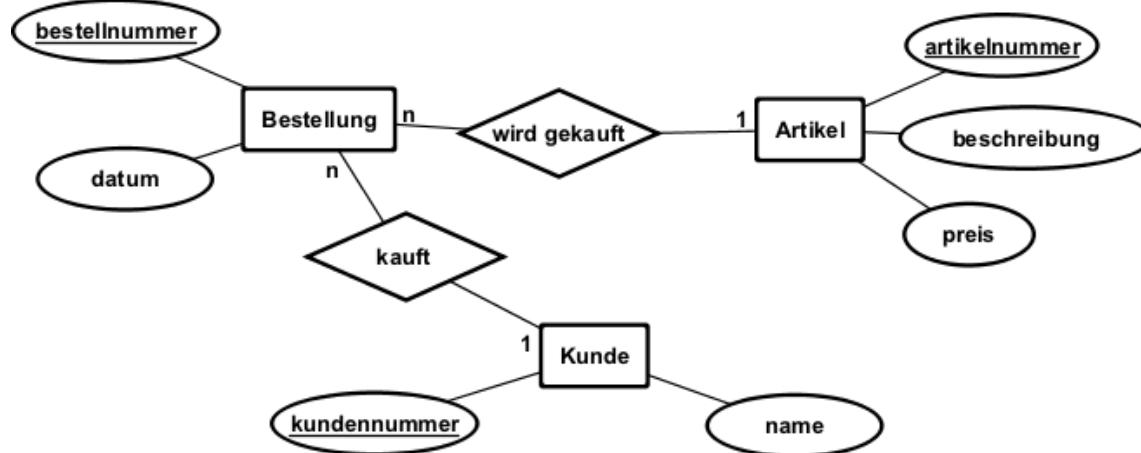
ER-Modellierung

Häufige Fehlerquellen in Prüfungen

- Wahl des Schlüsselattributs
 - In einem Programmkino findet täglich um 20 Uhr eine Vorführung eines Films statt. Für jede Vorführung soll die Anzahl der aktuell noch freien Plätze abgelegt werden. Filme haben eindeutige Titel.
- Zuordnung von Attributen
 - Es wurde bereits „Wein“ und „Flasche“ modelliert und es soll der Preis modelliert werden
- Relevanz
 - Ist es für die Anwendung relevant, dass ein Kunde mehrere Bankverbindungen hat?

