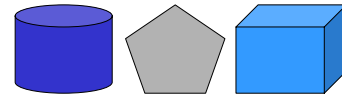


# Datenintegration

Datenintegration



## Kapitel 5: Anfrageverarbeitung

**Dr. Andreas Thor**  
**Sommersemester 2009**

**Universität Leipzig**  
**Institut für Informatik**  
<http://dbs.uni-leipzig.de>

1



## Inhalt

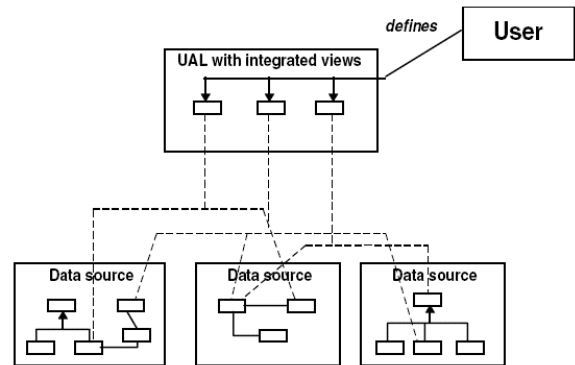
- Multidatenbanksprachen am Beispiel von SchemaSQL
  - Grundlegende Syntax inklusive Zugriff auf Metadaten
  - Horizontale Aggregation und dynamische Umstrukturierung
- Global-as-View (GaV) und Local-as-View (LaV)
  - GaV: Modellierung und Anfragebearbeitung
  - LaV: Modellierung
  - Vergleich
- LaV-Anfragebearbeitung
  - Query Containment
  - Answering Queries using Views
  - Quellen mit beschränkten Anfragemöglichkeiten

2

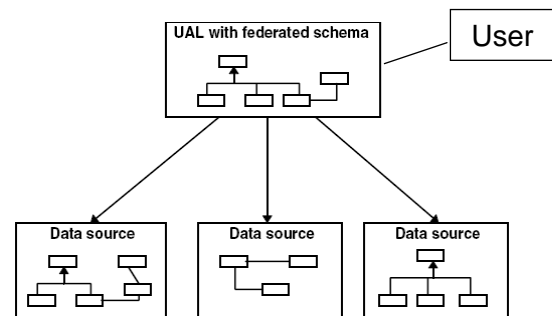


# Enge vs. lose Kopplung

- Lose Kopplung
  - Kein festes Schema
    - Nutzer müssen Semantik der Quellen kennen
    - Integrierte Sichten helfen
  - Multidatenbanksprachen



- Enge Kopplung
  - Globales / integriertes / föderiertes Schema
    - Nutzer stellen Anfrage bzgl. globalem Schema
    - Nutzer müssen Schema / Semantik der Quellen nicht kennen
  - Global-As-View (GaV)
  - Local-As-View (LaV)

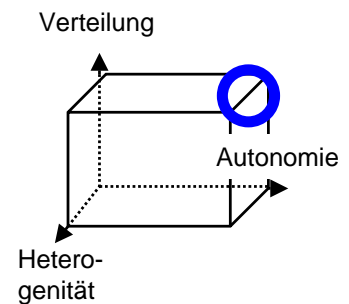


3



## Wdh: Multidatenbanken [VAh]

- Verteilt, autonom, und „etwas“ heterogen
  - Keine technische Heterogenität
  - Keine Datenmodellheterogenität
  - Schemata können strukturell und semantisch heterogen sein
  - Verteilte Systeme benutzen gleiche Techniken (RDBMS)
  - Autonomie bleibt bewahrt, aber Zugriff muss möglich sein (Kommunikationsautonomie)
- Kein globales Schema → Zugriff über Multidatenbanksprachen
- Ziel: Globaler Zugriff auf mehrere (lokale) Datenbanken
- Verwendung
  - Beantwortung von Anfragen
  - Erstellung integrierter Sichten

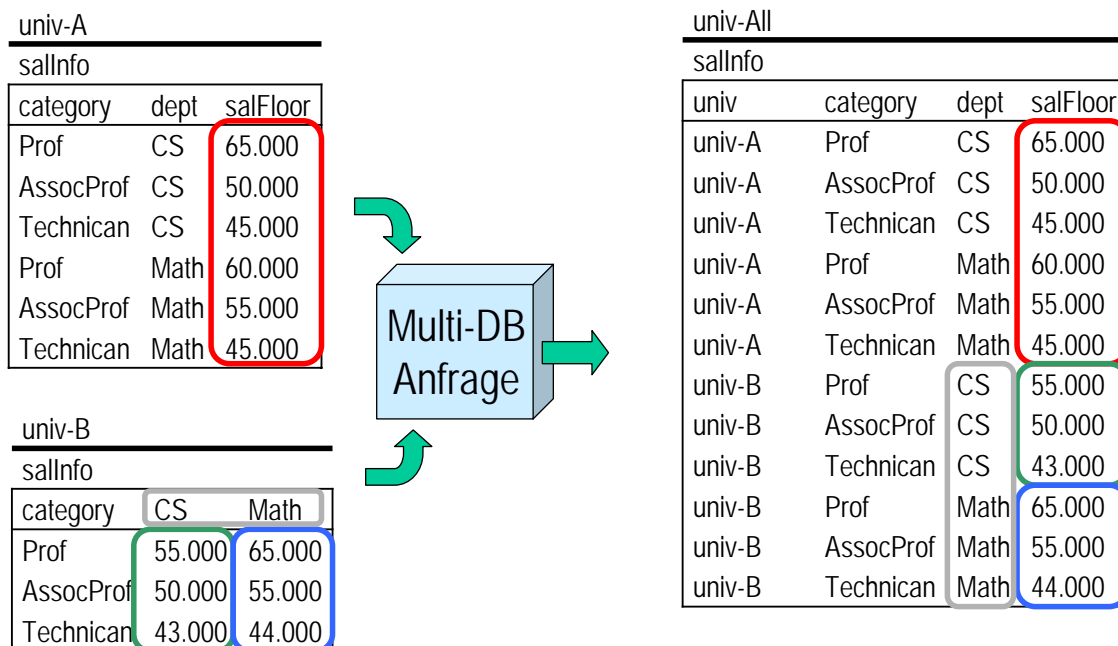


4



# Multidatenbanken: Beispiel

- Zwei Universitätsdatenbanken
- Ziel: Integrierte Sicht mittels "SQL-Definition"



5



# Multidatenbanksprachen

- Anforderungen
  - Schemaunabhängigkeit, d.h. Struktur darf nicht Ausdrucksfähigkeit beeinflussen
  - Umstrukturierung, d.h. Anfrageergebnisse müssen neue Struktur erhalten können
  - Verständlichkeit bei hoher Ausdrucksfähigkeit
  - Abwärtskompatibilität mit SQL
  - Implementierbar
    - Möglichst ohne Veränderung des DBMS
  - Alle Anfragen müssen in SQL / Programme übersetzbar sein
  - Effiziente Ausführung
- Beispiele
  - SchemaSQL [Lakshmanan et. al: SchemaSQL – A Language for Interoperability in Relational Multi-database Systems. VLDB'96]
  - FRAQL [Sattler et. al.: Adding Conflict Resolution Features to a Query Language for Database Federations. EFIS'00]
  - MSQL [Grant et. al.: Query Languages for Relational Multidatabases. VLDB Journal '93]
  - Kommerzielle Implementierungen (z.B. IBM Data Joiner / Information Integrator)

6



# Strukturelle Heterogenität

- Heterogenität erfordert Ausdrucksmächtigkeit in MultiDB-Sprache

- Vergleich von Modellelementen

- Relation vs. Attribut
- Attribut vs. Wert
- Relation vs. Wert

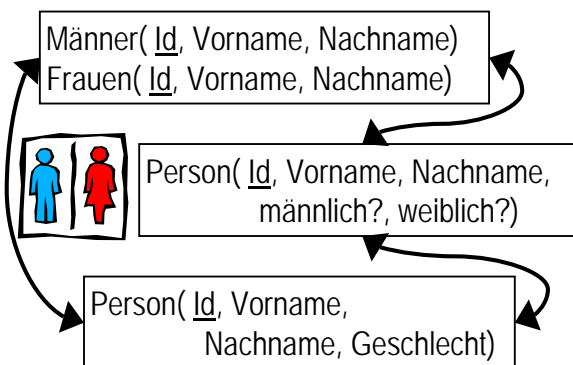
- Zugriff auf Namen

- Relationen
- Attribute

- Werte

- Behandlung unterschiedlicher Verteilung von Attributen

- Z.B. Normalisiert vs. Denormalisiert
- Fehlende/neue Attribute
- Geschachtelt vs. Fremdschlüssel



