

Data Warehousing und Data Mining

Eine sehr kurze Übersicht über
relationale Datenbanken

Ulf Leser

Wissensmanagement in der
Bioinformatik



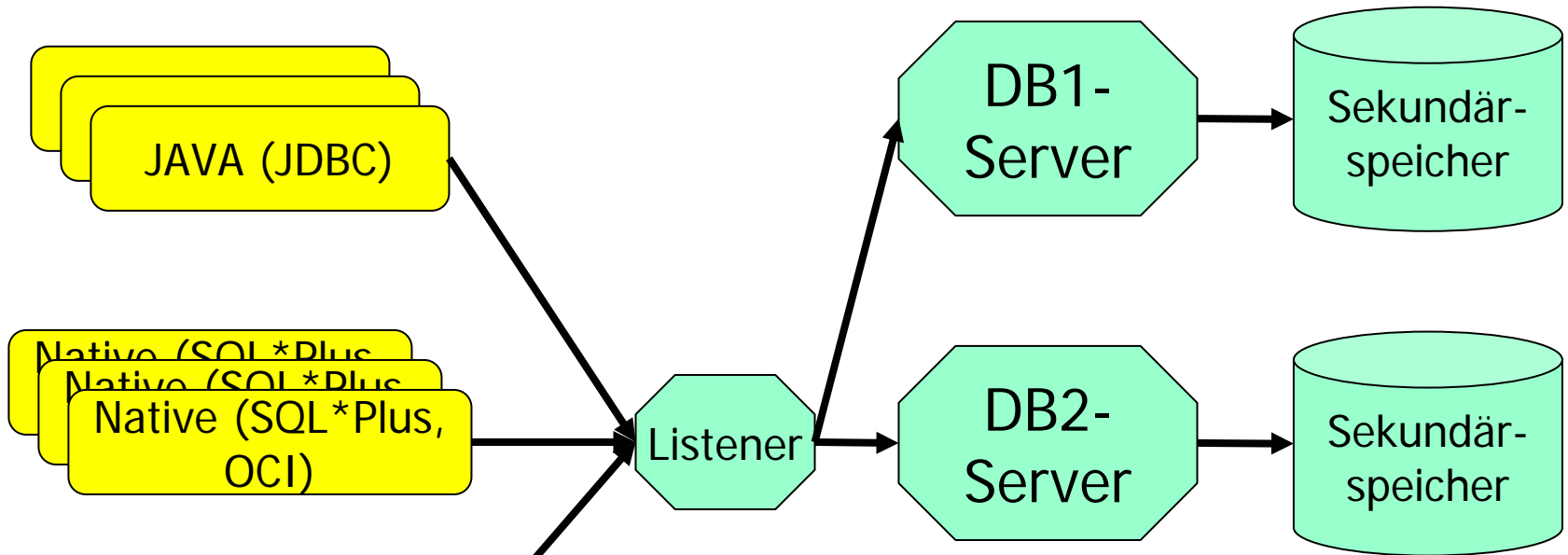
Inhalt dieser Vorlesung

- Relationale Datenbanken
- Relationales Modell und Datenbankentwurf
- Relationale Operatoren & Algebra
- SQL
- Anfragebearbeitung
- Indexstrukturen
- Normalformen

Relationale Datenbank

- **Relationale Datenbank Management Systeme (RDBMS)**
 - Serverbasierte Software
 - Ein RDBMS – viele RDB
 - Aufgaben des Servers
 - Hauptspeicher- und Sekundärspeicherverwaltung
 - Transaktionsmanagement
 - Anfragebearbeitung und –optimierung
 - Backup und Recovery
 - Datenkonsistenz
 - Userverwaltung
 - Systeme
 - Oracle, DB2 (Informix), Sybase, NCR Teradata, SQL-Server
 - PostgreSQL, Interbase, Berkeley-DB, DB4Objects, MySQL, Ingres, SAP-DB, Monet-DB, ...

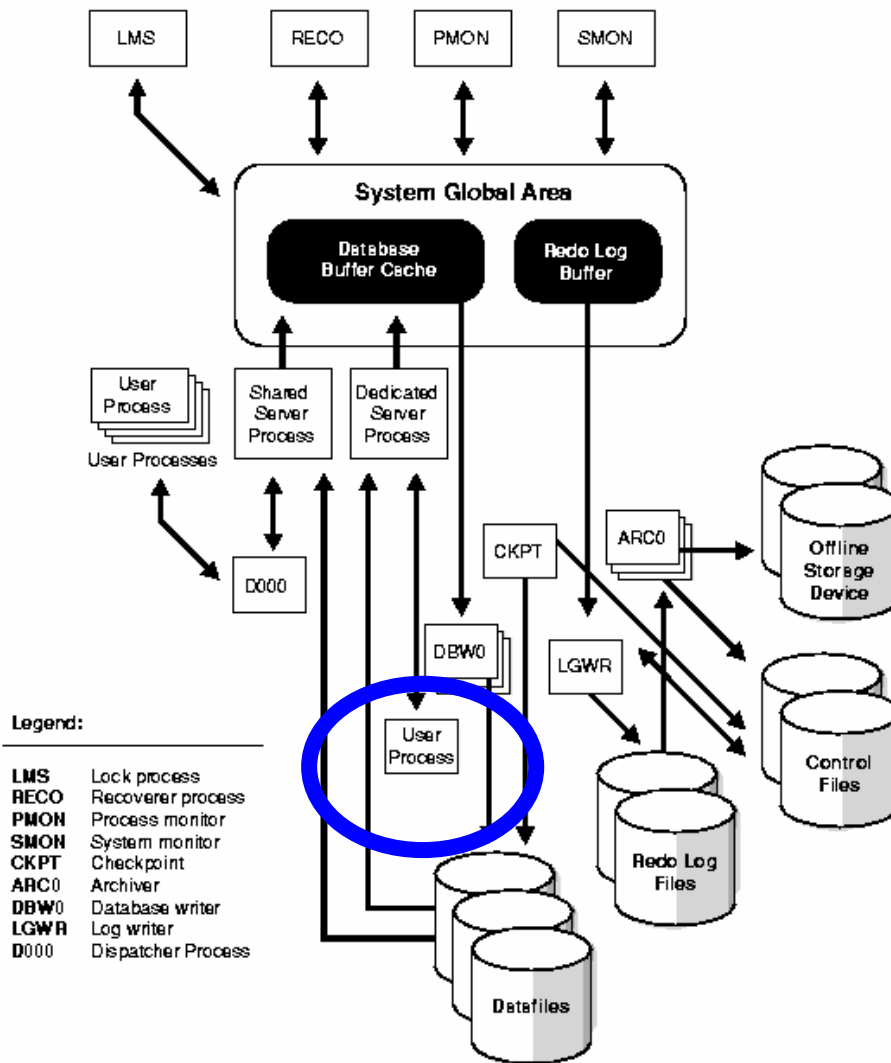
Client-Server



Konsistenz
Parallelisierung
Recovery
Lastverteilung
Authentifizierung
Autorisierung

....

