BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Grundlagen des relationalen Datenmodells

Medieninformatik Bachelor
Modul 9:
Datenbanksysteme

Prof. Dr. Petra Sauer, Prof. Dr. habil. Alexander Löser

Sie erinnern sich: Ihre Aufgaben

- **Aufgabe 1 Anfragen & Modellierung“**
  Denken Sie mal darüber nach, welche Anfragen Sie an die AOL Daten stellen möchten. Bitte Sie bitte ein logisches und physisches Schema zur Beantwortung dieser Anfragen.
- **Aufgabe 2 „SQL und Anfrageausführung“**
  Bitte formulieren Sie für Ihre Analyseideen aus 1.) die SQL Anfragen. Sie verstehen auch Möglichkeiten der Anfrageausführung bzw. Optimierung.
- **Aufgabe 3 „Datenintegration“**
  Zur Ausführung der Ausführung fehlen Ihnen noch externe Daten, z.B. aus dem Internet Archive, DMOZ oder Freebase.org. Bitte ergänzen Sie Ihr Schema und die Datenbasis.
- **Aufgabe 4 „Analyse, Erkenntnisgewinn und Wert“**
  Stellen Sie in 5 Minuten die wichtigsten Erkenntnisse aus den Daten vor. Bewerten Sie den Erkenntnisgewinn, z.B. gegenüber Ihren Kommilitonen oder der Literatur! Welche Erkenntnisse hätten einen kommerziellen Wert?

Die Themen

- Was sind Datenbanken?
  - Motivation, Historie, Datenunabhängigkeit, Einsatzgebiete
- Datenbankentwurf im ER-Modell & Relationaler Datenbankentwurf
  - Entities, Relationships, Kardinalitäten, Diagramme
  - Relationales Modell, ER -> Relational, Normalformen, Transformationseigenschaften
- Relationale Algebra & SQL
  - Kriterien für Anfragesprachen, Operatoren, Transformationen
  - SQL DDL, SQL DML, SELECT … FROM … WHERE …
- Datenintegration & Transaktionsverwaltung
  - JDBC, Cursor, ETL
  - Mehrbenutzerbetrieb, Serialisierbarkeit, Sperrprotokolle, Fehlerbehandlung, Isolationsebenen in SQL
- Ausblick
  - Map/Reduce, HDFS, Hive …
  - Wert von Daten

RDM – Operationen der Relationenalgebra

Basisoperationen (5+1 $\sigma,\pi,\cup,\times,-,\rho$)

- Projektion (Attributauswahl π)
- Selektion (Tupelauswahl σ)
- Kartesisches Produkt (Kreuzprodukt $\times$)
- Vereinigung (Union $\cup$)
- Differenz (Except / Minus -)
- Umbenennung (ρ)

Aus Basisoperatoren können weitere Mengenoperationen abgeleitet werden

- Vereinigung ($\cup$) bzw. Schnitt ($\cap$)
- Natürlicher Join ($\bowtie$)
- Theta-Join ($\bowtie$)
- Division (/)
- Symmetrische Differenz

Es gibt auch Erweiterungen, wie Duplikateliminierung, Aggregation, Gruppierung, Sortierung, erweiterte Projektion, Outer Join, Outer Union, Semijoin

… und Sichten bzw. einen Index

Duplikateliminierung (duplicate elimination, δ)

- Wandelt eine Multimenge in eine Menge um.
  - Durch Löschen aller Kopien von Tupeln
  - δ(R)

R

A	B
1	2
3	4
1	2
1	2

δ(R)

A	B
1	2
3	4

```
SELECT DISTINCT column_name(s) ...
FROM table_name(s)
WHERE (predicates);
```

Aggregation

- Aggregation fasst Werte einer Spalte zusammen.
  - Operation auf einer Menge oder Multimenge atomarer Werte (nicht Tupel)
  - Summe (SUM)
  - Durchschnitt (AVG)
  - Minimum (MIN) und Maximum (MAX)
    - Lexikographisch für nicht-numerische Werte
  - Anzahl (COUNT)
    - Doppelte Werte gehen auch doppelt ein.
    - Angewandt auf ein beliebiges Attribut ergibt dies die Anzahl der Tupel in der Relation.

- SUM(B) = 10
- AVG(A) = 1,5
- MIN(A) = 1
- MAX(B) = 4
- COUNT(A) = 4

R

A	B
1	2
3	4
1	2
1	2

```
SELECT COUNT(expression)
FROM tables
WHERE predicates;
```

Gruppierung (grouping, γ)

Partitionierung der Tupel einer Relation gemäß ihrer Werte in einem oder mehr Attributen. Hauptzweck: Aggregation auf Teilen einer Relation (den sogenannten ‚Gruppen').

- γL(R) wobei L eine Menge von Attributen ist. Ein Element in L ist entweder
  - ein Gruppierungsattribut nach dem gruppiert wird
  - oder ein Aggregationsoperator auf ein Attribut von R (inkl. neuem Namen für das aggregierte Attribut)
- Ergebnis wird wie folgt konstruiert:
  - Partitioniere R in Gruppen, wobei jede Gruppe gleiche Werte im Gruppierungsoperator hat
    - Falls kein Gruppierungsoperator angegeben: Ganz R ist die Gruppe
  - Für jede Gruppe erzeuge ein Tupel mit
    - Wert der Gruppierungsattribute
    - Aggregierte Werte über alle Tupel der Gruppe

Gruppering Beispiel in SQL

Für jedes Jahr soll die Summe, der Durchschnitt, der minimale und der maximale Umsatz (Spalte UMSATZ_EUR) ermittelt werden. Das erledigt eine ganz normale SQL-Abfrage mit Aggregatsfunktion und GROUP BY.