SchemaSQL

SQL

7



## Schema SQL

- Erweiterung von SQL

- Zugriff auf Tabellen in verschiedenen Schemata

- Daten und Metadaten werden gleich behandelt

- Umstrukturierungen innerhalb der Anfrage

- Daten zu Metadaten und umgekehrt

- Dynamische Sicht-Definition

- Ergebnisrelation hängt vom Zustand der Datenbank ab

- Horizontale Aggregation

- Über mehrere Spalten hinweg

- Überbrückung struktureller Heterogenität

8



# SchemaSQL: Syntax

- Grundlegende Syntax wie SQL
  - **SELECT ... FROM ... WHERE**
- Deklaration in FROM Klausel durch `<range> <var>`
- Variablen laufen über fünf verschiedene Wertbereichstypen
  - `->` Alle Datenbanknamen der Multidatenbank
  - `db->` Alle Relationennamen einer Datenbank `db`
  - `db::rel->` Alle Attributnamen einer Relation `rel` in `db`
  - `db::rel` Alle Tupel einer Relation `rel` in `db`
  - `db::rel.attr` Alle Werte eines Attributs `attr` in `rel` in `db`
- Weiterer Unterschied: Geschachtelte Deklarationen
  - Spätere Deklarationen referenzieren frühere
  - In SQL sinnlos

9



## Beispiel: Universitätsdatenbanken

univ-A

salInfo

category	dept	salFloor
Prof	CS	65.000
AssocProf	CS	50.000
Technican	CS	45.000
Prof	Math	60.000
AssocProf	Math	55.000
Technican	Math	45.000

univ-B

salInfo

category	CS	Math
Prof	55.000	65.000
AssocProf	50.000	55.000
Technican	43.000	44.000

univ-C

CS	
category	salFloor
Prof	60.000
AssocProf	55.000
Technican	42.000

Math

category	salFloor
Prof	70.000
AssocProf	60.000
Technican	46.000

univ-D

salInfo

dept	Prof	AssocProf	Technican
CS	75.000	60.000	40.000
Math	60.000	45.000	38.000

- Multidatenbank aus mehreren Universitätsdatenbanken
  - univ-A, ..., univ-D
- Informationen über Angestellte
  - Kategorie (category)
  - Gehalt (salInfo, salFloor)
  - Abteilung (dept)
- Informationen verteilt auf
  - Attributwerte
  - Attributnamen
  - Relationsnamen

10



# Anfragebeispiel 1

univ-A

salInfo

category	dept	salFloor
Prof	CS	65.000
AssocProf	CS	50.000
Technican	CS	45.000
Prof	Math	60.000
AssocProf	Math	55.000
Technican	Math	45.000

univ-B

salInfo

category	CS	Math
Prof	55.000	65.000
AssocProf	50.000	55.000
Technican	43.000	44.000

- Alle Abteilungen in univ-A, die Technikern mehr zahlen als gleiche Abteilungen in univ-B
- Lösungsweg:
  - Join zwischen beiden Tabellen (über welches Attribut?)
  - Selektionen jeweils auf `Technican`
  - Vergleich der Gehälter
- SchemaSQL-Anfrage

11



# Anfragebeispiel 2

univ-C

CS	
category	salFloor
Prof	60.000
AssocProf	55.000
Technican	42.000

Math

category	salFloor
Prof	70.000
AssocProf	60.000
Technican	46.000

univ-D

salInfo

dept	Prof	AssocProf	Technician
CS	75.000	60.000	40.000
Math	60.000	45.000	38.000

- Alle Abteilungen in univ-C, die Technikern mehr zahlen als gleiche Abteilungen in univ-D
- Lösungsweg:
  - Join zwischen beiden Tabellen (über welches Attribut?)
  - Selektionen jeweils auf `Technican`
  - Vergleich der Gehälter
- SchemaSQL-Anfrage

```

SELECT RelC
FROM univ-C-> RelC,
      univ-C::RelC C,
      univ-D::salInfo D
WHERE RelC = D.dept
AND C.category = `Technician`
AND C.salFloor > D.Technician
    
```

Tabellenname  
als Ausgabe

Geschachtelte  
Variablen

Iteration über  
Tupel beider  
Tabellen in univ-C

Join zwischen  
Relationennamen  
und Attributwert



## Anfragebeispiel 2: Ausführung (1)

univ-C

CS	
category	salFloor
Prof	60.000
AssocProf	55.000
Technican	42.000

Math	
category	salFloor
Prof	70.000
AssocProf	60.000
Technican	46.000

univ-D

salInfo			
dept	Prof	AssocProf	Technician
CS	75.000	60.000	40.000
Math	60.000	45.000	38.000

```
SELECT RelC
FROM   univ-C-> RelC,
       univ-C::RelC C,
       univ-D::salInfo D
WHERE  RelC = D.dept
AND    C.category = `Technician`
AND    C.salFloor > D.Technician
```

RelC	D

C

## Anfragebeispiel 2: Ausführung (2)

Kartesisches Produkt

RelC	C		D			
	category	salFloor	dept	Prof	AssoP	Techn
CS	Prof	60.000	CS	75.000	60.000	40.000
CS	Prof	60.000	MATH	60.000	45.000	38.000
CS	AssoP	55.000	CS	75.000	60.000	40.000
CS	AssoP	55.000	MATH	60.000	45.000	38.000
CS	Techn	42.000	CS	75.000	60.000	40.000
CS	Techn	42.000	MATH	60.000	45.000	38.000
MATH	Prof	70.000	CS	75.000	60.000	40.000
MATH	Prof	70.000	MATH	60.000	45.000	38.000
MATH	AssoP	60.000	CS	75.000	60.000	40.000
MATH	AssoP	60.000	MATH	60.000	45.000	38.000
MATH	Techn	46.000	CS	75.000	60.000	40.000
MATH	Techn	46.000	MATH	60.000	45.000	38.000

```
SELECT RelC
FROM   univ-C-> RelC, univ-C::RelC C, univ-D::salInfo D
WHERE  RelC = D.dept
AND    C.category = `Technician`
AND    C.salFloor > D.Technician
```



# Aggregation in SQL

- Aggregationsfunktionen
  - AVG, COUNT, SUM, MIN, MAX, (STDDEV, VARIANCE, ...)
- Gruppierung und Aggregation sind vertikal
  - Aggregation: Werte einer Spalte werden zusammengefasst
    - Beispiel: Anzahl aller Departments

univ-A		
salInfo		
category	dept	salFloor
Prof	CS	65,000
AssocProf	CS	50,000
Technican	CS	45,000
Prof	Math	60,000
AssocProf	Math	55,000
Technican	Math	45,000

- Gruppierung: Teilmengen von Werten einer Spalte werden zusammengefasst
  - Beispiel: Durchschnittliches Gehalt pro Category

18



## SchemaSQL: Horizontale Aggregation

- Ziel: Aggregation über Werte mehrerer Spalten
  - die evtl. in verschiedenen Tabellen liegen
- Beispiel 1: Durchschnittliches Gehalt in univ-B pro Kategorie für alle Abteilungen

```

SELECT  T.category, AVG(T.D)
FROM    univ-B::salInfo-> D,
        univ-B::salInfo T
WHERE   D <> `category`
GROUP BY T.category
    
```

Wird Liste von Werten

Iteration über alle Attribute

Iteration über alle Werte

univ-B		
salInfo		
category	CS	Math
Prof	55.000	65.000
AssocProf	50.000	55.000
Technican	43.000	44.000

- Beispiel 2: Durchschnittliches Gehalt in univ-C pro Kategorie für alle Abteilungen

univ-C	
CS	
category	salFloor
Prof	60.000
AssocProf	55.000
Technican	42.000

Math	
category	salFloor
Prof	70.000
AssocProf	60.000
Technican	46.000

20



# SchemaSQL: Blockweise Aggregation

- Ziel: Aggregation über einen Block
  - "Horizontal + Vertikal"
- Beispiel 3: Durchschnittliches Gehalt aller Angestellten pro Fakultät

```
SELECT  F.fname, AVG(T.C)
FROM    univ-D::salInfo-> C,
        univ-D::salInfo T,
        univ-D::faculty F
WHERE   C <> "dept"
AND     T.dept = F.dname
GROUP BY F.fname
```

univ-D			
salInfo			
dept	Prof	AssocProf	Technician
CS	75.000	60.000	40.000
Math	60.000	45.000	38.000

faculty	
dname	fname
CS	MatNat 1
Math	MatNat 1
Phys	MatNat 2

Aggregation über eine (Kategorie-) Liste von (Department-) Listen

22



# SchemaSQL: Umstrukturierung

- Erstellung integrierter Sichten hilfreich
- Anforderungen
  - Definition eines Output-Schemas (evtl. dynamisch)
  - Umstrukturierung der Daten
- Beispiel: Repräsentation der Daten von univ-B im Schema von univ-A und umgekehrt

```
CREATE VIEW BtoA AS
SELECT  T.category category,
        D dept,
        T.D salFloor
FROM    univ-B::salInfo-> D,
        univ-B::salInfo T
WHERE   D <> `category`
```

```
CREATE VIEW AtoB::salInfo(category, D) AS
SELECT  A.category, A.salFloor
FROM    univ-A::salInfo A, univ-A::A.dept D
```

univ-A		
salInfo		
category	dept	salFloor
Prof	CS	65.000
AssocProf	CS	50.000
Technican	CS	45.000
Prof	Math	60.000
AssocProf	Math	55.000
Technican	Math	45.000

univ-B		
salInfo		
category	CS	Math
Prof	55.000	65.000
AssocProf	50.000	55.000
Technican	43.000	44.000