Ein Oracle-Server



Relationales Datenmodell

- Repräsentation aller Daten in Tabellen
 - Tabellenname
 - Attribute
 - Datentypen
- Vergleich zu UML/objektorientierten Datenmodellen?

Mitarbeiter

Spalten/Attribute

Zeilen/
Rows/
Tupel

id	vorname	nachname	alter	adresse
1	Peter	Müller	32	10101 Berlin
2	Stefanie	Meier	34	11202 Berlin
5	Petra	Weger	28	80223 München
7	Andreas	Zwickel	44	80443 München
...

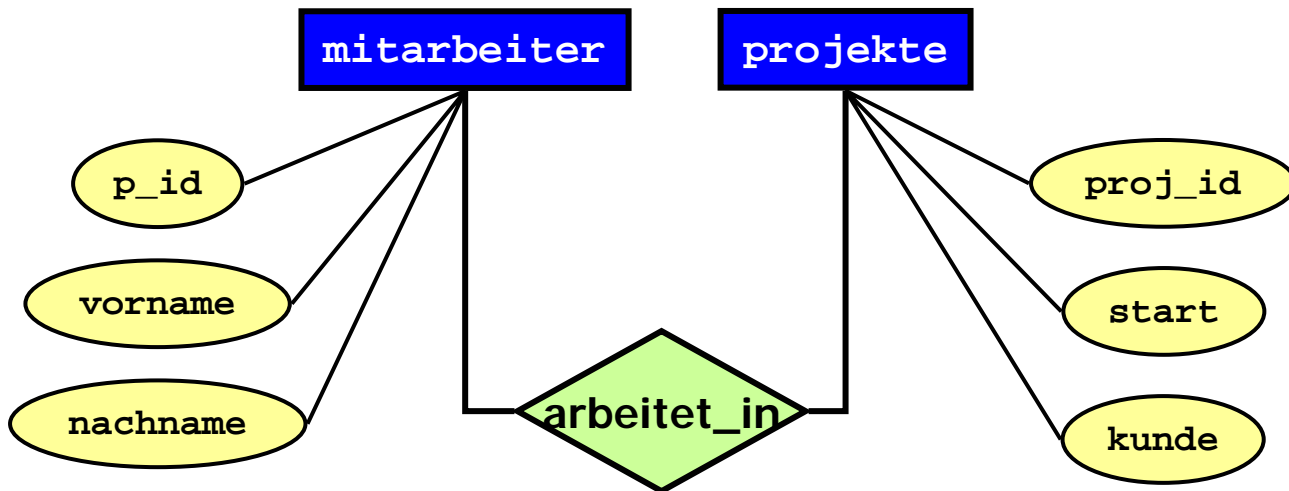
ER-Modellierung

- Relationales Modell „semantikarm“
- Modellieren in Tabellen wenig intuitiv
- Modellierungssprachen: ER, EER, UML, ...
- Entity-Relationship Modell

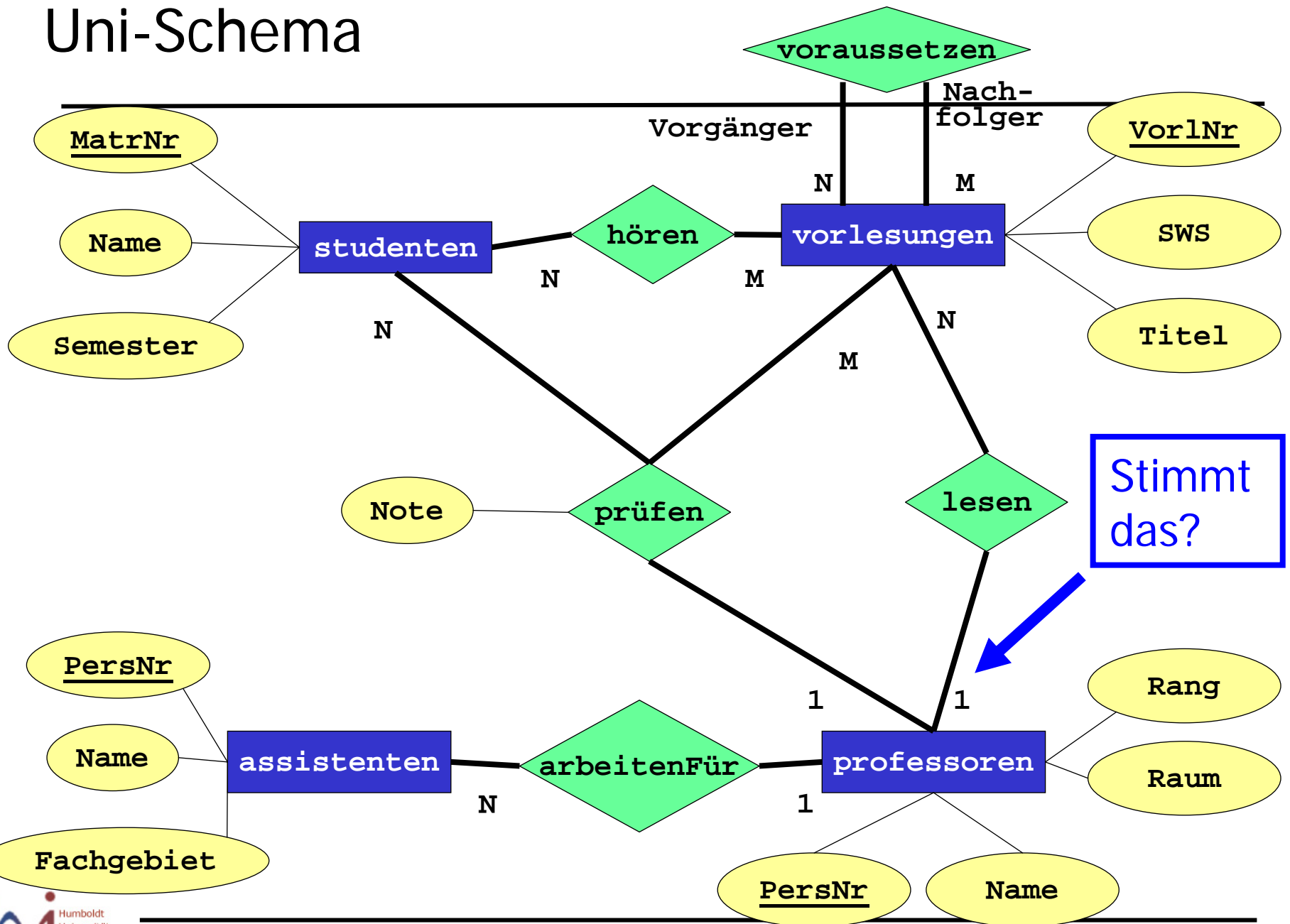
– Entitytypen

– Attribute

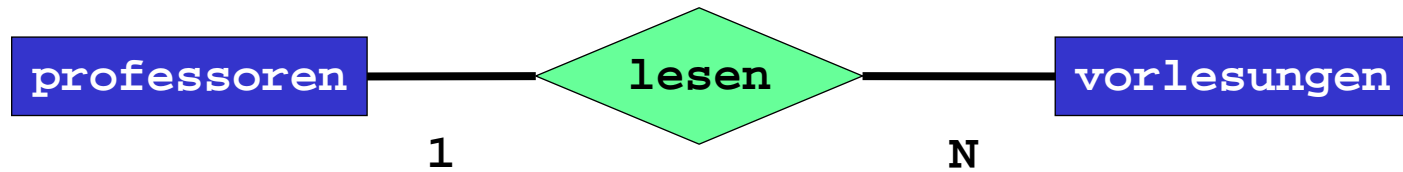
– Beziehungen



Uni-Schema



Ableitung eines relationalen Schemas



1:N-Beziehung

- Initial-Entwurf

vorlesungen : {VorlNr, Titel, SWS}
professoren : {PersNr, Name, Rang, Raum}
lesen: {VorlNr, PersNr}