JAHR	MONAT	UMSATZ_EUR	WERBUNG_TV_EUR	WERBUNG_ZEITG_EUR	ANZAHL_VERKAEUFER
2004	11	130000	14000	400	10
2004	12	140000	13200	700	10
2005	1	100000	9000	1000	10
2005	2	103000	9500	900	10
2005	3	112000	10000	1500	10
2005	4	90000	12000	1000	8
2005	5	98000	13000	1000	8
2005	6	70000	4000	5000	9
2005	7	109010	12000	2000	15
2005	8	120000	10000	500	15
2005	9	130000	9000	1000	15
2005	10	150000	15000	1000	15
2005	11	170000	18000	1000	15
2005	12	200000	18000	1000	15
2006	1	160000	10000	1000	10
2006	2	140000	10200	1500	10

```
select jahr, sum(umsatz_eur), avg(umsatz_eur), min(umsatz_eur), max(umsatz_eur)
from verkauf group by jahr
order by 1
```

JAHR	SUM(UMSATZ_EUR)	AVG(UMSATZ_EUR)	MIN(UMSATZ_EUR)	MAX(UMSATZ_EUR)
2004	270000,0	135000,0	130000,0	140000,0
2005	1452010,0	121000,8	70000,0	200000,0
2006	300000,0	150000,0	140000,0	160000,0

http://www.oracle.com/webfolder/technetwork/de/community/apex/tipps/aggregateFunctions/index.html

Gruppierung – Beispiel in relat. Algeba

Für jedes Jahr soll die Summe, der Durchschnitt, der minimale und der maximale Umsatz (Spalte UMSATZ_EUR) ermittelt werden. Das erledigt eine ganz normale SQL-Abfrage mit Aggregatsfunktion und GROUP BY.

- Gegeben: SpieltIn(Titel, Jahr, SchauspName)
- Gesucht: Für jeden Schauspieler, der in mindestens 3 Filmen spielte, das Jahr des ersten Filmes.
- Idee
  - Gruppierung nach SchauspName
  - Minimum vom Jahr und Count von Titeln
  - Selektion nach Anzahl der Filme
  - Projektion auf Schauspielername und Jahr
- πSchauspName,MinJahr(σAnzahlTitel$\geq$3(γSchauspName,MIN(Jahr)$\rightarrow$ MinJahr,COUNT(Titel)$\rightarrow$AnzahlTitel)))

Gruppierung – mit HAVING

HAVING ist ein Prädikat das wir in einer Gruppierung anwenden. Das Prädikat lässt nur die Tuple einer Aggregation (COUNT, SUM, AVG etc.) zu, deren Werte der Bedingung des Prädikats entsprechen.

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Beispiel: Namen von "Employees" mit mehr als 25 "orders":

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
WHERE LastName='Davolio' OR LastName='Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

http://www.w3schools.com/sql/trysql.asp?filename=trysql_select_having_where

Erweiterte Projektion

Wie können wir bei einer Projektion gleich Ausdrücke berechnen lassen?

- Motivation: Mehr Fähigkeiten in den Projektionsoperator geben.
  - Vorher: $\pi L(R)$ wobei L eine Attributliste ist
  - Nun: Ein Element von L ist eines dieser drei Dinge
    - Ein Attribut von R (wie vorher)
    - Ein Ausdruck X→Y wobei X ein Attribut in R ist und Y ein neuer Name ist.
    - Ein Ausdruck E→Z, wobei E ein Ausdruck mit Konstanten, arithmetischen Operatoren, Attributen von R und String-Operationen ist und Z ein neuer Name ist.
      - A1+A2 → Summe
      - Vorname || Nachname → Name

Erweiterte Projektion – Beispiel

A	B	C
0	1	2
0	1	2
3	4	5

R

$\pi_{A,B+C \rightarrow X}(R)$

A	X
0	3
0	3
3	9

Duplikate bleiben erhalten

$\pi_{B-A \rightarrow X, C-B \rightarrow Y}(R)$

X	Y
1	1
1	1
1	1

Neue Duplikate können entstehen

Generalisierte Projektion

- Die Kombination von Gruppierung und Aggregation wird auch als generalsierte Projektion angesehen
  - Duplikatelimierung während Gruppierung
  - Aggregation der eliminierten Werte

- Generalisierte Projektion auf G mit Aggregation Agg von Attribut A:

  - $\pi G,Agg(A) = \pi G,A((\gamma G,Agg(A) \rightarrow A(R))$

Beispiel: Generalisierte Projektion

Interessant, die Metzger haben also das meiste Gold. Warum auch immer... Wie viel Gold haben im Durchschnitt die einzelnen Bewohnergruppen je nach Status (friedlich, böse, gefangen)?

SELECT status, AVG(bewohner.gold)
FROM bewohner
GROUP BY status

status	AVG(bewohner.gold)
?	70.0
boese	1512.85714285714
friedlich	706.363636363636
gefangen	490.0

Yeah!

Sortierung (sorting, τ)

- τL(R) wobei L eine Attributliste aus R ist.
  - Falls L = (A1,A2,…,An) wir zuerst nach A1, bei gleichen A1 nach A2 usw. sortiert.
- Wichtig: Ergebnis der Sortierung ist keine Menge, sondern eine Liste.
  - Deshalb: Sortierung ist letzter Operator eines Ausdrucks. Ansonsten würden wieder Mengen entstehen und die Sortierung wäre verloren.
  - Trotzdem: Es macht manchmal auch Sinn zwischendurch zu sortieren.