Dynamische Schemadefinition: Liste von Werten einer Variable (D) wird zu Spalten  $d_1, d_2, \dots, d_n$  expandiert



# SchemaSQL: Implementation

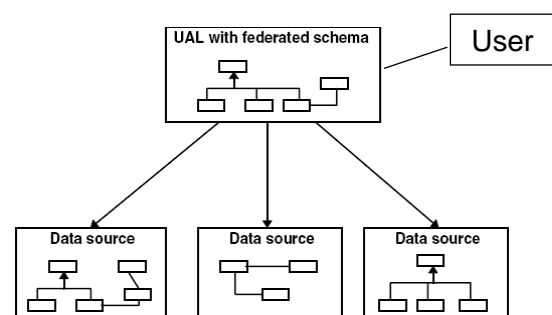
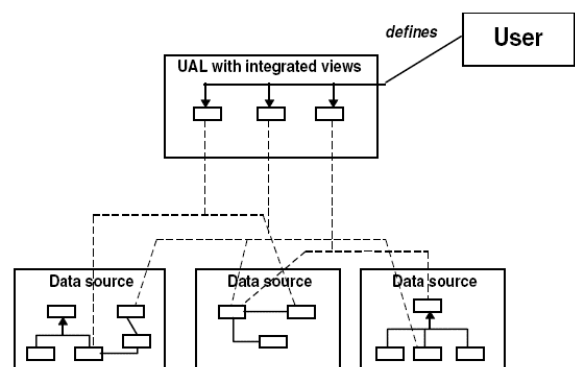
- Ziele
  - *Non-intrusive*
  - Ausnutzung vorhandener RDBMS
  - Übersetzung von Schema-SQL in Sequenz von (verteilten) SQL Befehlen
  - Optimierung
  - Metadatenverwaltung
- Vorgehen (grob)
  - Phase 1: Variablen instanzieren
    - Zugriff auf Metadaten durch entsprechende Systemtabellen, welche das Schema (Metadaten) einer Datenbank generisch mit Datensätzen (Instanzdaten) repräsentiert  
→ vergleiche Erstellung eines generischen globalen Schemas
  - Phase 2: SchemaSQL-Anfrage in (mehrere) SQL-Anfragen umschreiben und auf den instanziierten Variablen ausführen
    - Senden von "normalen" SQL-Anfragen an die beteiligten DBMS

24



## Enge vs. lose Kopplung (Wdh.)

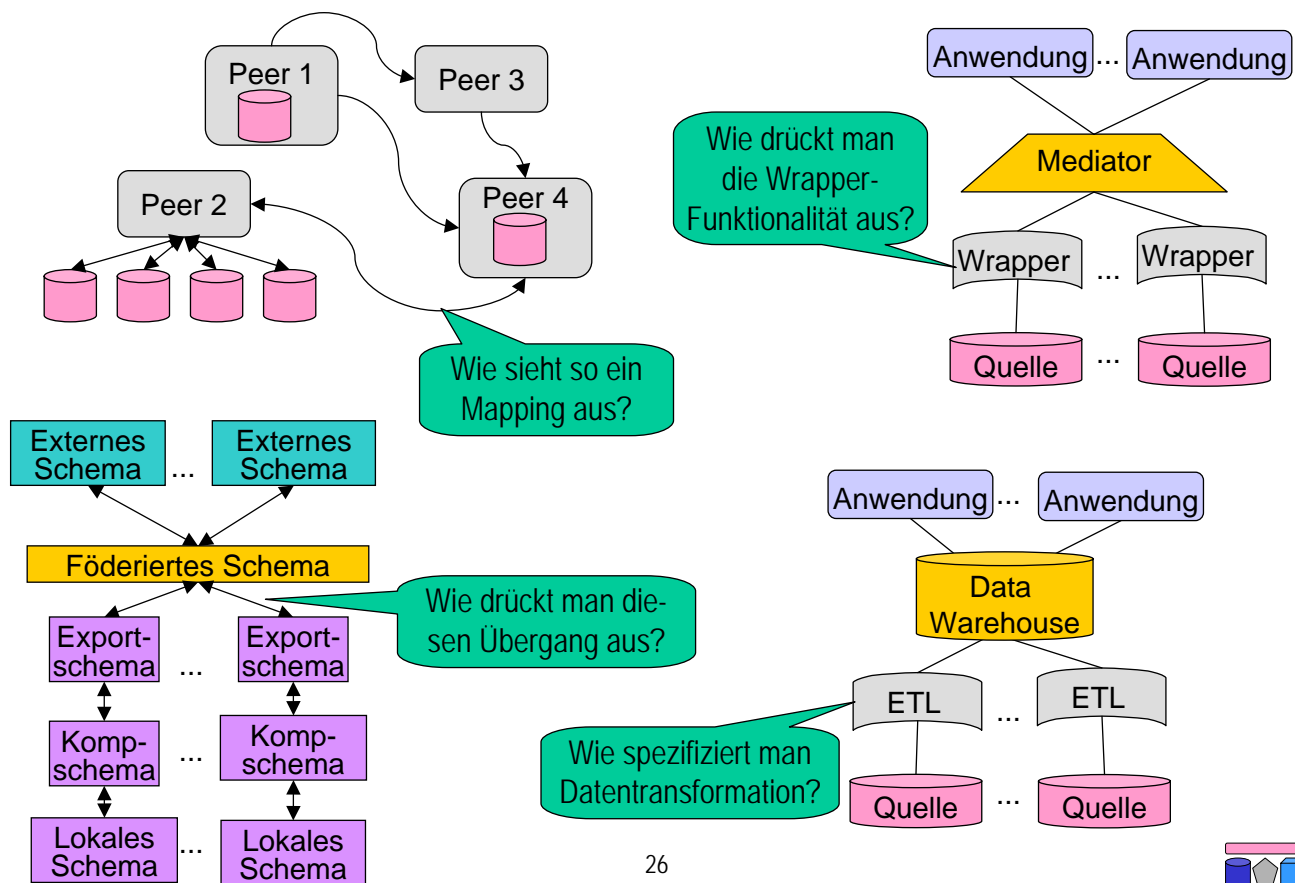
- Lose Kopplung
  - Kein festes Schema
    - Nutzer müssen Semantik der Quellen kennen
    - Integrierte Sichten helfen
  - Multidatenbanksprachen
- Enge Kopplung
  - Globales / integriertes / föderiertes Schema
    - Nutzer stellen Anfrage bzgl. globalem Schema
    - Nutzer müssen Schema / Semantik der Quellen nicht kennen
  - Global-As-View (GaV)
  - Local-As-View (LaV)



25



# Problem



26



## Problem und Teilprobleme

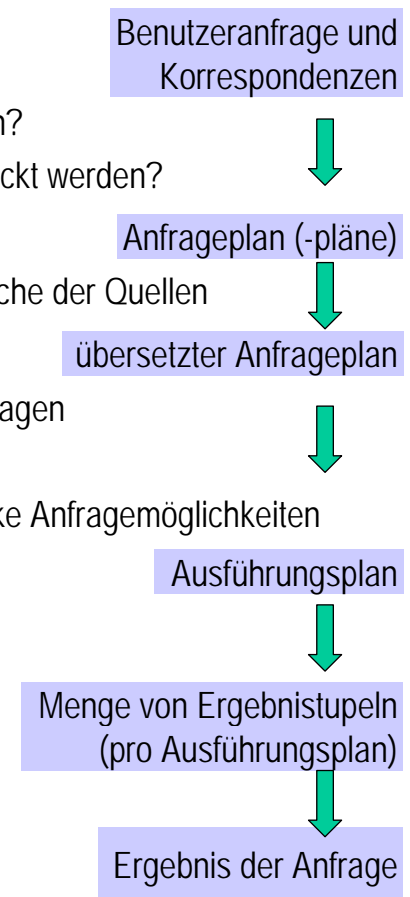
- Gegeben: Zwei heterogene Schemata
  - Quellschema: Hier sind die Daten
  - Zielschema: Hiergegen wird die Anfrage gestellt
- Teilproblem 1: Beziehungen zwischen Schemata finden
  - Spezifikation semantisch äquivalenter Zeile in Quell- und Zielschema
  - Korrespondenzen zwischen Schemaelementen (Mappings)
- Teilproblem 2: Anfrage übersetzen
  - Gegeben Anfrage an Zielschema
  - Finde semantisch äquivalente Sequenz von Anfragen an Quellschemata
  - Entfällt bei materialisierten Ansätzen (z.B. Data Warehouse)
- Teilproblem 3: Daten von der Quelle zum Ziel transformieren
  - Gegebene Ergebnisse der Teilanfragen (oder ganzer Datenbestand)
  - Transformation in das Zielschema
  - Anpassung der Werte an Konventionen des Zielschemas
  - Offline (ETL, DWH) oder Online (Föderation)