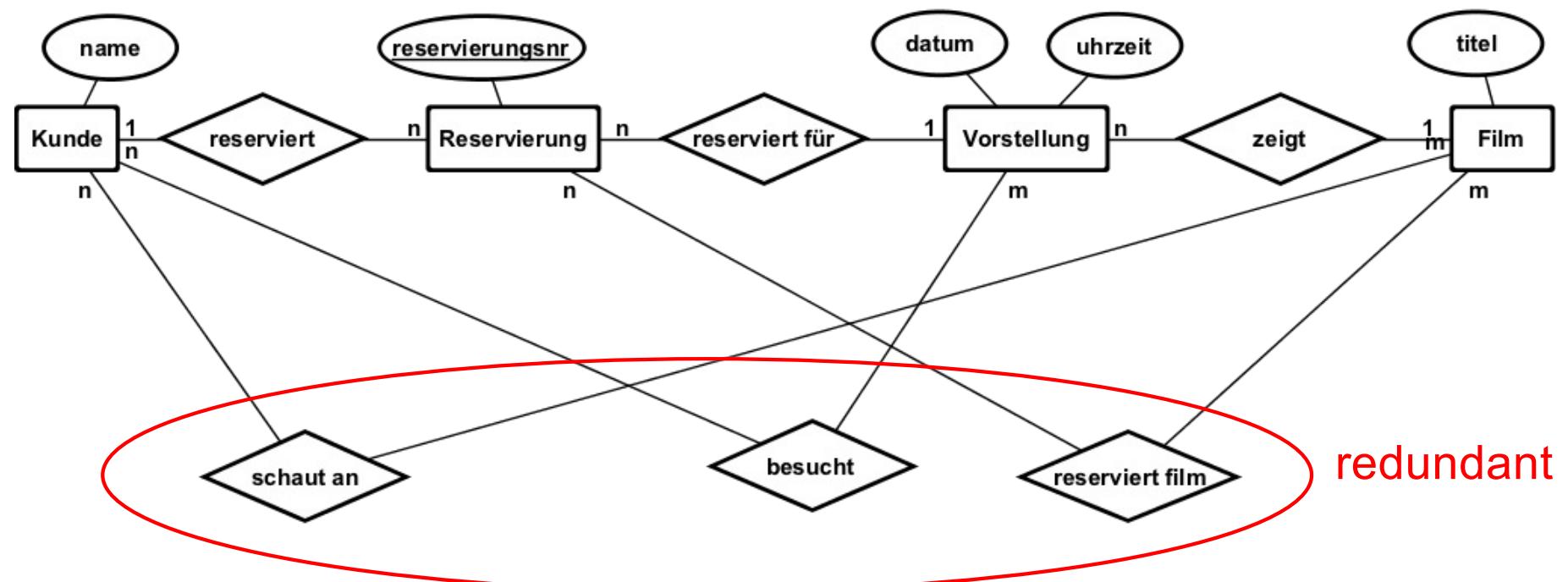
Typische Modellierungsfehler

- Attribute sollten zu ihrem zugehörigen Entitytyp modelliert werden



Typische Modellierungsfehler

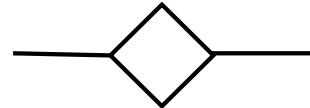
- Beziehungen nicht mehrfach modellieren



ER-Symbole Zusammenfassung



Entity



Relationship



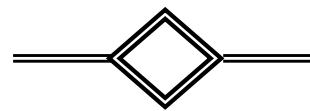
Attribut



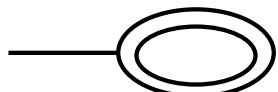
Schwaches Entity



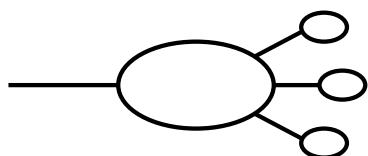
Schlüsselattribut



Identifizierende Relationship



Mehrwertiges Attribut



Zusammengesetztes Attribut

ER-Modellierung – Beispiele

Ein Filmstudio benötigt ein Datenbanksystem zur Verwaltung der im Studio gedrehten Filme und ihrer Mitwirkenden. Dabei soll folgender Sachverhalt gespeichert werden:

Für jeden Film sollen die beteiligten Schauspieler, der Regisseur, sowie der Titel, der Drehort und das Jahr abgelegt werden. Schauspieler können auch Regisseure sein. Für Schauspieler und Regisseure werden Name und Adresse gespeichert. Außerdem soll gespeichert werden, welche Gage ein Schauspieler für jeden gedrehten Film erhalten hat.

Damit Filme in Ländern verschiedener Sprache aufgeführt werden können, werden diese durch Synchronsprecher übersetzt. Damit ein Schauspieler in verschiedenen Filmen immer die gleiche Stimme hat, wird pro Sprache immer der gleiche Synchronsprecher für einen Schauspieler eingesetzt. In dem Datenbanksystem soll also gespeichert werden, welchen Synchronsprecher ein Schauspieler in welcher Sprache besitzt.

Damit ermittelt werden kann, wie erfolgreich ein Film ist, soll gespeichert werden, wie viel Euro der Film in einzelnen Ländern eingespielt hat.

Stellen Sie den beschriebenen Realweltausschnitt im erweiterten Entity-Relationship-Modell graphisch dar. Verwenden Sie dabei Kardinalitätsangaben.