Beispiel: ORDER BY

Das ist viel zu wenig. Schauen wir mal die gesamten sowie durchschnittlichen Goldvorräte der einzelnen Berufe an:

> SELECT beruf, SUM(bewohner.gold), AVG(bewohner.gold) FROM bewohner GROUP BY beruf ORDER BY AVG(bewohner.gold)

beruf	SUM(bewohner.gold)	AVG(bewohner.gold)
Erntehelfer	10	10.0
?	70	70.0
Kaufmann	250	250.0
Hufschmied	390	390.0
Waffenschmied	790	395.0
Hoerbuchautor	420	420.0
Pilot	490	490.0
Baecker	1750	583.333333333333
Schmied	1250	625.0
Haendler	2130	710.0
Metzger	11370	2842.5

SQL IN

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1*,*value2*,...);

IN Operator Example

The following SQL statement selects all customers with a City of "Paris" or "London":

Example

```
SELECT * FROM Customers
WHERE City IN ('Paris','London');
```

Try it yourself »

http://www.w3schools.com/sql/sql_in.asp

Verbund (aka Join) – Seite 1

- **Cross Join:** Berechnen der Kreuztabelle, Ergebnis meistens nutzlos
- **Inner Join (aka Equivalent Join):** Verbindet Datensätze aus zwei Tabellen, sobald ein gemeinsames Feld dieselben Werte enthält.
- **Theta Join (Non-Equivalent-Join):** Wie Equi Join, vergleicht aber Inhalt der Attribute i und j mit einer beliebigen Formel Theta(i,j), also z.B. <,>, <>, .
- **Natural Join:** Verknüpft zwei Tabellen über die Gleichheit der Felder, in Spalten mit gleichem Namen. Spalten mit gleichem Namen werden im Ergebnis nur einmal angezeigt. Haben die Tabellen keine Spalten mit gleichem Namen, wird der Natural Join automatisch zum Cross Join.
- **Semi Join**: Natural Join von zwei Tabellen mit anschließender Projektion auf die Attribute der ersten Tabelle.

Siehe http://www.hdm-stuttgart.de/~riekert/lehre/db-kelz/chap7.htm#innerjoin

Verbund (aka Join) – Seite 2

- **Left Join (aka Left Outer Join):** schließen alle Datensätze aus der ersten (linken) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der zweiten (rechten) Tabelle existiert.
- **Right Join (Right Outer Join):** schließen alle Datensätze aus der zweiten (rechten) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der ersten (linken) Tabelle existiert.
- **Full Join (Full Outer Join):** Eine Kombination von Left Outer Join und Right Outer Join.
- **Union Join (aka Outer Union)** : Es werden Datensätze beider Tabellen aufgenommen. Sie werden aber nicht über eine Bedingung verknüpft.
- **Self Join:** beliebiger Join, der 2xdie gleiche Tabelle benutzt.

Siehe http://www.hdm-stuttgart.de/~riekert/lehre/db-kelz/chap7.htm#innerjoin

AUSGEWÄHLTE VERTIEFUNGEN ZU JOINS

Inner Theta Join mit LIKE

Theta Joins können auch mit dem LIKE Operator verwendet werden. Das ist hilfreich um in einem String die Existenz eines Substrings mit Like zu testen und nur diese Tuples in die Ergebnismenge aufzunehmen.

```
SELECT *
FROM tableone
JOIN cities on querydata.query
LIKE ('%' || cities.nametext || '%' )
```

- Der underscore (_) bildet genau einen Character ab.
- Das Prozentzeichen (%) bildet 0 oder mehrere Patterns ab, kann aber nicht auf NULL testen.

Outer Joins (Äußere Verbünde, |⋈|)

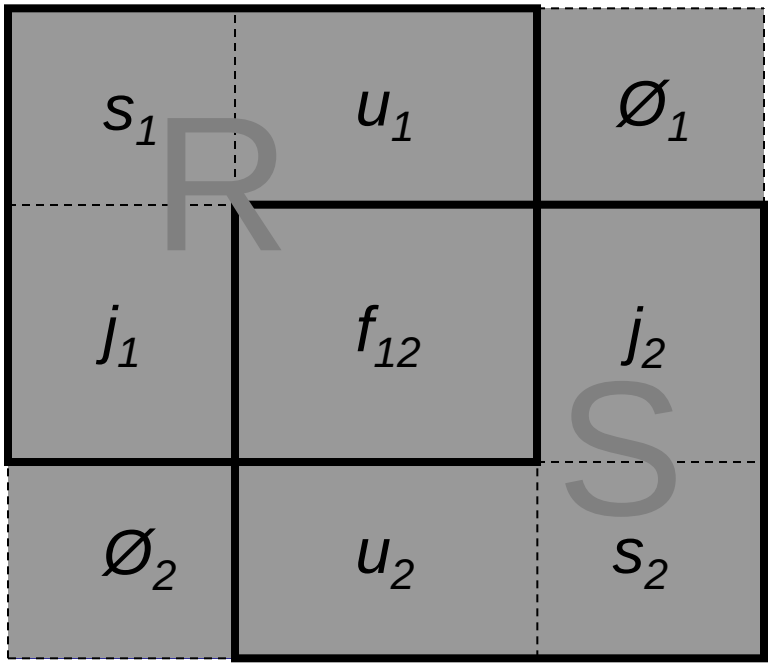
- Übernahme von „dangling tuples“ (!!!**referential integrity constraints !!)** in das Ergebnis und Auffüllen mit Nullwerten (padding)

- Full outer join
  - Übernimmt alle Tupel beider Operanden
  - R |⋈| S
- Left outer join (right outer join)
  - Übernimmt alle Tupel des linken (rechten) Operanden
  - R |⋈ S (bzw. R ⋈| S)