27



# Anfragebearbeitung

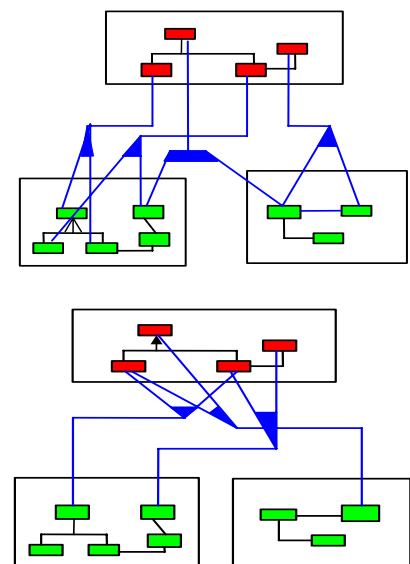
- Schritt 1: Anfrageplanung
  - Welche Quellen können / sollen / müssen benutzt werden?
  - Welche Teile der Anfrage sollen an welche Quelle geschickt werden?
- Schritt 2: Anfrageübersetzung
  - Übersetzung aus der Sprache der Föderation in die Sprache der Quellen
- Schritt 3: Anfrageoptimierung
  - Finde geeignete Reihenfolge der Ausführung der Teilanfragen
  - Wer führt was aus (Pushen)?
  - Wer kann was ausführen (Kompensation)? → Beschränke Anfragemöglichkeiten
- Schritt 4: Anfrageausführung
  - Monitoring, Puffern, Cachen
  - Dynamische Reoptimierung
- Schritt 5: Ergebnisintegration
  - Duplikaterkennung und Konfliktauflösung



28

## Global-as-View vs. Local-as-View

- Modellierung strukturell heterogener Quellen in Bezug auf ein globales Schema als Views (Sichten)
  - Eine Sicht verknüpft mehrere Relationen und produziert eine Relation
- Sichten zur Verknüpfung von Schemata
  - Sicht definiert auf Relationen eines Schemas und produziert eine Relation des anderen Schemas
- Global as View
  - Relationen des globalen Schemas werden als Sichten auf die lokalen Schemas der Quellen ausgedrückt
- Local as View
  - Relationen der Schemas der Quellen werden als Sichten auf das globale Schema ausgedrückt



29

# GaV und LaV: Beispiel

Globales Schema  
Film(Titel, Regie, Jahr, Genre)

Lokale Schemata:  
S1: IMDB(Titel, Regie, Jahr, Genre)  
S2: MyMovies(Titel, Regie, Genre)  
S3a: RegieDB(Titel, Regie)  
S3b: GenreDB(Titel, Jahr, Genre)

## Global as View

```
CREATE VIEW Film AS
SELECT * FROM IMDB
UNION
SELECT Titel, Regie, NULL AS Jahr, Genre
FROM MyMovies
UNION
SELECT RegieDB.Titel, RegieDB.Regie,
       GenreDB.Jahr, GenreDB.Genre
FROM RegieDB, GenreDB
WHERE RegieDB.Titel = GenreDB.Titel
```

Typisches  
GaV-Merkmal

Fehlendes Attribut

Join über  
mehrere Quellen

## Local as View

```
CREATE VIEW S1 AS
SELECT * FROM Film

CREATE VIEW S2 AS
SELECT Film.Titel, Film.Regie, Film.Genre
FROM Film
```

30



# GaV: Globale Nebenbedingung

Globales Schema  
NeuerFilm (Titel, Regie, Jahr, Genre)  
*Nebenbedingung: Jahr > 2000*

Lokale Schemata:  
S1: IMDB (Titel, Regie, Jahr, Genre)  
S2: MyMovies (Titel, Regie, Jahr, Genre)

```
CREATE VIEW NeuerFilm AS
SELECT *
FROM IMDB
WHERE Jahr > 2000
UNION
SELECT * AS Jahr, Genre
FROM MyMovies
WHERE Jahr > 2000
```

Globale Nebenbedingung  
kann modelliert werden

Lokale Schemata:  
S1: AlleFilmeNett (Titel, Regie, Jahr, Genre)  
S2: AlleFilmeBöse (Titel, Regie, Genre)

```
CREATE VIEW NeuerFilm AS
SELECT *
FROM AlleFilmeNett
WHERE Jahr > 2000
```

```
UNION
SELECT Titel, Regie, NULL AS Jahr, Genre
FROM MyMovies
WHERE ____ > 2000
```

S2 kann nicht mit globaler  
Nebenbedingung modelliert werden

32



# GaV: Lokale Nebenbedingung

Globales Schema  
 Film (Titel, Regie, Jahr, Genre)

S1: AlleFilmeNett(Titel, Regie, Jahr, Genre)  
 S2: AlleFilmeBöse(Titel, Regie, Genre)  
 S3: NeueFilmeNett(Titel, Regie, Jahr, Genre)  
*(Nebenbedingung: Jahr > 2000)*  
 S4: NeueFilmeBöse(Titel, Regie, Genre)  
*(Nebenbedingung: Jahr > 2000)*  
 S5: AktuelleFilme(Titel, Regie, Genre)  
*(Nebenbedingung: Jahr = 2008)*

- Eigenschaften
  - Jede Quelle kann modelliert werden
  - Jede Sicht exportiert Daten der Quelle
  - Lokale Nebenbedingung kann nicht immer modelliert werden (S4)

```
CREATE VIEW Film AS
SELECT *
FROM AlleFilmeNett
UNION
SELECT Titel, Regie, NULL, Genre
FROM AlleFilmeBöse
UNION
SELECT *
FROM NeueFilmeNett
WHERE Jahr > 2000
UNION
SELECT Titel, Regie, NULL, Genre
FROM NeueFilmeBöse
UNION
SELECT Titel, Regie, 2004, Genre
FROM AktuelleFilme
```



# GaV: Beitrag zum Anfrageergebnis (Beispiel)

```
CREATE VIEW Film AS

SELECT *
FROM AlleFilmeNett
UNION
SELECT Titel, Regie, NULL, Genre
FROM AlleFilmeBöse
UNION
SELECT * FROM NeueFilmeNett
WHERE Jahr > 2000
UNION
SELECT Titel, Regie, NULL, Genre
FROM NeueFilmeBöse
UNION
SELECT Titel, Regie, 2008, Genre
FROM AktuelleFilme
```

SELECT * FROM Film WHERE		
Jahr > 2001	Jahr = 2001	Jahr < 1950

# GaV: Modellierung

- Fehlende Attribute in Quellschemata durch NULL-Werte ersetzt
  - Darstellung im Endergebnis hängt von Datenfusionsmethode ab
- Nebenbedingungen im globalen Schema können Integration behindern
  - wenn die Bedingung nicht geprüft werden kann (siehe fehlendes Attribut)
- Nebenbedingungen in lokalen Schemata können modelliert werden, wenn
  - sie auf exportierten Attributen gelten
  - sie auf globalen Attributen gelten und die Form "Attribut = Konstante" haben
- Quellen tragen evtl. nicht zum Anfrageergebnis bei, wenn Nebenbedingung nicht modelliert wurde

36



# GaV: Anfrageverarbeitung

- Gegeben:
  - Anfrage an globales Schema, d.h. auf Relationen des globalen Schemas
  - Für jede globale Relation genau eine Sicht auf lokale Quellen
- Gesucht:
  - Alle Tupel, die die Anfragebedingungen erfüllen
  - Aber: Daten sind in lokalen Quellen gespeichert
- Idee: Ersetze jede Relation der Anfrage durch ihre Sicht (Sichtentfaltung, View Expansion, Query Unfolding) → Geschachtelte Anfrage

Globales Schema  
Film(Titel, Regie, Jahr, Genre)

Lokale Schemata:  
S1: IMDB(Titel, Regie, Jahr, Genre)  
S3a: RegieDB(Titel, Regie)  
S3b: GenreDB(Titel, Jahr, Genre)

```
SELECT * FROM Film
SELECT * FROM (
  SELECT * FROM IMDB
  UNION
  SELECT RegieDB.Titel, RegieDB.Regie,
         GenreDB.Jahr, GenreDB.Genre
  FROM RegieDB, GenreDB
  WHERE RegieDB.Titel = GenreDB.Titel ) AS Film
```

GaV Film

# GaV: Anfrageverarbeitung (Beispiel)

Globales Schema  
 Film(Titel, Regie, Jahr, Genre)  
 Programm (Kino, Titel, Zeit)

Lokale Schemata:  
 S1: IMDB(Titel, Regie, Jahr, Genre)  
 S3a: RegieDB(Titel, Regie)  
 S3b: GenreDB(Titel, Jahr, Genre)  
 S7: KinoDB (Kino, Titel, Zeit)

*Anfrage an globales Schema*  
 „Alle Filme mit Regie und Zeit,  
 die nach 20 Uhr laufen“

```
SELECT F.Titel, F.Regie, P.Zeit
FROM Film AS F, Programm AS P
WHERE F.Titel = P.Titel
AND P.Zeit > 20 Uhr
```

```
SELECT F.Titel, F.Regie, P.Zeit
FROM (
```

```
SELECT * FROM IMDB
UNION
SELECT RegieDB.Titel, RegieDB.Regie,
GenreDB.Jahr, GenreDB.Genre
FROM RegieDB, GenreDB
WHERE RegieDB.Titel = GenreDB.Titel
```

```
) AS F, (
```

```
SELECT *
FROM KinoDB
```

```
) AS P
```

```
WHERE F.Titel = P.Titel
AND P.Zeit > 20 Uhr
```