Verfeinerung des relationalen Schemas

1:N-Beziehung

- Initial-Entwurf

`vorlesungen : {VorlNr, Titel, SWS}`

`professoren : {PersNr, Name, Rang, Raum}`

`lesen: {VorlNr, PersNr}`

- Verfeinerung durch Zusammenfassung

`vorlesungen : {VorlNr, Titel, SWS, gelesenVon}`

`professoren : {PersNr, Name, Rang, Raum}`

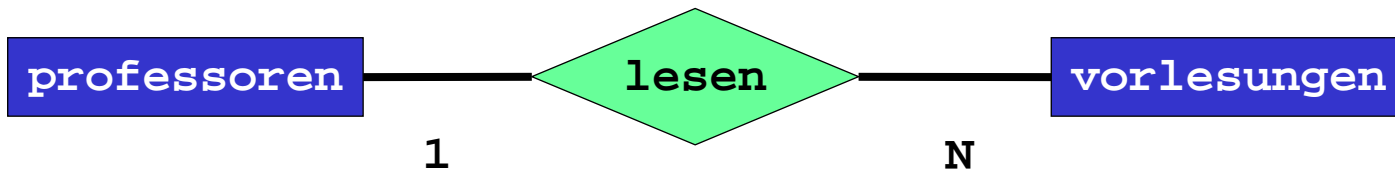
Regel

Relationen mit gleichem Schlüssel kann man zusammenfassen
aber nur diese und keine anderen!

Ausprägungen

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

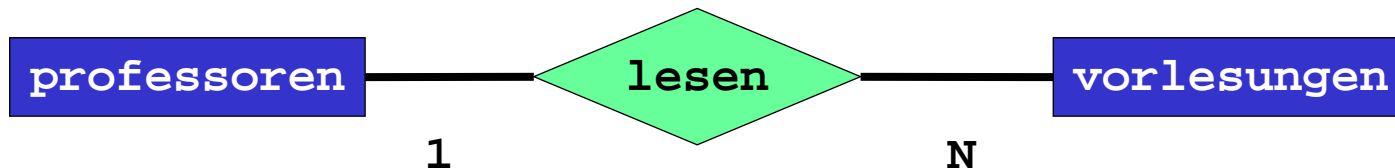
Vorlesungen			
VorlNr	Titel	SWS	Gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137



So geht es NICHT

Professoren				
PersNr	Name	Rang	Raum	liest
2125	Sokrates	C4	226	5041
2125	Sokrates	C4	226	5049
2125	Sokrates	C4	226	4052
...
2134	Augustinus	C3	309	5022
2136	Curie	C4	36	??

Vorlesungen		
VorlNr	Titel	SWS
5001	Grundzüge	4
5041	Ethik	4
5043	Erkenntnistheorie	3
5049	Mäeutik	2
4052	Logik	4
5052	Wissenschaftstheorie	3
5216	Bioethik	2
5259	Der Wiener Kreis	2
5022	Glaube und Wissen	2
4630	Die 3 Kritiken	4



Folgen: Anomalien

Professoren				
PersNr	Name	Rang	Raum	liest
2125	Sokrates	C4	226	5041
2125	Sokrates	C4	226	5049
2125	Sokrates	C4	226	4052
...
2134	Augustinus	C3	309	5022
2136	Curie	C4	36	??

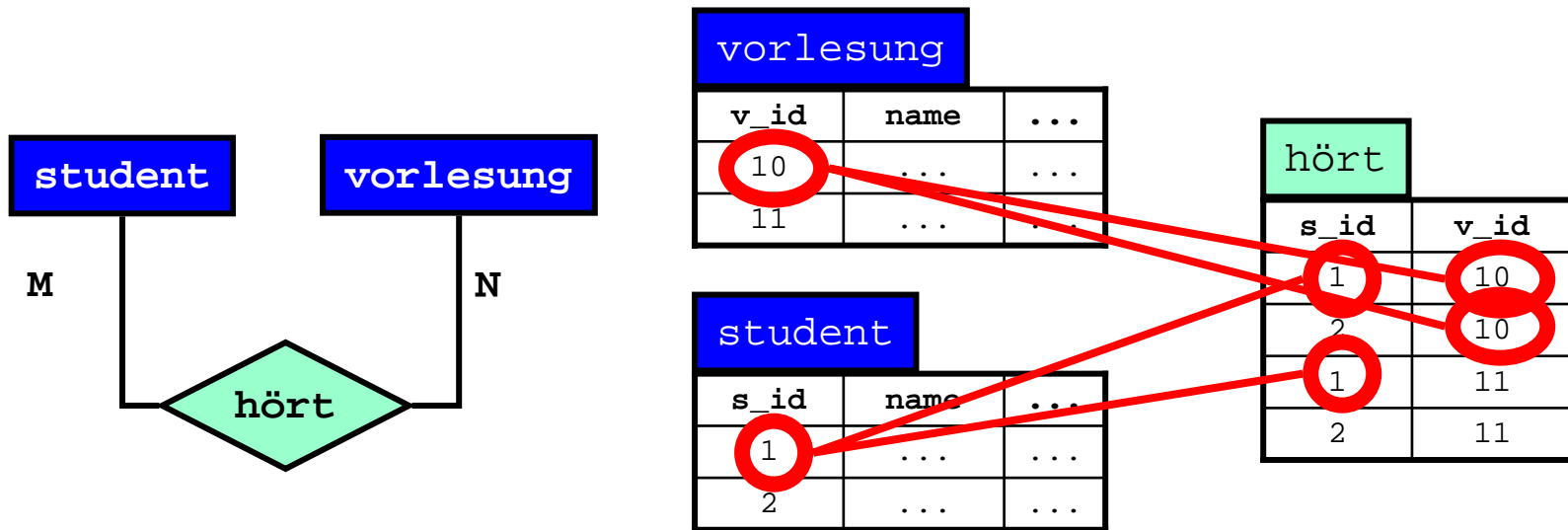
Vorlesungen		
VorINr	Titel	SWS
5001	Grundzüge	4
5041	Ethik	4
5043	Erkenntnistheorie	3
5049	Mäeutik	2
4052	Logik	4
5052	Wissenschaftstheorie	3
5216	Bioethik	2
5259	Der Wiener Kreis	2
5022	Glaube und Wissen	2
4630	Die 3 Kritiken	4