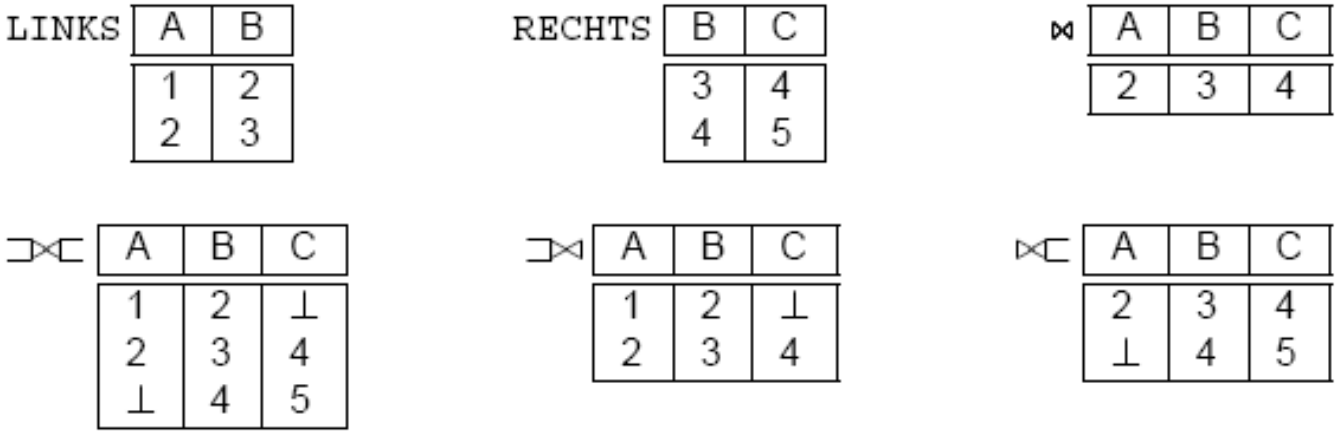
- Andere Schreibweisen:
  - Herkömmlicher Join = „Inner join“

Outer Joins

- R ⋈ S
- R |⋈ S
- R ⋈| S
- R |⋈| S

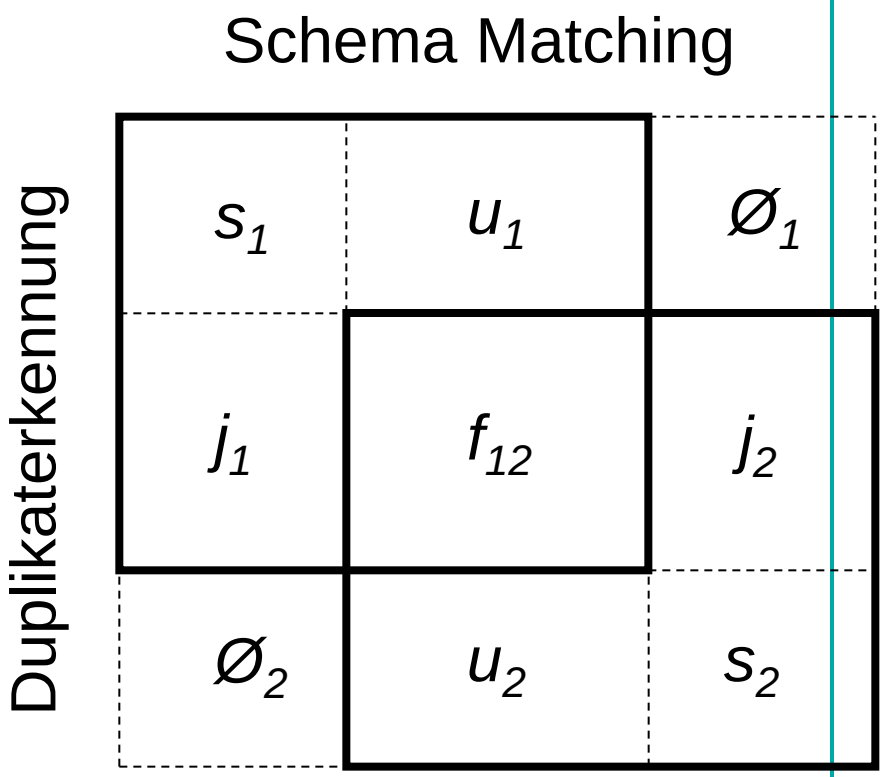


Outer Joins


LINKS
A B
1 2
2 3
RECHTS
B C
3 4
4 5
⋈
A B C
2 3 4
⟕
A B C
1 2 ⊥
2 3 4
⟗
A B C
1 2 ⊥
2 3 4
⟖
A B C
2 3 4
⊥ 4 5

Outer Joins und Informationsintegration

- Ziel: Möglichst viele Informationen
  - Viele Tupel
  - Viele Attribute
- Problem
  - Überlappende Attribute erkennen
    - = Schema Matching
  - Überlappende Tupel erkennen
    - = Duplikaterkennung



Outer Union (⊎)

- Wie Vereinigung, aber auch mit inkompatiblen Schemata
  - Schema ist Vereinigung der Attributmengen
  - Fehlende Werte werden mit Nullwerten ergänzt.