View Expansion

38



# GaV: Anfrageverarbeitung (Ausführung)

- Konzeptionell
  - "Sichten einsetzen" → beliebig tiefe Schachtelungen
  - Ausführung der Anfragen von innen nach außen und Speicherung in temporären Relationen
- Tatsächlich: Optimierungspotential durch Umschreiben der Anfrage
- Entschachtelung
  - Schachtelung wird i.d.R. bei entfernten Quellen belassen
  - Schachtelung wird i.d.R. bei materialisierten Sicht aufgelöst
- Bildung von Teilanfragen in WHERE-Bedingungung
  - Beispiel (skalare Teilanfrage): Relation MovieDB wird nur benutzt, um einen Wert (Id) zu ermitteln; Anfrageergebnis erhält nur Daten aus Relation RegieDB

```
SELECT Regie
FROM RegieDB, MovieDB
WHERE Titel = 'Star Wars'
AND RegieDB.movieID = MovieDB.ID
```



```
SELECT Regie
FROM RegieDB
WHERE movieID = (
SELECT ID
FROM MovieDB
WHERE Titel = 'Star Wars')
```

39





# GaV und LaV: Beispiel

Globales Schema  
Film(Titel, Regie, Jahr, Genre)

Lokale Schemata:

S1: IMDB(Titel, Regie, Jahr, Genre)

S2: MyMovies(Titel, Regie, Genre)

S3a: RegieDB(Titel, Regie)

S3b: GenreDB(Titel, Jahr, Genre)

## Global as View

```
CREATE VIEW Film AS
SELECT * FROM IMDB
UNION
SELECT Titel, Regie, NULL AS Jahr, Genre
FROM MyMovies
UNION
SELECT RegieDB.Titel, RegieDB.Regie,
       GenreDB.Jahr, GenreDB.Genre
FROM RegieDB, GenreDB
WHERE RegieDB.Titel = GenreDB.Titel
```

## Local as View

```
CREATE VIEW S1 AS
SELECT * FROM Film
```

Je ein View pro lokale Relation

```
CREATE VIEW S2 AS
SELECT Film.Titel, Film.Regie, Film.Genre
FROM Film
```

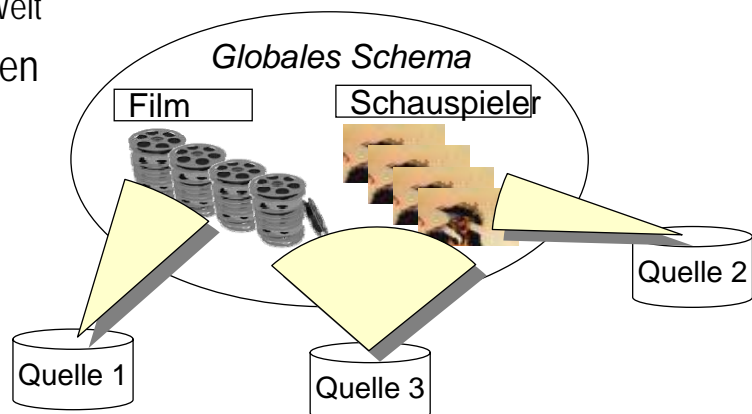
```
CREATE VIEW S3a AS
SELECT Film.Titel, Film.Regie FROM Film
```

```
CREATE VIEW S3b AS
SELECT Film.Titel, Film.Jahr, Film.Genre
FROM Film
```



## Warum LaV?

- Beispiel: Es gibt in der Welt eine Menge von Filmen, Schauspielern, ...
- Das globale Schema modelliert diese Welt
- Theoretisch steht damit die Extension (d.h. alle Datensätze) fest
  - Aber niemand kennt sie
  - Informationsintegration versucht sie herzustellen
- Quellen speichern Sichten auf die globale Extension
  - Also Ausschnitte der realen Welt
- Nur die können wir verwenden



# LaV: Globale Nebenbedingung

Globales Schema  
NeuerFilm (Titel, Regie, Jahr, Genre)  
*Nebenbedingung: Jahr > 2000*

Lokale Schemata:  
S1: IMDB (Titel, Regie, Jahr, Genre)  
S2: MyMovies (Titel, Regie, Jahr, Genre)

Lokale Schemata:  
S1: AlleFilmeNett (Titel, Regie, Jahr, Genre)  
S2: AlleFilmeBöse (Titel, Regie, Genre)

```
CREATE VIEW S1 AS
SELECT * FROM NeuerFilm
WHERE Jahr > 2000
```

```
CREATE VIEW AlleFilmeNett AS
SELECT * FROM NeuerFilm
WHERE Jahr > 2000
```

```
CREATE VIEW S2 AS
SELECT * FROM NeuerFilm
WHERE Jahr > 2000
```

```
CREATE VIEW AlleFilmeBöse AS
SELECT Titel, Regie, Genre
FROM NeuerFilm
WHERE _____ > 2000
```

Globale Nebenbedingung  
kann modelliert werden

S2 kann nicht mit globaler  
Nebenbedingung modelliert werden

42



# LaV: Lokale Nebenbedingung

Globales Schema  
Film (Titel, Regie, Jahr, Genre)

S1: AlleFilmeNett(Titel, Regie, Jahr, Genre)  
S2: AlleFilmeBöse(Titel, Regie, Genre)  
S3: NeueFilmeNett(Titel, Regie, Jahr, Genre)  
*(Nebenbedingung: Jahr > 2000)*  
S4: NeueFilmeBöse(Titel, Regie, Genre)  
*(Nebenbedingung: Jahr > 2000)*  
S5: AktuelleFilme(Titel, Regie, Genre)  
*(Nebenbedingung: Jahr = 2008)*

- Eigenschaften
  - Jede Quelle kann modelliert werden
  - Lokale Nebenbedingung können modelliert werden, dienen jedoch "nur" der Optimierung

```
CREATE VIEW S1 AS
SELECT * FROM Film
```

```
CREATE VIEW S2 AS
SELECT Titel, Regie, Genre
FROM Film
```

```
CREATE VIEW S3 AS
SELECT * FROM Film
(WHERE Jahr > 2000)
```

```
CREATE VIEW S4 AS
SELECT Titel, Regie, Genre
FROM Film
(WHERE Jahr > 2000)
```

```
CREATE VIEW S5 AS
SELECT Titel, Regie, Genre
FROM Film
(WHERE Jahr = 2008)
```

43



# Vergleich: GaV vs. LaV

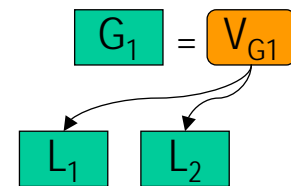
	Global-as-View	Local-as-View
globale Relation		
lokale Relation		
Anzahl der Sichten		
Typischer Aufbau der Sichtdefinition		
Nebenbedingungen	global: lokal:	global: lokal:
Anfrage-umschreibung	<ul style="list-style-type: none"> <li>• View Unfolding</li> <li>• 1 große umgeschriebene Anfrage</li> <li>• schwierige Optimierung</li> </ul>	<ul style="list-style-type: none"> <li>• Answering Queries using Views</li> <li>• UNION vieler umgeschriebene Anfragen</li> <li>• sehr schwieriges Problem</li> </ul>
Flexibilität		
Hinzufügen neuer Quellen		
Geeignet für		

## Global-Local-as-View (GLaV)

- Kombination GaV und LaV, auch Both-as-View (BaV)

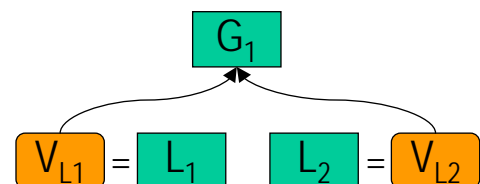
- GaV:

- Globale Relation = Sicht auf lokale Relationen
- (Globale Relation  $\supseteq$  Sicht auf lokale Relationen)



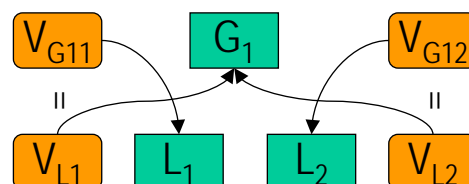
- LaV:

- Sicht auf globale Relationen = lokale Relation
- Sicht auf globale Relationen  $\supseteq$  lokale Relation



- GLaV:

- Sicht auf globale Relationen = Sicht auf lokale Relationen
- Sicht auf globale Relationen  $\supseteq$  Sicht auf lokale Relationen



# Vergleich: GaV, LaV und GLaV

	GaV	LaV	GLaV
Nebenbedingungen	Probleme bei lokalen NB	Probleme bei globalen NB	lokale und globale NB können modelliert werden
Spezialisierung von Konzepten	Konzept im globalen Schema spezieller als im lokalen Schema (ohne dass man eine Einschränkung im lokalen Schema angeben kann) → keine Integration	Konzept im lokalen Schema spezieller als im globalen Schema (ohne dass man eine Einschränkung im globalen Schema angeben kann) → keine Integration	Konzepte können lokal oder global eingeschränkt werden
Techniken der Anfrageplanung	View Unfolding	Query Containment und Answering Queries using Views (gleich)	erst wie bei LaV, dann Unfolding wie bei GaV