1. Übersicht Datenbanksysteme 1

1. Übersicht
2. Konzeptueller Datenbankentwurf (ER-Modell)
3. Logischer Datenbankentwurf (relationales Datenmodell)
4. Datenbanksprachen
 - 4.1 SQL DDL und DML
 - 4.2 SQL-Anfragen 1
 - 4.3 SQL-Anfragen 2
 - 4.4 SQL Programmiersprachen-Anbindung

Literaturhinweise zur Vorlesung

- DAS Datenbank-Buch im deutschsprachigen Raum
 - A. Kemper, A. Eickler: Datenbanksysteme – Eine Einführung, De Gruyter Verlag, 10. Auflage, 2015
- Vorlesungsvideos zu Datenbanksysteme
 - <https://db.in.tum.de/teaching/bookDBMSeinf/aufzeichnungen/index.shtml?lang=de>
- Dokumentation Datenbanksystem Oracle
 - <http://oracle19c.in.hwg-konstanz.de>

Einführung

- Informationsexplosion im Internetzeitalter ("Datenzeitalter")
 - Daten als dritter Produktionsfaktor neben Kapital, Arbeit, Rohstoffe (s.a. Gadatsch)
 - Daten als "das neue Öl"
 - Daten als Quelle von Wertschöpfung und Innovation
 - Ziele von Datenbanksystemen
 - Effiziente Verwaltung großer Datenmengen
 - Daten mehreren Benutzern zur Verfügung stellen
- Steigende Bedeutung von Datenbanken



Bedeutung von Datenbanken

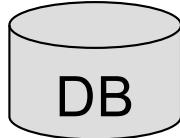
"If any technology was essential to the rebuilding of our daily lives around digital infrastructures, it was the database management system."

Thomas Haigh : How Charles Bachman Invented the DBMS, a Foundation of Our Digital World,
Communication of the ACM, 2016

- Datenbanksysteme als „wichtigste Vorlesung im Studium“
 - Ziel sollte nicht nur das Bestehen der Vorlesung sein

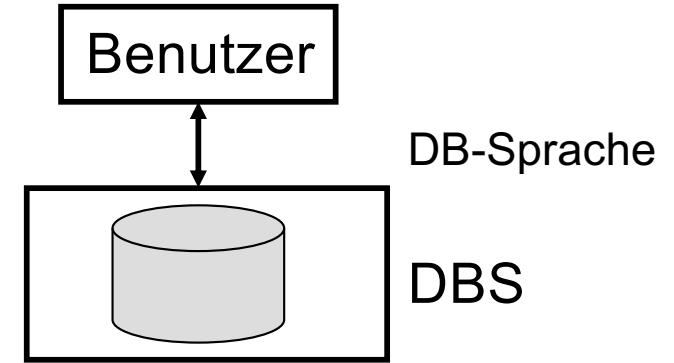
Datenbanken und Datenbanksysteme

Definitionen

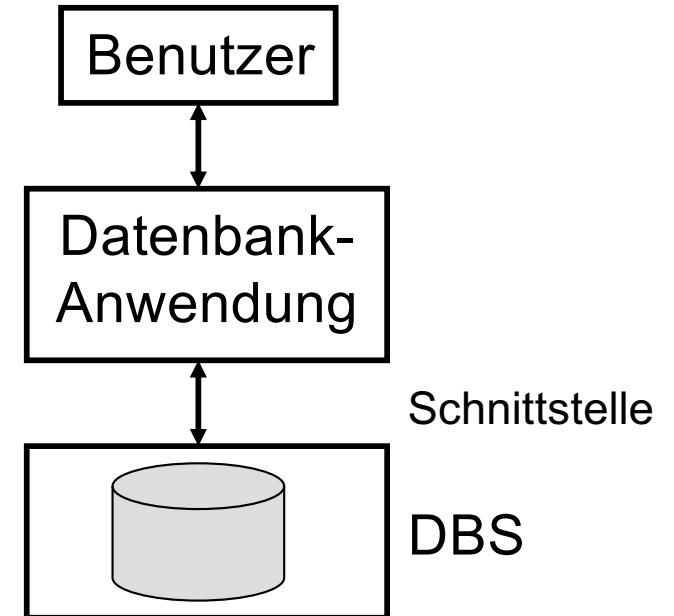
- Sehr allgemeine Definition von Datenbanken
 - Eine Datenbank ist eine selbständige und auf Dauer ausgelegte Datenorganisation, welche einen Datenbestand sicher und flexibel verwalten kann
- Datenbank (DB)
 - Sammlung von Daten von einem Datenbankmanagementsystem
 - Symbol
- Datenbankmanagementsystem (DBMS)
 - Programm zur Kontrolle des Datenbankzugriffs
- Datenbanksystem (DBS)
 - Datenbank + Datenbankmanagementsystem