- Update-Anomalie
 - Was passiert wenn „Sokrates“ umzieht?
- Lösch-Anomalie
 - Was passiert wenn „Glaube und Wissen“ wegfällt
- Einfügeanomalie
 - „Curie“ ist neu und liest noch keine Vorlesungen

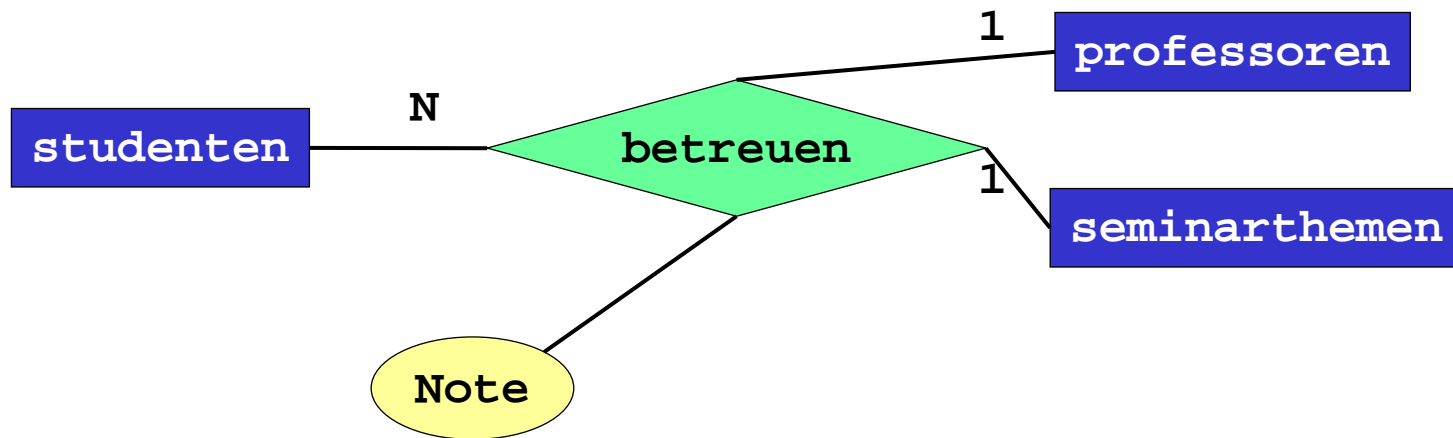


Beziehung „hören“

- m:n Beziehung
- „Brückentabelle“ mit zwei Fremdschlüsseln



Beziehung „betreuen“



betreuen : professoren x studenten → seminarthemen

betreuen : seminarthemen x studenten → professoren

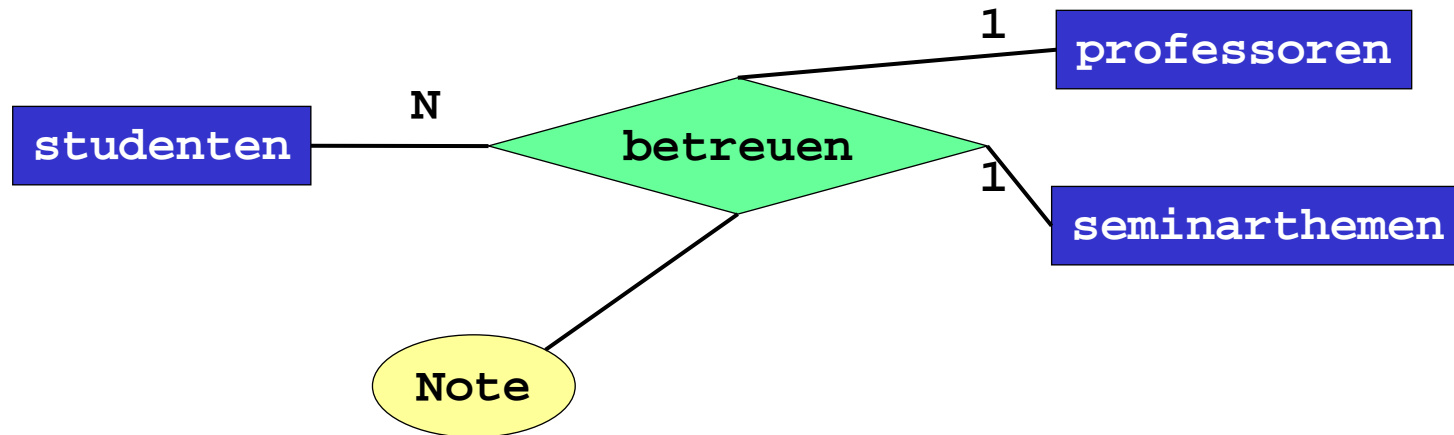
Erzwungene Konsistenzbedingungen

1. Studenten dürfen bei demselben Professor bzw. derselben Professorin nur ein Seminarthema „ableisten“
2. Studenten dürfen dasselbe Seminarthema nur einmal bearbeiten – sie dürfen also nicht bei anderen Professoren ein schon einmal erteiltes Seminarthema nochmals bearbeiten.

Es sind aber folgende Datenbankzustände möglich:

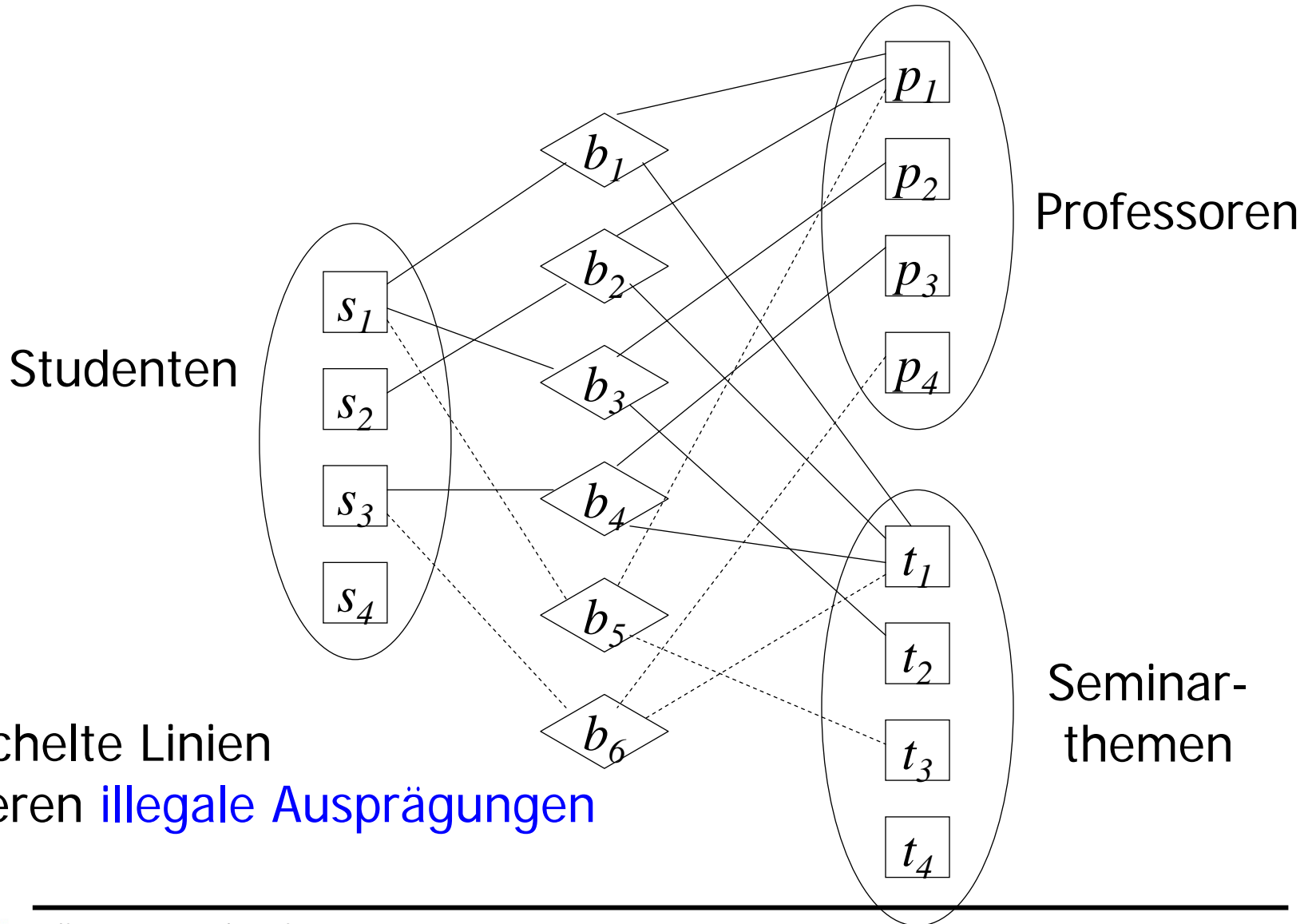
- Professoren können dasselbe Seminarthema „wiederverwenden“ – also dasselbe Thema auch mehreren Studenten erteilen
- Ein Thema kann von mehreren Professoren vergeben werden – aber an unterschiedliche Studenten

Wie sieht das relationale Schema aus?



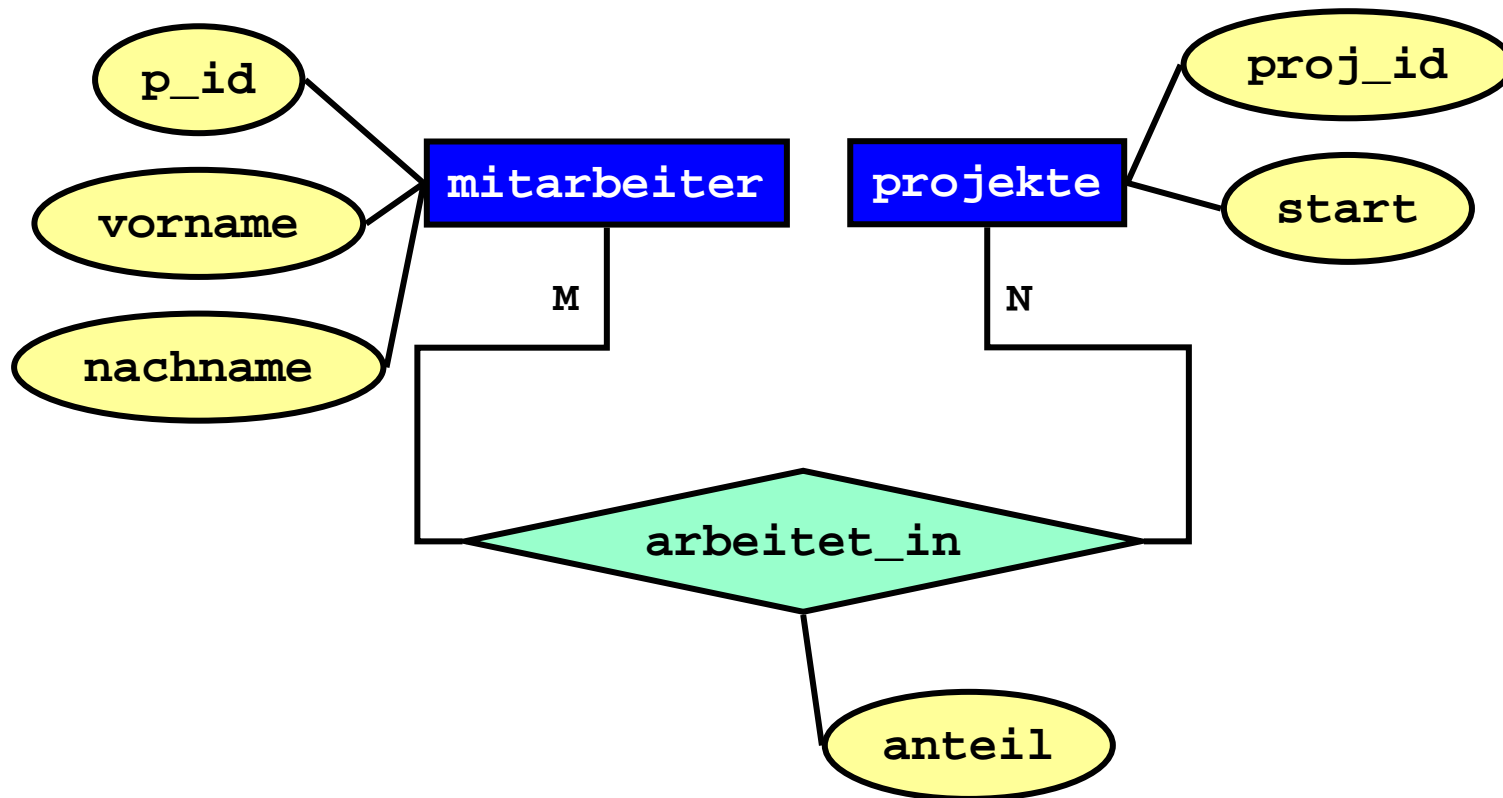
- `betreuen{ student, professor, thema, note }`
- `unique(student, professor)`
 - Jeder Student nur ein Thema pro Professor
- `unique(student, thema)`
 - Jedes Thema nur einmal pro Student

Ausprägung der Beziehung *betreuen*



Gestrichelte Linien
markieren illegale Ausprägungen

Relationale Operationen



Operationen auf Relationen/Tabellen

- Selektion
 - Alle Zeilen von `mitarbeiter` mit `alter>40` und `name=„Müller“`
- Projektion
 - Nur die `mitarbeiter`-Spalten `vorname`, `nachname`
- Kartesisches Produkt
 - Alle Zeilen von `mitarbeiter` verknüpft mit jeweils allen Zeilen von `arbeitet_in`
- Komposition von Operationen
 - Die Spalten `nachname` und `proj_id` aller Zeilen des kartesischen Produkts von `mitarbeiter` und `arbeitet_in`, bei denen `mitarbeiter.p_id=arbeitet_in.p_id` mit `anteil` größer als 10%