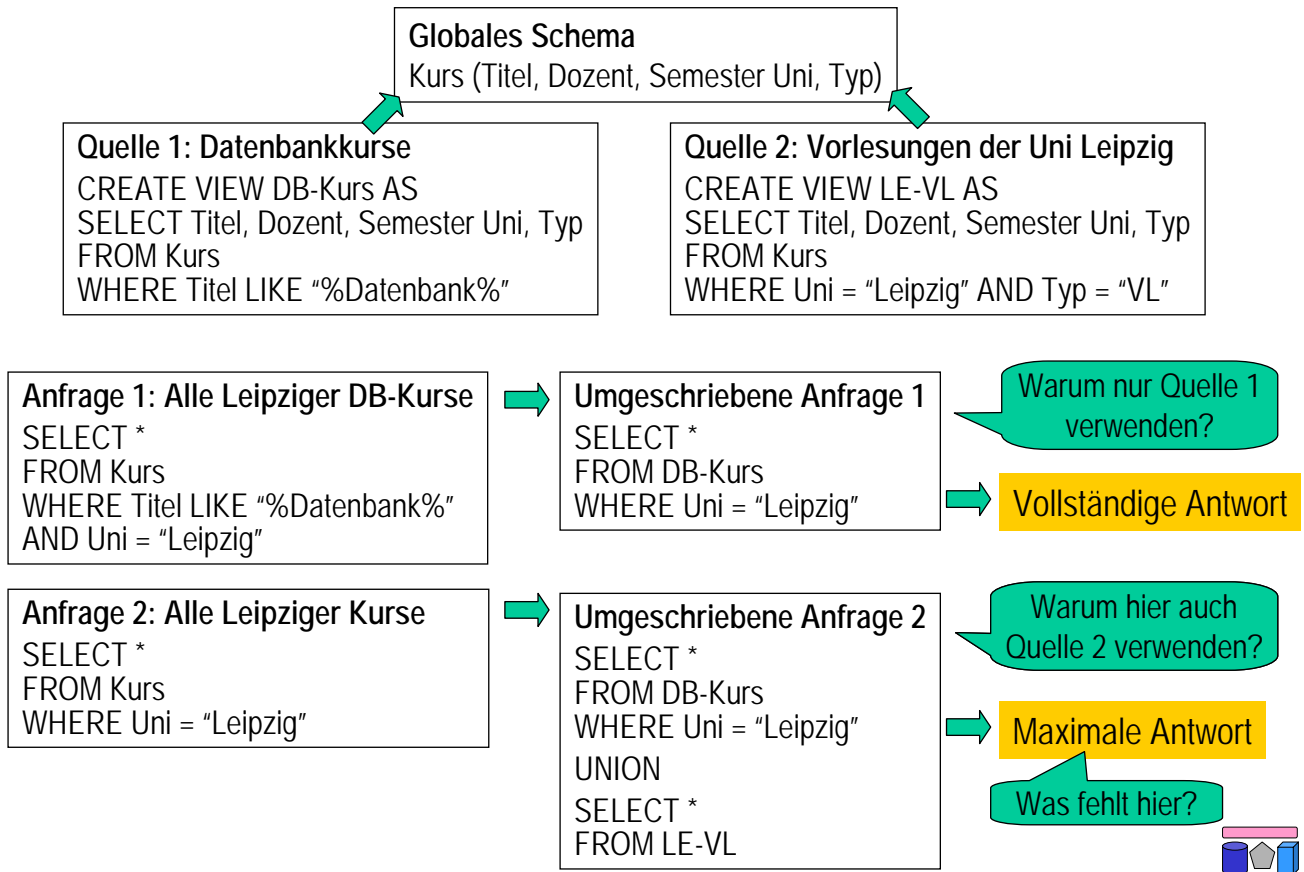


## LaV-Anfrageverarbeitung

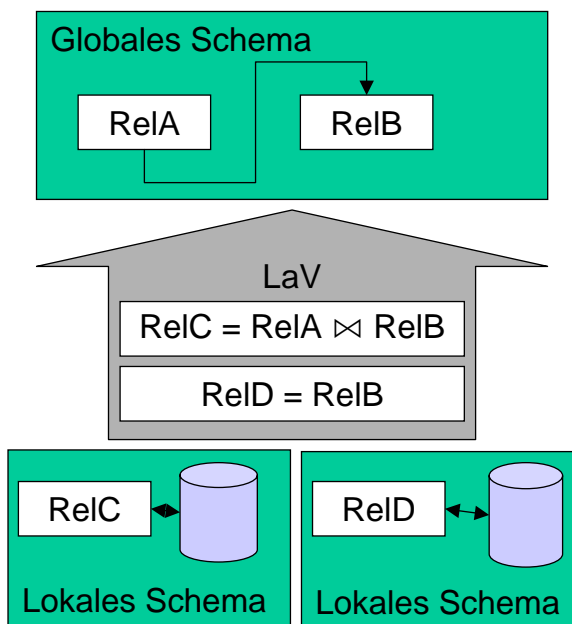
- Gegeben:
  - Anfrage an globales Schema, d.h. auf Relationen des globalen Schemas
  - Für jede lokale Relation genau eine Sicht auf globales Schema
- Gesucht:
  - Alle Tupel, die die Anfragebedingungen erfüllen
  - Aber: Daten sind in lokalen Quellen gespeichert.
- Idee: Answering Queries using Views
  - Anfrageumschreibung durch Einbeziehung der Sichten
  - Kombiniere geschickt die einzelnen Sichten zu einer Anfrage, so dass deren Ergebnis einen Teil der Anfrage (oder die ganze Anfrage) beantworten
  - Gesamtergebnis ist dann die UNION der Ergebnisse mehrerer Anfrageumschreibungen



# LaV-Anfrageverarbeitung: Beispiel



# LaV-Anfrageverarbeitung: Visualisierung



Nutzer-Anfrage

```
SELECT ???  
FROM RelB  
WHERE ???
```



Umgeschriebene Anfrage

```
SELECT ???  
FROM RelD  
WHERE ???  
UNION  
SELECT Attr (B)  
FROM RelC  
WHERE ???
```

Nutzer-Anfrage

```
SELECT ???  
FROM RelA, RelB  
WHERE ???
```



Umgeschriebene Anfrage

```
SELECT ???  
FROM RelC  
WHERE ???
```



# Ein Problem – viele Anwendungen

- Anfrageoptimierung unter Verwendung materialisierter Sichten
  - Welche Sichten helfen bei der Beantwortung einer Datenbankanfrage durch Vorberechnung von Prädikaten?
  - Nicht immer besser eine materialisierte Sicht zu verwenden (Indizes, Aktualisierung)
- Adaptive Erstellung materialisierter Sichten für ein Data Warehouse
  - Gegeben: Query workload = Menge von Anfragen mit Häufigkeiten
  - Welche Sichten materialisieren, um Query Workload optimal zu beantworten
- Semantisches Caching
  - Welche Daten im Cache können zur Beantwortung einer Anfrage verwendet werden?
  - Welche Daten müssen neu angefragt werden?
- Datenintegration: Beantwortung von Fragen an globales Schema ausschließlich unter Verwendung von Sichten auf lokale Quellen
  - Kann die Anfrage vollständig (extensional) beantwortet werden?
  - Unterschied zu Anfrageoptimierung: Keine Basistabellen verfügbar

51



## Korrespondenzen

- Anfragen werden bzgl. ihrer Extension (=Menge der Ergebnistupel) miteinander verglichen
  - Überlappungsfrei:  $q_1 \cap q_2 = \emptyset$
  - Inklusion:  $q_1 \supseteq q_2$
  - Äquivalenz:  $q_1 \equiv q_2$
  - Überlappung:  $\neg (q_1 \cap q_2 = \emptyset) \wedge \neg (q_1 \supseteq q_2) \wedge \neg (q_2 \supseteq q_1)$
- GaV und LaV-Modellierung durch Sichtendefinition mittels Anfragen
  - Beziehungen (Korrespondenzen) zwischen Anfragen  $q$
- Eine Korrespondenz  $q_1 \supseteq q_2$  heißt
  - GaV (Global-as-View), wenn  $q_1$  (globales Schema) eine einzelne Relation ohne Selektionen oder Joins ist (Typ: Relation  $\supseteq$  Anfrage)
  - LaV (Local-as-View), wenn  $q_2$  (lokales Schema) eine einzelne Relation ohne Selektionen oder Joins ist (Typ: Anfrage  $\supseteq$  Relation)
  - Sonst heißt sie GLaV (Global-local-as-view) (Typ: Anfrage  $\supseteq$  Anfrage)

52



# Query Containment

- Gegeben: Anfrage  $Q$  und Sichten  $V_1, \dots, V_n$
- Gesucht: Umgeschriebene Anfrage  $Q'$ , die
  - bei Optimierung unter Verwendung materialisierter Views: äquivalent ist ( $Q = Q'$ ).
  - bei Integration: maximal enthalten ist.
    - D.h.  $Q \supseteq Q'$  und es existiert kein  $Q''$  mit  $Q \supseteq Q'' \supseteq Q'$  wobei  $Q'' \neq Q'$ .
- Problem: Wie definiert und testet man Äquivalenz bzw. maximal containment?