Nutzung von Datenbanksystemen

- Direkte Nutzung
 - Direkte Nutzung des Endbenutzers
 - Verwendung einer Datenbanksprache, z.B. SQL

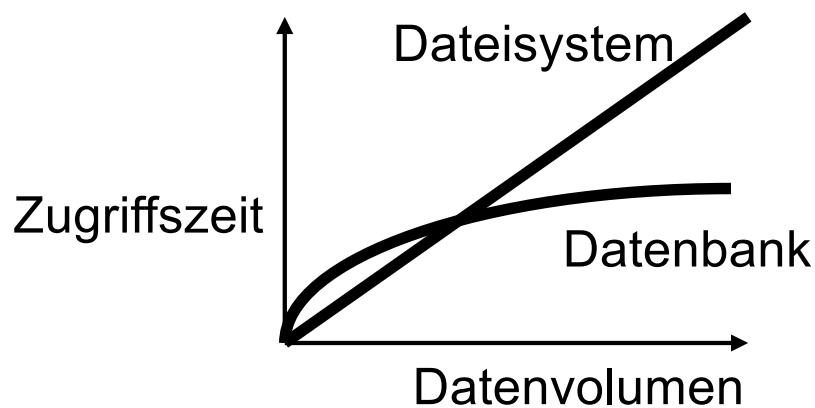


- Programmorientierte Systeme
 - Kommunikation mit Datenbank erfolgt über ein Programm
 - Programmierung des Programms durch Datenbankanweisungen



Eigenschaften von Datenbanksystemen

- Dauerhafte Speicherung und Ausfallsicherheit
 - Daten bleiben nach Programmende erhalten
 - Daten bleiben nach Systemausfall erhalten (Recovery)
- Verwaltung großer Datenmengen
 - Datenmenge im Bereich TB
 - Effiziente Suchmethoden



Eigenschaften von Datenbanksystemen

- Zugriffskontrolle und Sicherheit
 - Einschränkung der für Benutzer zugreifbaren Daten
 - Zugriffsrechte
- Konsistenz- und Integritätsprüfungen
 - Konsistenzbedingungen werden formuliert und überprüft
 - Beispiel: Kontostand von Sparbuch wird niemals negativ
 - "Alles-oder-Nichts"-Eigenschaft
- Redundanzfreiheit
 - Jedes Datum wird genau einmal gespeichert (Ausnahme Speicherungen aus Gründen der Ausfallsicherheit)