R

A	B	C
1	2	3
6	7	8
9	7	8

S

B	C	D
2	5	6
2	3	5
7	8	10

R ⊎ S

A	B	C	D
1	2	3	⊥
6	7	8	⊥
9	7	8	⊥
⊥	2	5	6
⊥	2	3	5
⊥	7	8	10

Sub-Queries

Sie können SQL Anfragen ineinander verschachteln – Sub-Queries. Wir unterscheiden zwischen unkorrelierten und korrelierten Sub-Queries

```
SELECT department_id, MIN (salary)
FROM employees
GROUP BY department_id
HAVING MIN (salary) <
(
        SELECT AVG (salary) FROM employees
)
```

Nicht-korrelierende Sub-Query

```
SELECT EMPLOYEE_ID, salary, department_id
FROM employees E
WHERE salary >
(
        SELECT AVG(salary)
        FROM EMP T
        WHERE E.department_id = T.department_id
)
```

Korrelierende Sub-Query. Für jede Zeile der inneren Anfrage wird die äußere Anfrage *auch* ausgeführt. Die innere Anfrage hängt von der äußeren Anfrage ab.

http://www.tutorialspoint.com/sql_certificate/subqueries_to_solve_queries.htm

Create your Own Exam: Relationale Algebra

- Bitte erstellen Sie eine Multiple Choice Aufgabe zum Thema Relationale Algebra
  - Formulieren Sie eine Frage und 3 Antworten (A, B, C)
  - Davon sollte mindestens eine Antwort richtig und mindestens eine Antwort falsch sein
- Geben Sie die Aufgabe an Ihren rechten Nachbarn. Diskutieren Sie gemeinsam und markieren Sie die richtigen Lösungen
- Geben Sie am Ende der Vorlesung Ihre Aufgabe bei mir ab

5 min

The bar exam after the bar exam.

RDM – Operationen der Relationenalgebra

Basisoperationen (5+1 $\sigma,\pi,\cup,\times,-,\rho$)

- Projektion (Attributauswahl π)
- Selektion (Tupelauswahl σ)
- Kartesisches Produkt (Kreuzprodukt $\times$)
- Vereinigung (Union $\cup$)
- Differenz (Except / Minus -)
- Umbenennung (ρ)

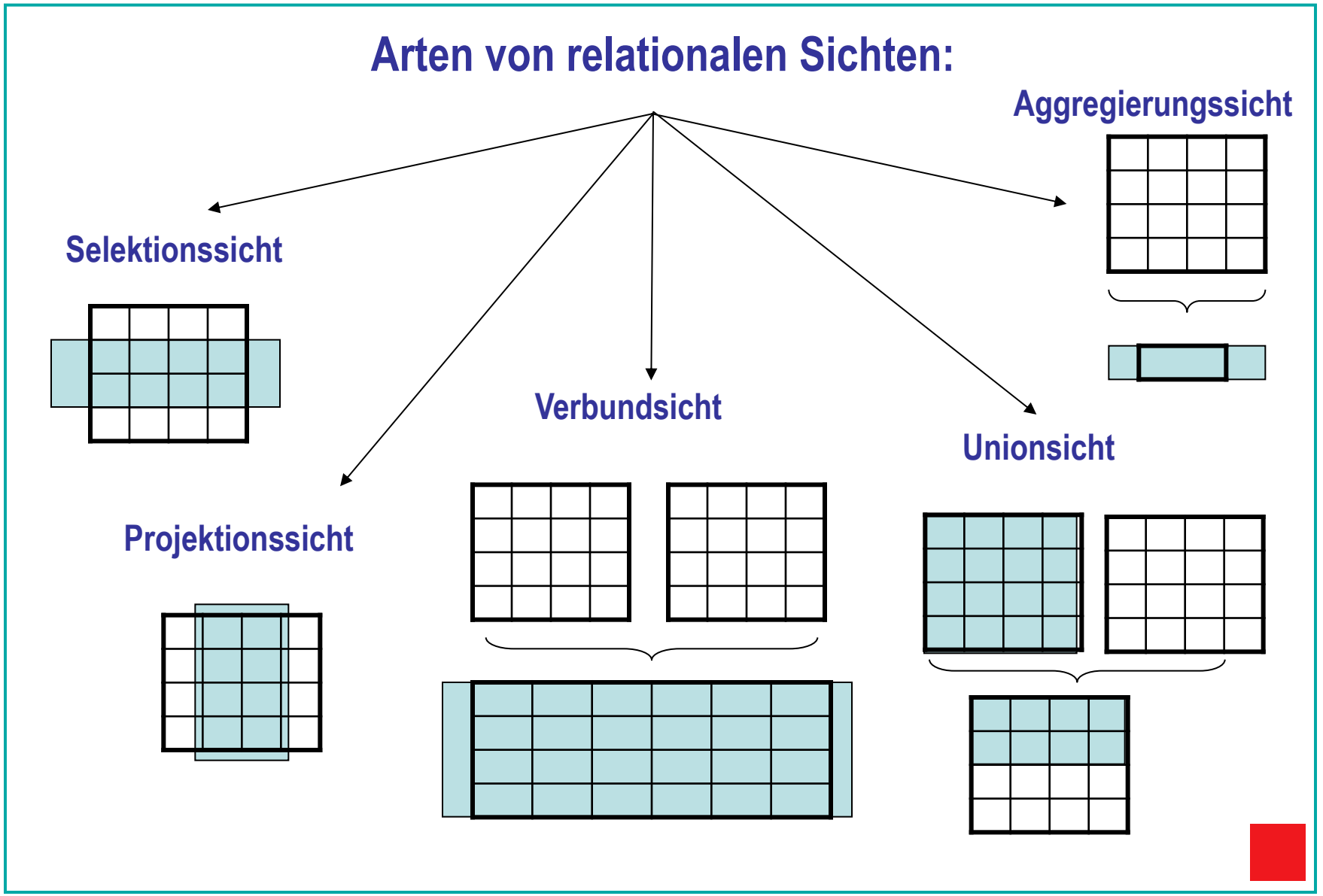
Aus Basisoperatoren können weitere Mengenoperationen abgeleitet werden

- Vereinigung ($\cup$) bzw. Schnitt ($\cap$)
- Natürlicher Join ($\bowtie$)
- Theta-Join ($\bowtie$)
- Division (/)
- Symmetrische Differenz

Es gibt auch Erweiterungen, wie Duplikateliminierung, Aggregation, Gruppierung, Sortierung, erweiterte Projektion, Outer Join, Outer Union, Semijoin

… und Sichten bzw. einen Index

Arten von Sichten



Sichten und Virtuelle Relationen

- Relationen aus **`CREATE TABLE`** Ausdrücken existieren tatsächlich (materialisiert, physisch) in der Datenbank.
  - Persistenz
  - Updates sind möglich

- Die Daten aus Sichten (*views*) existieren nur virtuell.
  - Sichten entsprechen Anfragen, denen man einen Namen gibt. Sie wirken wie physische Relationen.
  - Funktionieren wie verschachtelte Tabellenausdrücke mit festem Namen
  - Manche Systeme lassen auf bestimmte Arten von Views auch Updates zu.