## Query Containment (2)

- Query containment (Anfrage-“Enthaltensein”)
  - Sei  $S$  ein Schema. Seien  $Q$  und  $Q'$  Anfragen gegen  $S$ .
  - Eine Instanz von  $S$  ist eine beliebige Datenbank  $D$  mit Schema  $S$ .
  - Das Ergebnis einer Anfrage  $Q$  gegen  $S$  auf einer Datenbank  $D$ , geschrieben  $Q(D)$ , ist die Menge aller Tupel, die die Ausführung von  $Q$  in  $D$  ergibt.
  - $Q'$  ist contained (enthalten) in  $Q$ , geschrieben  $Q' \subseteq Q$ , gdw.  $Q'(D) \subseteq Q(D)$  für jedes mögliche  $D$ .
- Query equivalence (Anfrageäquivalenz)
  - $Q$  ist äquivalent mit  $Q'$ , geschrieben  $Q = Q'$ , gdw.  $Q(D) \subseteq Q'(D)$  und  $Q'(D) \subseteq Q(D)$  für jede mögliche Datenbank  $D$ .
- Es zählt das Ergebnis einer Anfrage, nicht deren Syntax!



# Query Containment: Beispiele

```
SELECT Titel, Dozent, Semester Uni, Typ
FROM Kurs
WHERE Titel LIKE "%Datenbank%"
AND Uni = "LE"
```

```
SELECT Titel, Dozent, Semester Uni, Typ
FROM Kurs
WHERE Titel LIKE "%Datenbank%"
```

```
SELECT Titel, Dozent, Semester Uni, Typ
FROM Kurs JOIN Person
ON (Kurs.Dozent = Person.Name)
WHERE Titel LIKE "%Datenbank%"
```

```
SELECT Titel, Dozent, Semester Uni, Typ
FROM Kurs
WHERE Titel LIKE "%Datenbank%"
```

```
SELECT Titel, Dozent
FROM Kurs
WHERE Titel LIKE "%Datenbank%"
```

```
SELECT Titel, Uni, Typ
FROM Kurs
WHERE Titel LIKE "%Datenbank%"
```

```
SELECT Titel, Dozent, Semester Uni
FROM Kurs
WHERE Typ IN ("VL", "Praktikum")
```

```
SELECT Titel, Dozent, Semester Uni
FROM Kurs
WHERE Typ IN ("Praktikum", "Seminar")
```

55



## Prüfung auf Query Containment

- Prüfung von Containment durch Prüfung aller möglichen Datenbanken?
  - Definition: "... für jedes mögliche D" → zu komplex!
- Prüfung von Containment durch Existenz eines Containment Mapping.
  - NP-vollständig in  $|Q|+|Q'|$  nach  
*Chandra & Merlin. Optimal implementation of conjunctive queries in relational data bases. ACM Symposium on Theory of Computing, 1977.*
  - Mehrere Algorithmen
- Syntaktische Analyse (der Anfrage / View-Definition), um semantische Aussage ("Stets alle Tupel von Q in Q' enthalten?") zu treffen
  - Containment Mapping = Symbolmapping
  - CM von  $q_2$  nach  $q_1$  bedeutet, dass  $q_1 \subseteq q_2$  durch
  - Idee (intuitiv): Nachweis, dass  $q_1$ 
    - nicht mehr Tupel berechnet (mehr Joins, strengere Bedingungen)
    - nicht weniger Spalten berechnet (evtl. mehr durch andere Projektion)

57





# Anfrageplanung mittels Query Containment

- Basis: Algorithmus zum Containment zweier Queries
- Anfrageplanung "light"
  - Globale Anfrage  $q$ , für deren Beantwortung die Verwendung einer LaV-Korrespondenz ausreicht (ohne JOIN)
  - Menge von Korrespondenzen  $K$  (LaV-Korrespondenzen)
  - Deren globaler Anfrageteil ist eine Menge von Views  $V$
  - Query-Containment-Test für jedes  $v \in V$ : Gilt  $v \subseteq q$ ?
  - Die  $v$ , für die das zutrifft, berechnen nur korrekte Tupel für  $q$
  - Ihre Vereinigung ist das (bestmögliche) Ergebnis
- Anfrageplanung "real-world"
  - Einbeziehung mehrerer Korrespondenzen
  - Answering Queries using Views: Antworten durch Kombination von Views
  - Welche Views sollen wie miteinander verknüpft werden?

58



## Korrespondenzen: LaV-Beispiel

- Beispiel: Filmdatenbank
  - globales und lokale Schema
  - LaV-Regeln in Datalog-Notation

### Globales Schema

film (titel, typ, regisseur, laenge)  
 schauspieler (schauspieler\_name, nationalitaet)  
 spielt (titel, schauspieler\_name, rolle, kritik)

Datenquelle	Beschreibung
spielfilme (titel, regisseur, laenge)	Spielfilme mit mind. 80min Länge
kurzfilme (titel, regisseur)	Kurzfilme; sind max. 10min lang
filmkritiken (titel, regisseur, schauspieler, kritik)	Kritiken zu Hauptdarstellern von Filmen
us_spielfilme (titel, laenge, schauspieler_name)	Spielfilme mit US-Schauspielern
spielfilm_kritiken (titel, rolle, kritik)	Kritiken zu Rollen in Spielfilmen
kurzfilm_rollen (titel, rolle, schauspieler_name, nationalitaet)	Rollenbesetzungen in Kurzfilmen

film (T,Y,R,L), $L > 79$ , $Y = \text{'Spielfilm'}$	$\supseteq$	spielfilme (T,R,L)
film (T,Y,R,L), $L < 11$ , $Y = \text{'Kurzfilm'}$	$\supseteq$	kurzfilme (T,R)
film (T,_,R,_), spielt (T,S,O,K), $O = \text{'Hauptrolle'}$	$\supseteq$	filmkritiken (T,R,S,K)
film (T,Y,_,L), spielt (T,S,_,_), schauspieler (S,N), $N = \text{'US'}$ , $Y = \text{'Spielfilm'}$	$\supseteq$	us_spielfilme (T,L,S)
film (T,Y,_,_), spielt (T,_,O,K), $Y = \text{'Spielfilm'}$	$\supseteq$	spielfilm_kritiken (T,O,K)
film (T,Y,_,_), spielt (T,S,O,_), schauspieler (S,N), $Y = \text{'Kurzfilm'}$	$\supseteq$	kurzfilm_rollen (T,O,S,N)

SELECT titel, regisseur, laenge  
 FROM film  
 WHERE laenge > 79  
 AND typ = 'Spielfilm'

SELECT f.titel, f.regisseur, s.schauspieler\_name, s.kritik  
 FROM film f JOIN spielt s ON (f.titel = s.titel)  
 WHERE s.nationalitaet = 'US'

# LaV: Anfragen

- Alle Filme kürzer als 100 Minuten
  - spielfilme (filtern) und kurzfilme (direkt) und us\_spielfilm (filtern)
- Alle Filme, die länger als 60 Minuten sind
  - spielfilme (filtern), us\_spielfilm (filtern)
- Schauspieler von Hauptrollen in Filmen unter 100 Minuten
  - spielfilme  $\bowtie$  filmkritiken ? Ok, da nur Hauptrollen kritisiert werden
  - spielfilme  $\bowtie$  spielfilm\_kritiken ? Ok, aber ohne Schauspielernamen
  - spielfilme  $\bowtie$  us\_spielfilm ? Ok, aber Rolle fehlt
  - spielfilme  $\bowtie$  kurzfilm\_rollen ? Nutzlos, da JOIN leere Menge ergibt

```
SELECT titel, 'Spielfilm', regisseur, laenge
FROM spielfilm
WHERE laenge < 100

UNION

SELECT titel, 'Kurzfilm', regisseur, NULL
FROM kurzfilm

UNION

SELECT titel, 'Spielfilm', NULL, laenge
FROM us_spielfilme
WHERE laenge < 100
```

film (T,Y,R,L), L>79, Y='Spielfilm'	$\supseteq$	spielfilme (T,R,L)
film (T,Y,R,L), L<11, Y='Kurzfilm'	$\supseteq$	kurzfilme (T,R)
film (T,_,R,_), spielt (T,S,O,K), O='Hauptrolle'	$\supseteq$	filmkritiken (T,R,S,K)
film (T,Y,_,L), spielt (T,S,_,_), schauspieler (S,N), N='US', Y='Spielfilm'	$\supseteq$	us_spielfilme (T,L,S)
film (T,Y,_,_), spielt (T,_,O,K), Y='Spielfilm'	$\supseteq$	spielfilm_kritiken (T,O,K)
film (T,Y,_,_), spielt (T,S,O,_), schauspieler (S,N), Y='Kurzfilm'	$\supseteq$	kurzfilm_rollen (T,O,S,N)