Eigenschaften von Datenbanksystemen

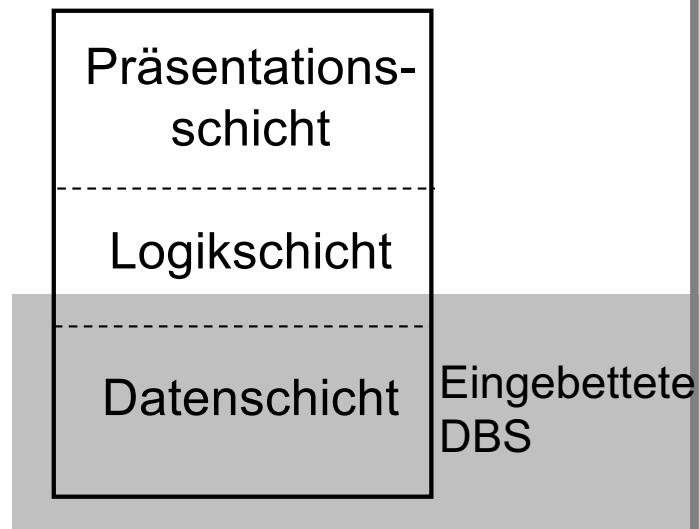
- Mehrbenutzerbetrieb (Concurrency control)
 - Gleichzeitiger Zugriff auf gleiche Datenbestände zugreifen durch mehrere Benutzer
 - Vermeidung von Anomalien durch Synchronisationsverfahren
 - Verwendung von Sperrprotokollen
- Hohe Verfügbarkeit und Fehlertoleranz
 - Keine separaten Zeiträume für Reorganisation der Daten oder Erstellen von Sicherheitkopien

Eigenschaften von Datenbanksystemen

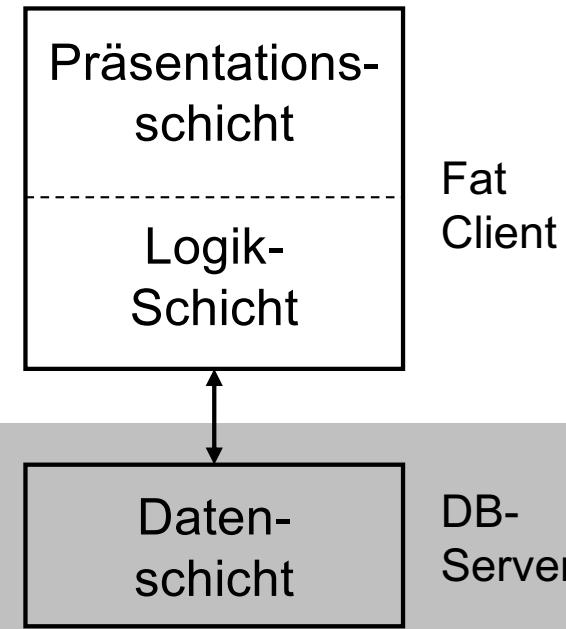
- Anfragesprache
 - Inhaltsbezogener Zugriff durch deklarative Sprachen
 - Suchmechanismen
 - Beispiel SQL
- Kopplung mit Anwendungssystemen
 - Integration mit Programmiersprachen
 - Verkürzung der Entwicklungszeit von Anwendungen, da auf eine Schnittstelle zur Datenverwaltung aufgesetzt werden kann
- Skalierbarkeit
 - Anpassbarkeit an Größe und Leistungsanforderungen
 - Lineare Leistungssteigerung durch
 - Vergrößerung von Haupt- oder Externspeicher
 - mehr oder leistungsfähigere Prozessoren