Die relationale Algebra

- σ Selektion
- π Projektion
- \times Kreuzprodukt
- \bowtie Join (Verbund)
- ρ Umbenennung
- $-$ Mengendifferenz
- \div Division
- \cup Vereinigung
- \cap Mengendurchschnitt
- \ltimes linker Semijoin
- \rtimes rechter Semijoin
- $\ltimes\bowtie$ Left outer join
- $\bowtie\rtimes$ Right outer join
- $\bowtie\bowtie$ Full outer join

Der natürliche Verbund (Join)

Gegeben seien:

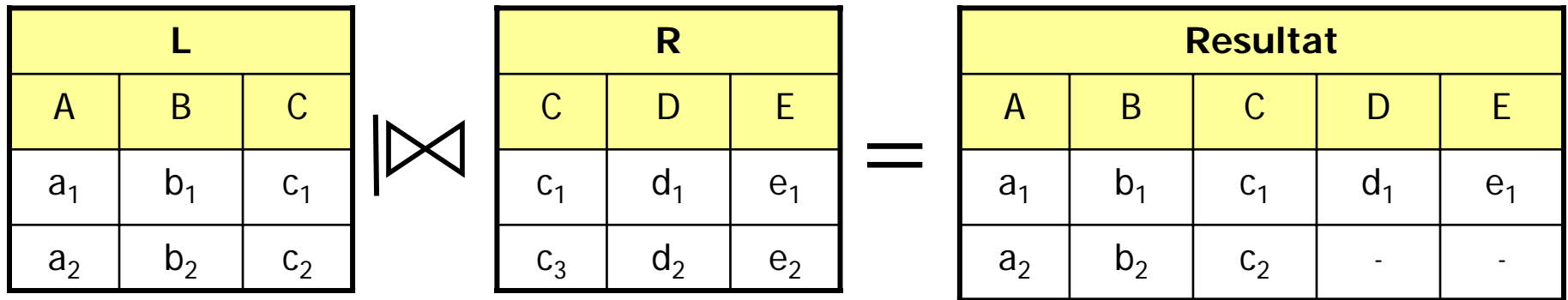
- $R(A_1, \dots, A_m, B_1, \dots, B_k)$
- $S(B_1, \dots, B_k, C_1, \dots, C_n)$

$$R \bowtie S = \Pi_{A_1, \dots, A_m, R.B_1, \dots, R.B_k, C_1, \dots, C_n} (\sigma_{R.B_1=S.B_1 \wedge \dots \wedge R.B_k=S.B_k} (R \times S))$$

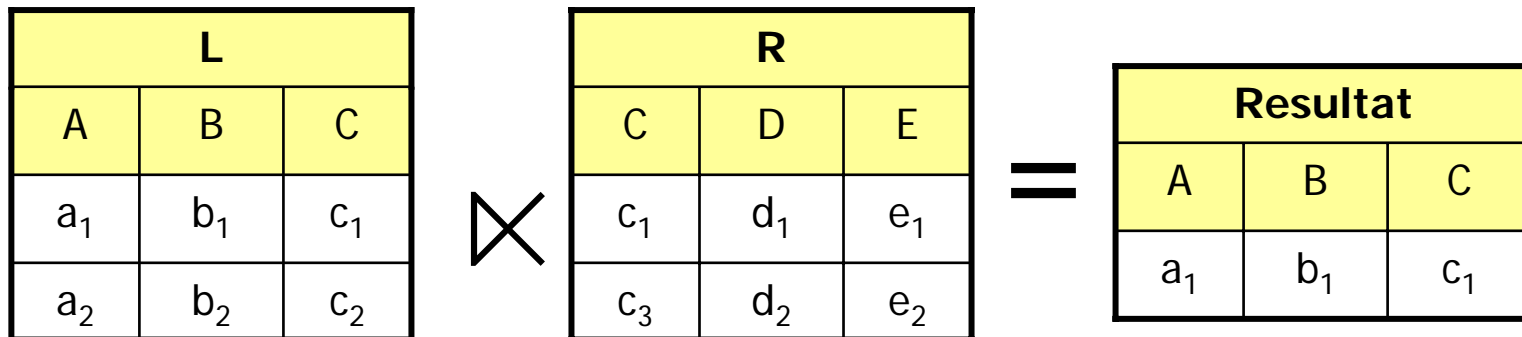
R \bowtie S											
R – S				R \cap S				S – R			
A ₁	A ₂	...	A _m	B ₁	B ₂	...	B _k	C ₁	C ₂	...	C _n
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Andere Join-Arten

- Linker äußerer Join



- Linker Semi-Join



SQL – Grundkonzepte

```
SELECT    M.nachname, A.arbeitet_in
FROM      mitarbeiter M, arbeitet_in A
WHERE     M.p_id = A.p_id AND
          A.anteil > 0.1
```

- SQL: Structured Query Language
- ANSI-SQL, SQL-92, SQL-99, SQL-3
- **Deklarativer Charakter**: Was, nicht wie
- Vier Grundbefehle: `insert`, `update`, `delete`, `select`
- DDL versus DML
- Andere Sprachen
 - Tupel/Domänenkalkül, relationale Algebra, QBE, **Datalog**

Insert

- Einfügen von Werten in Tabelle
- Prinzipiell ein Tupel pro Insert

```
INSERT INTO mitarbeiter VALUES  
    (1, „Peter“, „Müller“, 38, „10101 Berlin“);
```

```
INSERT INTO projekte (proj_id, name, kunde) VALUES  
    (1, „Stammhaus-BMW“, „BMW“);
```

- Erweiterungen
 - Bulk-Insert
`INSERT INTO ... SELECT ...`
 - Konditionales Insert in mehrere Tabellen
`INSERT INTO ... INTO ... INTO ...`

Update

- **Ändern von Werten** in Tabellen
- Mengensemantik: Ändert mehrere Werte

```
UPDATE projekte
SET   status = „abgebrochen“,
      kunde = kunde || „-insolvent“
WHERE kunde=„Grundig“
```

- Typisches Muster

```
UPDATE tabelle
SET ... = (SELECT ... FROM ... WHERE)
WHERE id in (SELECT ... FROM ... WHERE)
```

- Erweiterungen
 - **UPSERT / MERGE**

Delete

- Löschen von Tupeln in einer Tabelle

```
DELETE FROM projekte  
WHERE status=„abgeschlossen“
```

- Typisches Muster

```
DELETE FROM projekte  
WHERE id in (SELECT ... FROM ... WHERE)
```

- Löschen ist eine **performancekritische Operation**
 - **DELETE, DROP TABLE, TRUNCATE**, Partitionen, ...