Sichten in SQL

- `CREATE VIEW` *Name* `AS` *Anfrage*

- Beispiel:

```
CREATE VIEW ParamountFilme AS
     SELECT Titel, Jahr
     FROM Film
     WHERE StudioName = 'Paramount Pictures';
```

  - Beliebig komplexe Anfragen möglich!

- Semantik
  - Bei jeder Anfrage an die Sicht wird die SQL Anfrage der Sicht ausgeführt.
  - Die ursprüngliche Anfrage verwendet das Ergebnis als Relation.

- Daten der Sicht ändern sich mit der Änderung der zugrundeliegenden Relationen.

- Entfernen der Sicht: `DROP VIEW ParamountFilme`
  - Basisdaten bleiben unverändert.

Anfragen an Sichten

- ```
  CREATE VIEW ParamountFilme AS
        SELECT Titel, Jahr
        FROM Film
        WHERE StudioName = 'Paramount Pictures';
  ```

- ```
  SELECT Titel
  FROM ParamountFilme
  WHERE Jahr = 1994;
  ```

- Umwandlung der ursprünglichen Anfrage in eine Anfrage an Basisrelationen
  - ```
    SELECT Titel
    FROM Film
    WHERE StudioName = 'Paramount Pictures' AND Jahr = 1994;
    ```
  - Übersetzung durch DBMS

- Anfrage zugleich an Sichten und Basisrelationen möglich
  - ```
    SELECT DISTINCT Schauspieler
    FROM ParamountFilme, spielt_in
    WHERE Titel = FilmTitel AND Jahr = FilmJahr;
    ```

Anfragen an Sichten

- Film(Titel, Jahr, Laenge, inFarbe, StudioName, ProduzentID)
- Manager(ID, Name, Adresse, Gehalt)

- ```
  CREATE VIEW FilmProduzenten AS
       SELECT Titel, Name
       FROM Film, Manager
       WHERE ProduzentID = ID;
  ```

- Anfrage
  - ```
    SELECT Name
    FROM FilmProduzenten
    WHERE Titel = ,Burn After Reading'
    ```

- Übersetzung
  - ```
    SELECT Name
    FROM Film, Manager
    WHERE ProduzentID = ID
    AND Titel = 'Burn After Reading';
    ```
    ```

Umbenennung in Sichten

- Umbenennung von Attributen
  - ```
    CREATE VIEW FilmeProduzenten(FilmTitel,
    Produzentenname) AS
      SELECT Titel, Name
      FROM Film, Manager
      WHERE ProduzentID = ID;
    ```
- Oder Sicht einfach nur zur Umbenennung
  - ```
    CREATE VIEW Movies(title, year, length, inColor,
    studio, producerID) AS
            SELECT *
            FROM Film;
    ```

Diskussion

- Vorteile
  - Vereinfachung von Anfragen
  - Strukturierung der Datenbank
  - Logische Datenunabhängigkeit
    - Sichten stabil bei Änderungen der Datenbankstruktur
  - Beschränkung von Zugriffen (Datenschutz)
  - Später: Optimierung durch materialisierte Sichten
- Probleme
  - Automatische Anfragetransformation schwierig
  - Durchführung von Änderungen auf Sichten
  - Updatepropagierung für materialisierte Sichten

Exkurs: Materialisierte Sichten

- Viele Anfragen an eine Datenbank wiederholen sich häufig
  - Business Reports, Bilanzen, Umsätze
  - Bestellungsplanung, Produktionsplanung
  - Kennzahlenberechnung

- Viele Anfragen sind Variationen mit gemeinsamem Kern
  - → Einmaliges Berechnen der Anfrage als Sicht
  - → Automatische, transparente Verwendung in folgenden Anfragen

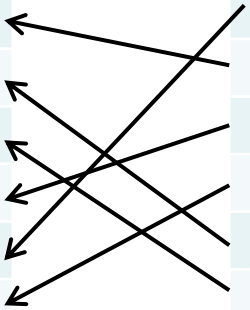
- ➔ Materialisierte Sicht (*Materialized View*, MV)
  - Auch: Materialized Query Table (MQT)

Indexe: Motivation

- Tabellen speichern Tupel normalerweise ohne Ordnung hintereinander.
- Suche nach bestimmten Tupeln (z.B. die Prädikate der WHERE Klausel erfüllen) standardmäßig durch einen "Table-Scan", ein Lesen der gesamten Tabelle.
  - Wegwerfen aller nicht qualifizierenden Tupel
- Index invertiert das Layout: Er beschreibt, an welchen Positionen in der Tabelle die Tupel mit bestimmten Werten liegen.
- Suche nach Tupeln mittels Index: Durchsuche Index nach dem zu suchenden Wert oder Wertebereich. Folge den Zeigern zu den tatsächlichen Tupeln.