Architekturen von Datenbankapplikationen

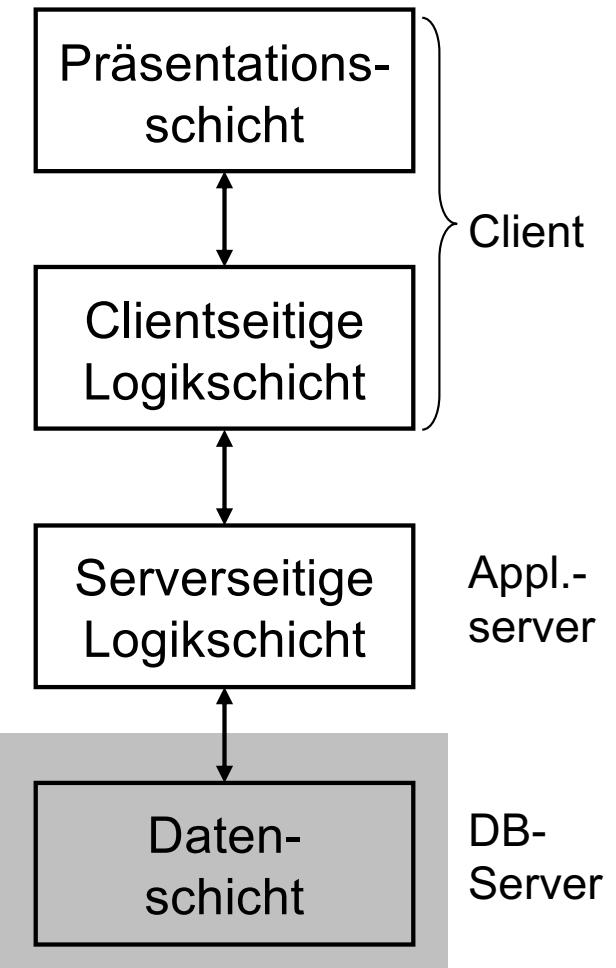
1-tier Architektur



2-tier Architektur



n-tier Architektur

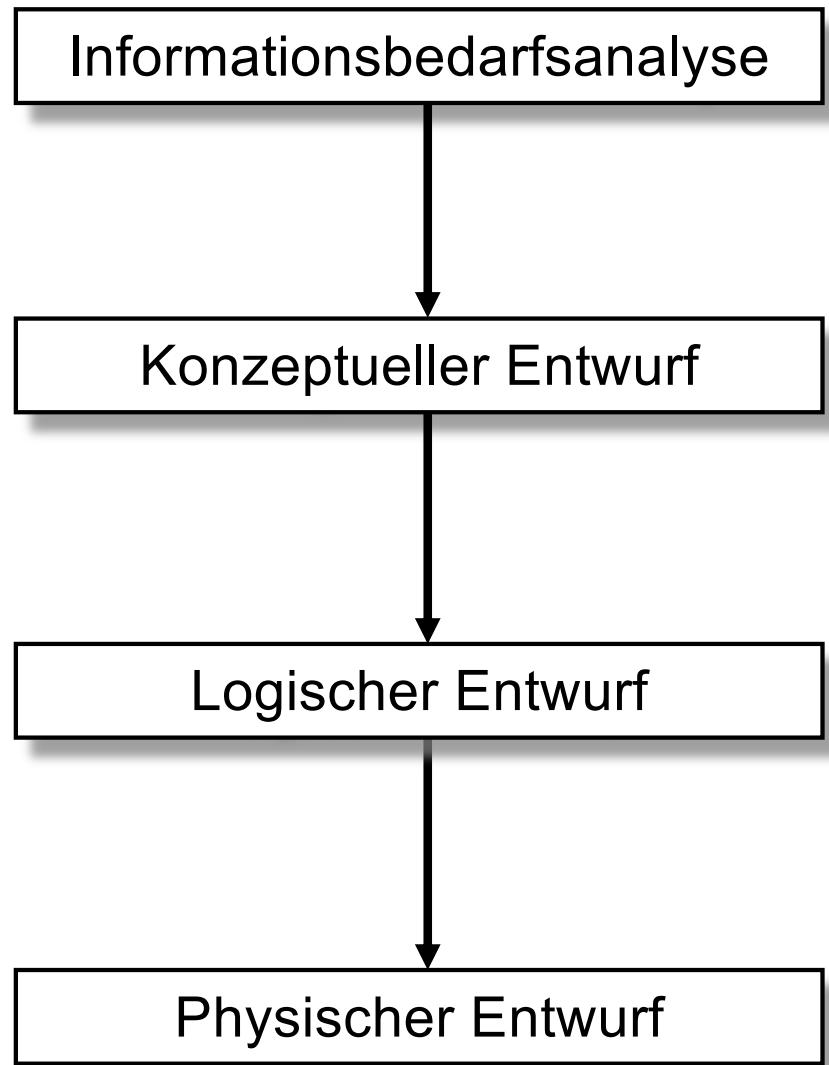


Zeittafel der Datenbanksystem-Generationen

- 1. Generation
 - Filesysteme auf Band
- 2. Generation
 - Filesysteme auf Platte
- 3. Generation
 - Prärelationale Systeme (Netzwerk-, hierarchische Systeme)
- 4. Generation
 - Relationale Systeme
- 5. Generation
 - Postrelationale Systeme (u.a. objektbasierte Systeme, NoSQL)

Datenbankentwurf

Entwurfsschritte



Sammlung aller für eine Miniwelt bedeutsamen Gegenstände, Eigenschaften, Beziehungen und Operationen

Präzise Beschreibung einer Miniwelt durch relationale oder objektorientierte Modelle