Select

- **Selektieren von Werten** aus mehreren Tabellen

```
SELECT      M.nachname, A.anteil
FROM        mitarbeiter M, arbeitet_in A
WHERE       M.p_id = A.p_id AND
           a.anteil > 0.1
```

```
SELECT      M.nachname, P.name, A.anteil
FROM        mitarbeiter M, projekte P, arbeitet_in A
WHERE       M.p_id = A.p_id AND
           A.proj_id = P.proj_id
```

```
SELECT      M.nachname, A.anteil
FROM        mitarbeiter M, arbeitet_in A
WHERE       M.p_id = A.proj_id
```



- Ergebnis ist immer eine Tabelle
- Ausführung ist Sache des RDBMS - Optimierung

Varianten

- Subqueries

- Korreliert oder nicht
- Unkorrelierte Form?

```
SELECT  vorname, nachname
FROM    mitarbeiter M
WHERE   EXISTS (
        SELECT A.p_id
        FROM  arbeitet_in A
        WHERE A.p_id = M.p_id
```

- Self-Join

- „Begrenzte Rekursion“

```
SELECT P1.name, P2.name
FROM  projekte P1, projekte P2
WHERE P1.vorgaenger=p2.proj_id AND
      P2.status=„abgeschlossen“
```

- SQL in FROM Klausel

- „In-Line Views“
- Nützlich bei Top-Ten / Sortieranfragen
- Top Ten Anfragen?

```
SELECT X.nachname, X.status
FROM  (
    SELECT M.nachname, p.status
    FROM  mitarbeiter, projekte, arbeitet_in
    WHERE M.p_id=A.p_id AND
          A.proj_id=P.proj_id
    ) X
WHERE  X.status=„Akquisition“
```

Unkorrelierte versus korrelierte Unteranfragen

Korrelierte Formulierung – Umformung ?

```
SELECT s.*
FROM studenten s
WHERE EXISTS
  (SELECT p.*
   FROM professoren
   WHERE p.geb_datum > s.geb_datum);
```

Besser: Äquivalente unkorrelierte Formulierung

```
SELECT s.*
FROM studenten s
WHERE s.geb_datum <
  (SELECT max (p.geb_datum)
   FROM professoren p);
```

- Unteranfrageergebnis kann materialisiert werden
- Unteranfrage braucht nur einmal ausgewertet werden

Weitere Operationen

- Aggregation und GROUP BY

```
SELECT      p.proj_id, COUNT(*), SUM(alter)/COUNT(*)
FROM        mitarbeiter M, arbeitet_in A, projekte P
WHERE       M.p_id=A.p_id AND A.proj_id=P.proj_id
GROUP BY    P.proj_id
```



```
SELECT      p.proj_id, P.name, COUNT(*)
FROM        mitarbeiter M, arbeitet_in A, projekte P
WHERE       M.p_id=A.p_id AND A.proj_id=P.proj_id
GROUP BY    P.proj_id
```

- ORDER BY

```
SELECT      P.name, M.nachname
FROM        mitarbeiter M, arbeitet_in A, projekte P
WHERE       M.p_id=A.p_id AND A.proj_id=P.proj_id
ORDER BY    P.name, M.nachname
```

Views

- Definition von „benannten“ Queries

```
CREATE VIEW proj_pers AS
  SELECT      P.proj_id, P.name,
             M.P_id, M.name, M.alter,
  FROM        mitarbeiter M, arbeitet_in A, projekte P
  WHERE       M.p_id=A.p_id AND A.proj_id=P.proj_id
```

- Können viel Schreibarbeit sparen

```
SELECT      proj_id, COUNT(*), SUM(alter)/COUNT(*)
FROM        proj_pers
GROUP BY    proj_id
```

- Verwendung für tupelgenauen Zugriffsschutz
- Views werden i.d.R. syntaktisch expandiert
- Erweiterungen
 - Materialisierte Sichten
 - Rekursive Views

DDL

- DML: Data Manipulation Language
- DDL: [Data Definition Language](#)
- Definition von
 - Tabellen, Indexen, Views, ...
 - Administration: Tablespaces, Segmente, Rollen
 - Benutzerverwaltung: User, Gruppen, Rechte, ...

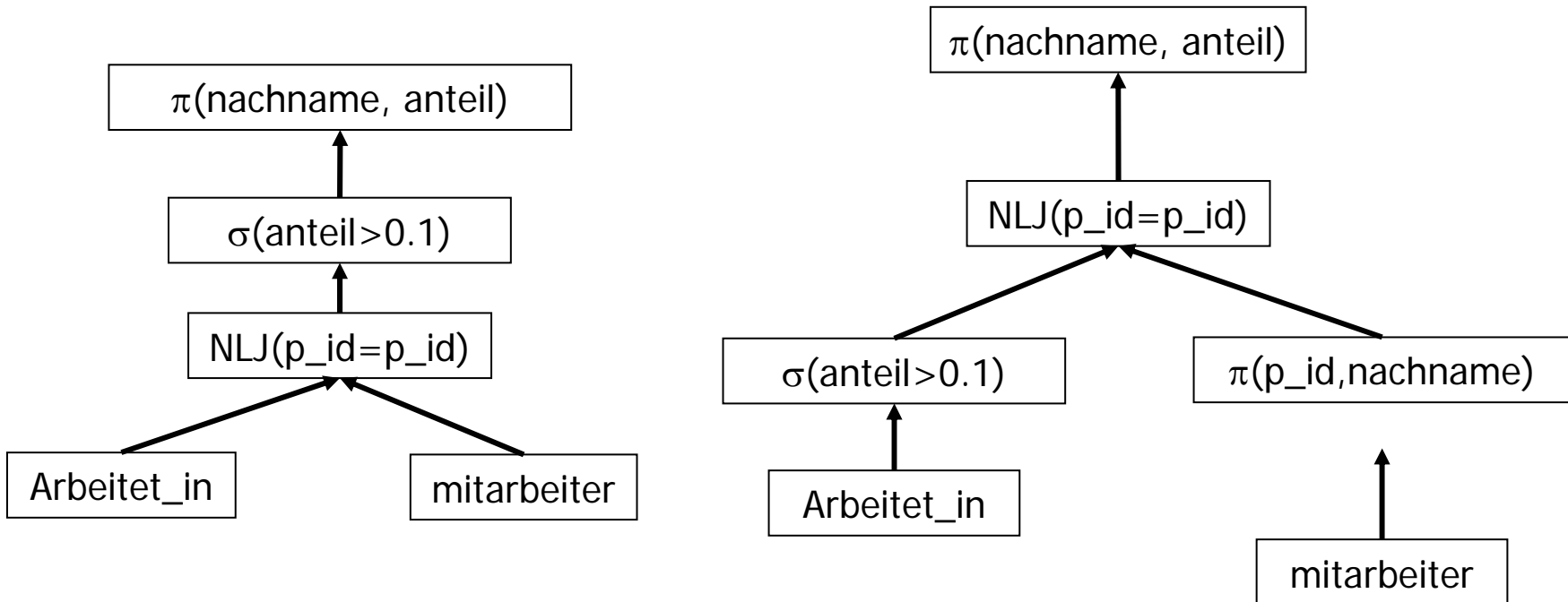
```
CREATE TABLE mitarbeiter (  
    p_id          NUMBER,  
    vorname      VARCHAR2(100),  
    nachname     VARCHAR2(100),  
    alter        NUMBER(2) CHECK (alter>0 AND alter<150),  
    adresse      VARCHAR2(1000)  
);
```

Anfrageübersetzung und -optimierung

- Prinzipieller Ablauf
 - Parsen der Anfrage (Syntax)
 - Überprüfen der Schemaelemente („Semantik“)
 - Expandieren von Views
 - Berechnung von Ausführungsplänen
 - Exponentiell viele
 - Wahl des optimalen Ausführungsplans
 - Regelbasierter Optimierer
 - Kostenbasierter Optimierer - Statistik
 - Ausführung
 - Dynamisches Re-Planen?