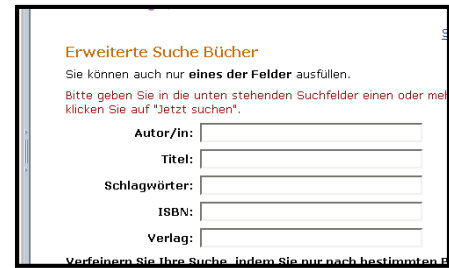
# LaV: Anfrageumschreibung

- Prinzipiell: Prüfe jede Kombination an Sichten auf Containment
  - Unendlich viele Kombinationen, da eine Sicht auch mehrfach in eine Umschreibung eingehen kann.
- Verbesserung: Umschreibung mit maximal so vielen Sichten wie Relationen in Anfrage (ist ausreichend)
  - Beweis: *Levy, Mendelzon, Sagiv, Srivastava: Answering Queries Using Views. PODS 1995*
  - Geschickte Vorauswahl der Sichten: Nutzbarkeit
- Frage 1: Wann sind Sichten nutzbar? (informell)
  - Mindestens eine Relation mit Anfrage gemeinsam
  - Mindestens einige Attribute der Anfrage
  - Prädikate sind schwächer oder gleich (äquivalente Umschreibung)
  - Prädikate sind stärker (contained Umschreibung)
- Frage 2: Wann sind Sichten nützlich?
  - Bei Optimierung mit MVs: Schnellere Ausführung
  - Bei Integration mit LaV: zusätzliche Tupel, zusätzliche Attribute



# Beschränkte Quellen

- Quellen, die nicht alle Anfragen an ihr Exportschema beantworten können
  - Webquellen hinter Formularen
  - Web Services, CORBA, RPC, ...
  - Legacy Systeme
- Quellen verlangen bestimmte Bedingungen
  - `SELECT * FROM BOOK ...` geht nicht
  - `SELECT * FROM BOOK WHERE AUTHOR like ...` geht
  - `SELECT * FROM BOOK WHERE TITLE like ...` geht
- Variablen und Bindungen. Ein Attribut ...
  - ... heißt frei (f), wenn es in einer Anfrage nicht belegt sein muss (aber sein kann)
  - ... heißt gebunden (b), wenn es in einer Anfrage belegt sein muss
  - ... heißt unbindbar (n), wenn es in einer Anfrage nicht belegt sein darf
- Belegung = Selektion der Art „V=c“ oder „V in [c1,...cn]“
  - Verallgemeinerung auf andere Prädikate möglich (siehe Literatur)



62



# Auswirkungen

- Manche Anfragepläne sind nicht mehr ausführbar (`SELECT * FROM BOOK`)
- Pläne sind nicht mehr in jeder Reihenfolge ausführbar

## Beispiel (rechts)

- Alle Titel von Filmen, bei denen ein 90jähriger Schauspieler mitspielt
- Anfrage: `q(T) :- order(O,T), info(O,_,A,_), A=90`

```
order (OID,Title)
• keine Einschränkung
info (OID,Name,Age,Role)
• Suche nach OID oder Name
```

- Plan A: "erst `info(O,_,A,_), A=90`, dann `order`"
  - Gut wegen hoher Selektivität der Bedingung
  - Aber: nicht ausführbar
- Plan B: "erst `order`, dann `info`, dann `A=90`"
  - Schlecht wegen sehr teurem `SELECT * FROM ORDER`, aber ausführbar
  - Dann Bindungen von `OID` "pushen"
  - Bedingung `A=90` erst im Mediator auswerten

63



# Binding Patterns

- Binding Pattern (BP) = Die Menge der Einschränkungen pro Attribut einer Anfrage

`order (OID, Title)`

- Suche nur nach Title

`info (OID, Name, Age, Role)`

- Suche nach OID oder Name  
(nach genau einem)

- Beispiel (rechts): `order (On, Tb)`

- Ist `info (Of, Nf, An, Rn)` korrekt?

- Nein, hier auch Anfrage ohne Bindung und Anfrage nach OID und Name möglich

- Darstellung durch zwei (i.A. mehrere) Binding Patterns

64



## Anfrageplanung mit Binding Patterns

- BP müssen während der Planung zunächst nicht beachtet werden
  - Anfrage  $q \rightarrow$  korrekter Anfrageplan  $p$
- Für eine gegebene Ausführungsreihenfolge von  $p$  kann man nun testen, ob sie ausführbar ist
  - Kostenbasierte Optimierung wählt unter allen ausführbaren Reihenfolgen die beste

- Beispiel (rechts)

- `order` nur Titelsuche, `info` wie vorher

`order (On, Tb)`

`info1 (On, Nb, An, Rn)`

`info2 (Ob, Nn, An, Rn)`

- Algorithmus: Initialisierung

- Vektor  $k$  mit einer Position für jedes Attribut in  $p$
- Initialwert: Bindet  $q$  eine Variable  $v$ , dann "b", sonst "f"

Anfrage:  $q(T, N) :- order(O, T), info(O, N, A, _), T = \text{"Marnie"}$

Vektor:  $k [O, T, N, A, R] =$




# Anfrageplanung mit Binding Patterns (2)

- Analyse jedes Plans  $p = \langle q_1, \dots, q_n \rangle$
- Jedes  $q_i$  hat ein Binding Pattern  $b_i$
- FOR  $i=1$  TO  $n$  DO  $k_{neu} = k_{alt} \diamond b_i$

Plan A: order ( $O^n, T^b$ ), info1 ( $O^n, N^b, A^n, R^n$ )  
 Plan B: order ( $O^n, T^b$ ), info2 ( $O^b, N^n, A^n, R^n$ )

- Berechnung der Verknüpfung  $\diamond$ 
  - Sei  $x=k_{alt}[j]$  und  $y=b_i[j]$
  - $j$  referenziert in beiden Vektoren die gleiche Variable
  - IF  $x=„b“$  und  $y=„n“$   $\rightarrow$  Plan nicht ausführbar, da Bindung nicht weitergegeben werden kann
  - IF  $x=„f“$  und  $y=„b“$   $\rightarrow$  Plan nicht ausführbar, da notwendige Bindung nicht vorhanden ist
  - $k_{neu}[j]=b$ , wenn Variable zu  $j$  in  $q_i$  exportiert wird, sonst  $k'[j]=k[j]$

Init:  $k = fbfff$   
 Plan A1: order, info1  
 •  $fbfff \diamond nb\_ = bbfff$   
 •  $bbfff \diamond n\_bnn \not\rightarrow (b \rightarrow n)$   
 Plan A2: info1, order  
 •  $fbfff \diamond n\_bnn \not\rightarrow (f \rightarrow b)$   
 Plan B1: order, info2  
  
 Plan B2: info2, order



# Post-Processing

- Mediator kann nicht ausführbare Pläne durch Nachverarbeitung "simulieren"
- Übergang:  $b \rightarrow n$ 
  - Bindung kann nicht an Quelle weitergegeben werden
- Workaround:
  - Führe eine Anfrage ohne Bindung aus
  - Falls im Plan schon eine Bindung entstanden ist, teste diese im Mediator während eines Post-Processings
- Auswirkung: „n“ kann wie „f“ behandelt werden
  - Es entstehen aber teure Pläne ohne Pushen von Selektionen

info3 ( $O^n, N^f, A^f, R^f$ )  
 Plan C1: order, info3  
 •  $fbfff \diamond nb\_ = bbfff$   
 •  $bbfff \diamond n\_fff \not\rightarrow (b \rightarrow n)$

info3 ( $O^n, N^f, A^f, R^f$ )  
 Plan C1: order, info3  
 •  $fbfff \diamond nb\_ = bbfff$   
 •  $bbfff \diamond \mathbf{f\_}fff = bbfff$

- Suche mit Titel in order
- Hole alle info-Tupel (teuer)
- Join (über O) in Mediator



# Erweiterungen / Weitere BP-Anwendungen

- Anhänge (adornments = Verzierungen)
  - f: free = Kann in Anfrage spezifiziert werden, muss aber nicht
  - n: unspecifiable = Kann nicht spezifiziert werden
  - b: bound = Muss spezifiziert werden
  - c[s]: constant = Auswahl aus einer Menge s von Konstanten (free)
  - o[s]: optional = Auswahl aus einer Menge s von Konstanten (bound)
  - Behandlung
    - Im Prinzip: „c“ wie „b“ und „o“ wie „f“
    - Weitere Einschränkungen über die Wertemengen bei „c“ und „s“ möglich
- Berechnung von BP einer globalen Relation aus BP der Quellen
  - Wenn ein Mediator beschränkte Quellen integriert – welchen Beschränkungen unterliegt er dann selber?
  - Erstellung eines Anfrageformulars im Mediator aus denen der Quellen



## Zusammenfassung

- Multidatenbanksprachen
  - Lose Kopplung: Nutzer integriert durch Anfragen die mehrere Quellen involvieren
  - Verwendung u.a. zur Erstellung integrierter Sichten
  - SchemaSQL: Syntax, horizontale Aggregation und dynamische Umstrukturierung
- Global-as-View (GaV) und Local-as-View (LaV)
  - Enge Kopplung: Anfragebearbeitung durch System auf Basis von Korrespondenzen
  - Automatische Erstellung von Korrespondenzen → Kap. 6
  - Vergleich GaV- und LaV-Modellierung sowie GLaV
- LaV-Anfragebearbeitung
  - Erstellung eines Anfrageplans
    - Query Containment, Answering Queries using Views
  - Erstellung eines Ausführungsplans
    - Binding Patterns für Quellen mit beschränkten Anfragemöglichkeiten
  - Integration der Ergebnisse → Kap. 7