Tabelle „person“

Name	Geburtsdatum	Geburtsort
Peter	03.06.1985	Berlin
Lena	16.11.1983	Münster
Tim	26.09.1990	Stuttgart
Heike	04.07.1984	Köln
Susanne	17.05.1987	Berlin
Thomas	13.09.1981	München



Index „gebOrtIDX“

TID	Geburtsort
•	Berlin
•	Berlin
•	Köln
•	München
•	Münster
•	Stuttgart

Operationen die Indexe verwenden

- Suchen qualifizierender Tupel in einer SELECT Anfrage (Optional via Index)
  - Prädikate der WHERE Klausel
- Joins zwischen Tabellen (Optional via Index)
  - Für jedes Tupel der einen Tabelle, suche über einen Index auf dem Join-Key die passenden Tupel der anderen Tabelle
- Einzigartigkeit des Primary Key (zwingend via Index)
  - Bei INSERT oder UPDATE, checke den Index, ob schon ein Tupel mit gleichem Key enthalten ist
- Einzigartigkeit sekundärer Schlüssel (in einigen Systemen zwingend via Index)
  - Bei INSERT oder UPDATE, checke den Index, ob schon ein Tupel mit gleichem Key enthalten ist
- Foreign Keys (in einigen Systemen zwingend via Index)
  - Bei Veränderungen mit 'CASCADE' Option werden via Index betroffene Tupel gefunden.

Indexstrukturen

- Verschiedene Index-Strukturen existieren.
  - Bäume
  - Bitmaps
  - Hash-Tables
- Unterschiedliche Eigenschaften machen sie für unterschiedliche Szenarien interessant.
  - Read-Only/Mostly Szenarien
  - Update-Intensive Szenarien

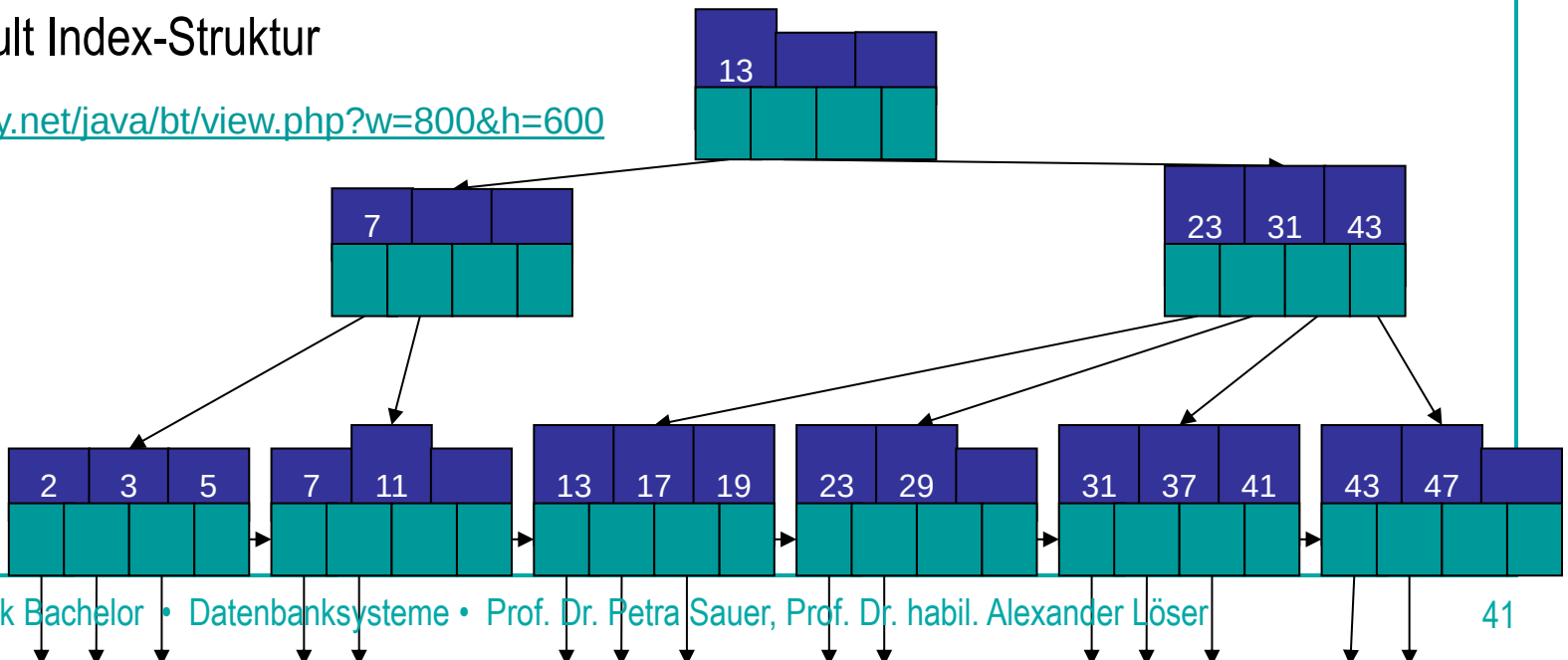
B-Trees

- Strikt Balancierte Bäume mit Ordnung über die Schlüssel
  - Punkt-Prädikate und Range-Prädikate
  - Zusammengesetzte Schlüssel für mehrere Spalten
    - Ranges nur in den ersten Spalten
- Ein Knoten ist eine I/O Einheit
  - Design ist die Anzahl der I/O Requests zu minimieren
  - Veränderungen sind hochgradig lokal