Ausführungspläne

```
SELECT      M.nachname, A.anteil
FROM        mitarbeiter M, arbeitet_in A
WHERE       M.p_id = A.p_id AND
           a.anteil > 0.1
```

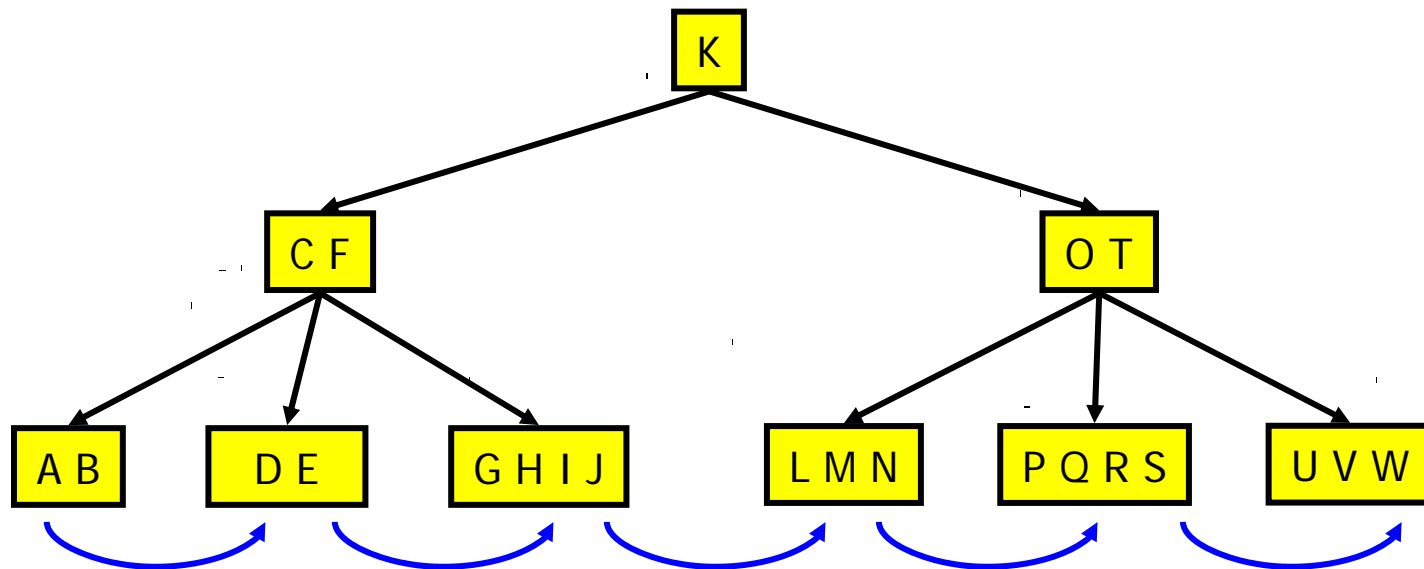


Ausführungspläne

- Freiheitsgrade
 - Algebraische Anfrageumformung
 - Joinreihenfolge
 - Joinmethode (Nested Loop, Sort-Merge, Hash ...)
 - Access path: Indexzugriff (welcher?), Full-Table-Scan
 - 3,5,7% Regel (?)
 - Operatorreihenfolge
- **Kostenbasierter Optimierung**
 - Einbeziehung von Werteverteilungen, Tabellengrößen, Anzahl NULL-Werten, Histogrammen, Selektivität, ...
- Heuristische Ziele
 - Minimierung von Zwischenergebnissen
 - Minimierung von Sekundärspeicherzugriff

Balancierter B und B* Baum

- Knoten enthalten höchstens $2m$ Schlüssel
- Knoten enthalten mindestens m Schlüssel, die Wurzel mindestens 1 Schlüssel
- Knoten mit x Schlüsseln hat $x+1$ Kinder
- **Balancierter Baum**: Alle Blätter haben gleiche Tiefe
- B*: Daten nur in Blättern



B und B* Bäume

- + Sehr schneller Zugriff auf einzelne Elemente ($O(\log(n))$)
- + Schnelle Bereichsanfragen im B* Baum
- + Indexierung von Attributkombinationen

- Elemente müssen geordnet werden können
- Degenerieren bei vielen Duplikaten
- Sequentieller Blockzugriff eventuell schneller (<5%)
- Zusammengesetzte Indexe duplizieren Tabellen
- Indexaktualisierung kostet Zeit
- Auch Indexe müssen gesperrt werden

Integrität

- **Semantisch konsistenter** Zustand der Daten
 - Kann nur im Anwendungskontext definiert werden
- Überwachung nach Möglichkeit durch RDBMS
 - Datenmodell selber (Kardinalitäten)
 - Referentielle Integrität (**Fremdschlüssel**)
 - CHECK Constraints in DDL
 - Trigger
- Zeitpunkt der Überwachung?
 - Nach jeder Operation
 - Nach jeder Transaktion

Normalformen

- Attribute hängen funktional voneinander ab
 - p_id → Vorname, Nachname, Alter ...
 - proj_id → Kunde, Status, ...
- Schlüsselkandidaten
 - Minimale Menge von Attributen, die alle anderen Attribute einer Tabelle funktional bestimmen
- Zerlegung nach funktionalen Abhängigkeiten
 - Datenbankentwurf
 - Ziel: Redundanzfreies Schema ohne Anomalien
 - Aber: Performanzsteigerung macht oftmals Denormalisierung notwendig

Normalformen

- Relationenschema R , Schlüsselkandidaten P
- Erste Normalform (1NF)
 - Alle Attribute von R sind atomar (Adresse!)
 - Kann man nicht automatisch überprüfen
 - Designentscheidung mit Auswirkung auf alle anderen NFs
- Zweite Normalform (2NF)
 - R ist in 1NF
 - Kein Attribut A , A nicht Teil eines Schlüssels, hängt von Teilmenge $P' \subset P$ funktional ab
 - Schlecht: `lehrt(p_id, s_id, termin, stud_name)`
- Dritte Normalform (3NF)
 - R in 1NF
 - Kein Attribut A hängt von einem $A' \notin P$ ab
 - Schlecht: `mitarbeiter(p_id, plz, ort)`