Abbildung auf ein rechnergestützt interpretierbares Schema, z.B. relationales Schema

Abbildung des logischen Datenbankschemas in eine effiziente physische Datenbasisstruktur

Datenmodelle

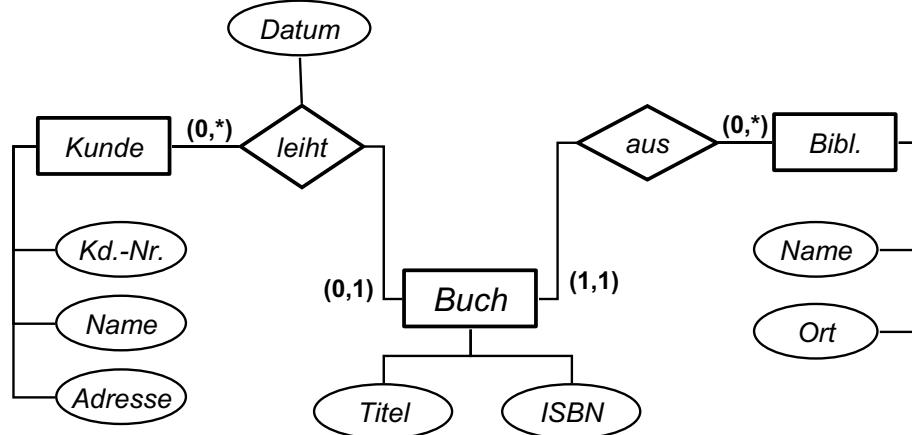
- Relationales Datenmodell
 - Basiert auf Relationenkalkül der Algebra
 - Mächtige Datenzugriffssprache SQL
 - Datenunabhängigkeit
 - Vertreter DB2 von IBM, Oracle, Informix

Pers							
pnr	name	jahrg	eindat	gehalt	beruf	anr	vnr
406	Coy	1972	01.03.06	100.000	Kaufmann	K55	123
123	Mueller	1980	01.09.00	88.000	Programmierer	K51	
829	Schmidt	1982	01.06.10	94.000	Kaufmann	K53	123
874	Abel		01.05.14	82.000	Softw.Entwickler	K55	829
503	Junghans	1997		80.000	Programmierer	K51	123
...