➔ Gute Indexstruktur für Update-Intensive Szenarios

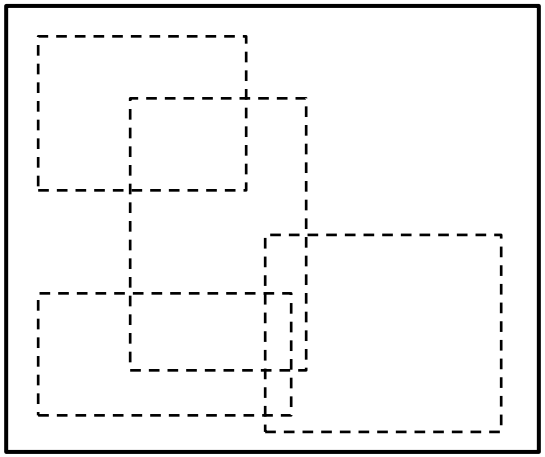
➔ Default Index-Struktur

http://slady.net/java/bt/view.php?w=800&h=600



R-Tree (Region Tree)

- Mehrdimensionale Variante des B-Tree
  - Zerlegt den mehrdimensionalen Raum in Regionen
- Dimensionen können unabhängig von einander eingeschränkt werden
- ➔ Nützliche Indexstruktur für Räumliche oder Geographische Daten



Bitmap-Indexe

- Verwendet eine "Bitmap" für jeden unterschiedlichen Wert in einer Spalte
  - Effiziente Kompression der Indexe möglich

- Sehr teuer bei Updates
- Sehr effizient bei Punkt-Anfragen
- Sehr effizient bei der Verknüpfung mit anderen Indexen

➔ Guter Index für Data-Warehousing Szenarien

➔ Read-Only (mostly) Szenarien

Bitmaps auf Gender
M: 01110
F: 10001

emp-id	gender	salary
1	F	L3
2	M	L2
3	M	L1
4	M	L3
5	F	L1

Clustered Index

- Tabelle wird normalerweise unabhängig vom Index organisiert.
  - Tupel mit gleichem Schlüssel liegen physikalisch an unterschiedlichen Orten.

- Bei einem Clustered-Index wird die Tabelle entsprechend dem Index organisiert.
  - Erhöht beim Zugriff die I/O Performanz
  - Nur ein Index kann Clustered sein.

Tabelle „person“

Name	Geburtsdatum	Geburtsort
Peter	03.06.1985	Berlin
Susanne	17.05.1987	Berlin
Heike	04.07.1984	Köln
Thomas	13.09.1981	München
Lena	16.11.1983	Münster
Tim	26.09.1990	Stuttgart

Index „gebOrtIDX“

TID	Geburtsort
•	Berlin
•	Berlin
•	Köln
•	München
•	Münster
•	Stuttgart

Index anlegen/löschen

Syntax:

```
CREATE INDEX indexname ON tablename ( columnname , )
       IN DBSpaceName
```

Beispiel:

```
CREATE INDEX gebOrtIDX ON person (geburtsort) IN tbSpc1
```

- Legt einen Standard B-Tree Index an
  - Für andere Index Typen:
  - … ON <table> (columns) USING BITMAP …
  - … ON <table> (columns) USING RTREE …
- Einrichten eines Index kann einige Zeit dauern
- In einigen Fällen legt das Datenbanksystem selber Indexe an.
  - Bsp: Informix legt immer B-Tree Indexe für Primary Keys and Foreign Keys an.

Sie erinnern sich: Ihre Aufgaben

- **Aufgabe 1 Anfragen & Modellierung“**
  Denken Sie mal darüber nach, welche Anfragen Sie an die AOL Daten stellen möchten. Bitte Sie bitte ein logisches und physisches Schema zur Beantwortung dieser Anfragen.
- **Aufgabe 2 „SQL und Anfrageausführung“**
  Bitte formulieren Sie für Ihre Analyseideen aus 1.) die SQL Anfragen. Sie verstehen auch Möglichkeiten der Anfrageausführung bzw. Optimierung.
- **Aufgabe 3 „Datenintegration“**
  Zur Ausführung der Ausführung fehlen Ihnen noch externe Daten, z.B. aus dem Internet Archive, DMOZ oder Freebase.org. Bitte ergänzen Sie Ihr Schema und die Datenbasis.
- **Aufgabe 4 „Analyse, Erkenntnisgewinn und Wert“**
  Stellen Sie in 5 Minuten die wichtigsten Erkenntnisse aus den Daten vor. Bewerten Sie den Erkenntnisgewinn, z.B. gegenüber Ihren Kommilitonen oder der Literatur! Welche Erkenntnisse hätten einen kommerziellen Wert?

Die Themen

- Was sind Datenbanken?
  - Motivation, Historie, Datenunabhängigkeit, Einsatzgebiete
- Datenbankentwurf im ER-Modell & Relationaler Datenbankentwurf
  - Entities, Relationships, Kardinalitäten, Diagramme
  - Relationales Modell, ER -> Relational, Normalformen, Transformationseigenschaften
- Relationale Algebra & SQL
  - Kriterien für Anfragesprachen, Operatoren, Transformationen
  - SQL DDL, SQL DML, SELECT … FROM … WHERE …
- Datenintegration & Transaktionsverwaltung
  - JDBC, Cursor, ETL
  - Mehrbenutzerbetrieb, Serialisierbarkeit, Sperrprotokolle, Fehlerbehandlung, Isolationsebenen in SQL
- Ausblick
  - Map/Reduce, HDFS, Hive …
  - Wert von Daten

Zusammenfassung und Ausblick

- Einführung
- Basisoperatoren
  - Selektion (σ)
  - Projektion (π)
  - Vereinigung ($\cup$)
  - Differenz (\ oder -)
  - Cartesisches Produkt ($\times$)
  - Umbenennung (ρ)
- Abgeleitete Operatoren
  - Join ($\bowtie$)
  - Schnitt ($\cap$)
  - Division (/)
- Erweiterte Operatoren
  - Duplikateliminierung
  - Generalisierte Projektion (Gruppierung und Aggregation)
  - Outer-Joins und Semi-Joins
  - Sortierung

In der nächsten Veranstaltung:
Query Optimierung