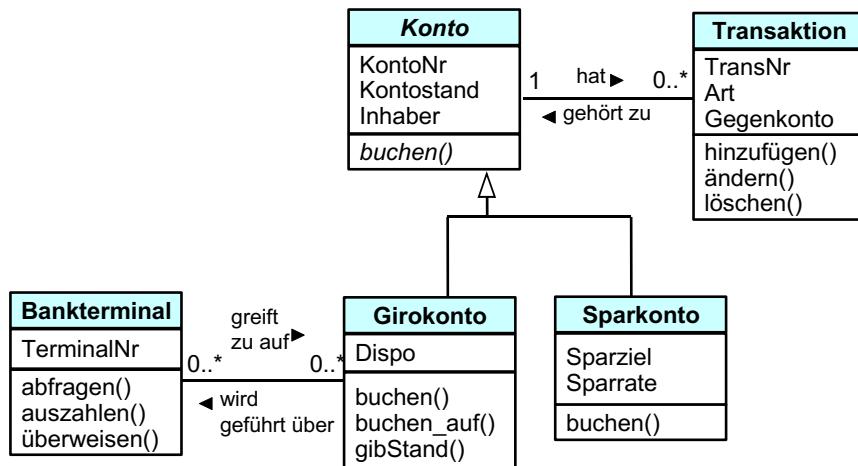
Konzeptueller Entwurf

Relationale und objektorientierte Modellierung

- Entity-Relationship-Modell

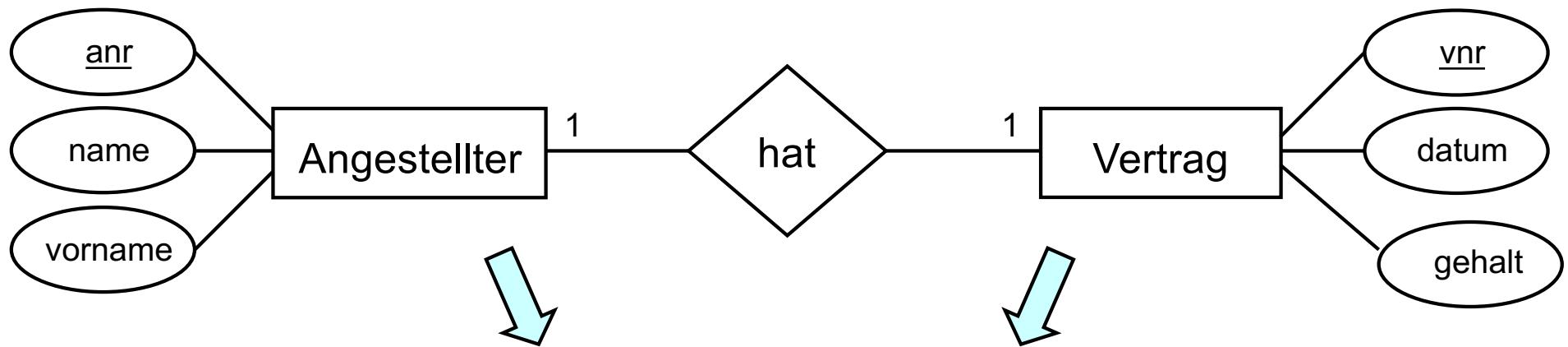


- Objektorientiertes Modell



Logischer Entwurf

- Abbildung auf relationales Schema, Normalisierung



Angestellter = (anr, name, vorname, vnr)

Vertrag = (vnr, datum, gehalt)

Beispiel:

Angestellter			
<u>anr</u>	name	vorname	<u>vnr</u>
3004	Müller	Hans	v206
1208	Zimmer	Jochen	v128
1001	Abel	Kai	v771

Vertrag		
<u>vnr</u>	datum	gehalt
v206	01.10.2014	6000
v128	17.09.2020	7500
v771	22.03.2024	5900

Physischer Entwurf

- Verwendung der Datenbanksprache SQL

```
CREATE TABLE Angestellter
( anr          integer,
  name         varchar(20),
  vorname      varchar(20),
  vnr          integer,
  CONSTRAINT pk_Angestellter PRIMARY KEY(anr),
  CONSTRAINT fk_Angestellter FOREIGN KEY(vnr)
    REFERENCES Vertrag(vnr)
    ON DELETE SET NULL );

CREATE TABLE Vertrag
( vnr          integer,
  datum        date,
  gehalt       integer,
  CONSTRAINT pk_Vertrag PRIMARY KEY(vnr) );
```