Anfragebearbeitung

Komplexe Informationssysteme

Fabian Panse

panse@informatik.uni-hamburg.de

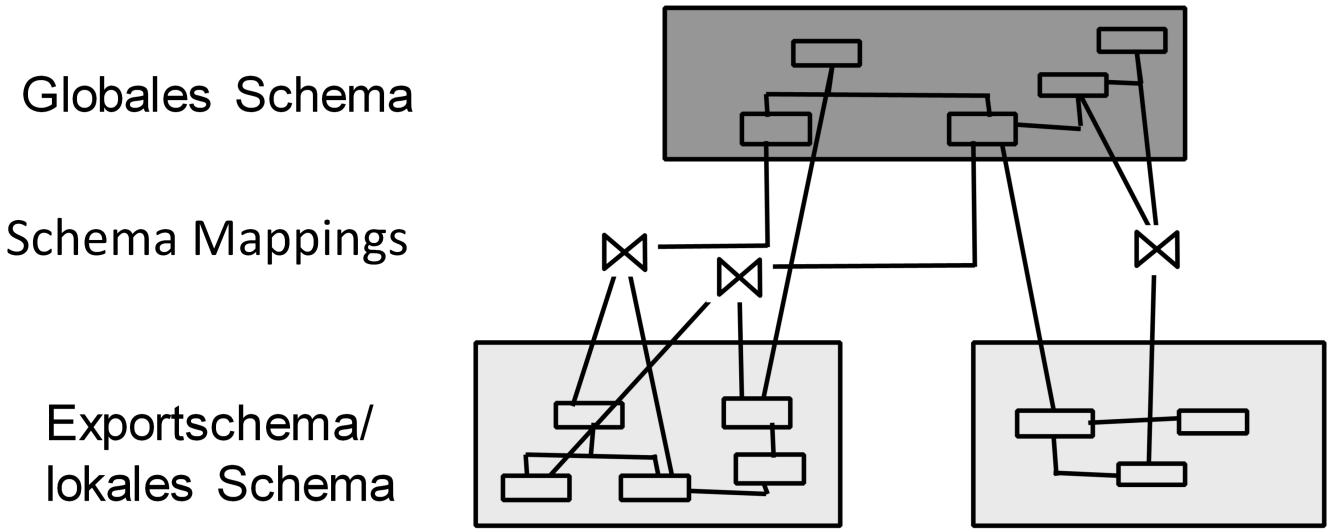
Universität Hamburg

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Anfragebearbeitung in Integrations Systemen

Aufgabe



- Aufgabe der Anfragebearbeitung
  - Gegeben eine Anfrage q gegen das globale Schema
  - Gegeben eine Menge von Mappings zwischen globalem und lokalen Schemata
  - Finde alle Antworten auf q

Anfragebearbeitung

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

Beispiel aus der Filmwelt

Vereinfachende Annahmen

- Quellen bestehen nur aus einer (Export-)Relation.
- Keine Datenmodellheterogenität
  - Nehmen wir jetzt immer an
  - Aufgabe des Wrappers
- Zunächst vereinfachend: Keine strukturelle oder schematische Heterogenität
- Keine semantische Heterogenität
  - Gleiche Namen = gleiche Konzepte
  - Nur offensichtliche Schreibvarianten (Sprachunterschiede)

Anfragebearbeitung (Beispiel)

- Globales Schema

  $film(titel, regisseur)$
  $schauspieler(schauspieler\_name, nationalitaet)$
  $spielt(titel, schauspieler\_name, rolle)$

- Datenquellen:

Datenquelle & exportierte Relation	Beschreibung	Globale Relation
`imdb.movie (titel, director)`	Alle Filme der IMDB	`film`
`imdb.acts(title, actor_name, role)`	Zuordnung von Schauspielern zu Filmen der IMDB	`spielt`
`imdb.actor (actor_name, nationality)`	Alle Schauspieler der IMDB	`schauspieler`
`fd.film(titel, regisseur)`	Alle Filme des Filmdienstes	`film`
`fd.spielt (titel, schauspieler_name, rolle)`	Zuordnung von Schauspielern zu Filmen des Filmdienstes	`spielt`
`fd.schauspieler(schauspieler_name, nationalitaet)`	Alle Schauspieler des Filmdienstes	`schauspieler`

Anfragebearbeitung (Beispiel)

- Globales Schema

  film(*titel*, *regisseur*)
  schauspieler(*schauspieler_name*, *nationalitaet*)
  spielt(*titel*, *schauspieler_name*, *rolle*)

- Datenquellen:

Datenquelle & exportierte Relation	Beschreibung	Globale Relation
`imdb.movie (titel,`		`film`
`imdb.acts(title, ac`	auspielern	`spielt`
`imdb.actor (actor_n`	er IMDB	`schauspieler`
`fd.film(titel, regisseur)`	Alle Filme des Filmdienstes	`film`
`fd.spielt (titel, schauspieler_name, rolle)`	Zuordnung von Schauspielern zu Filmen des Filmdienstes	`spielt`
`fd.schauspieler(schauspieler_name, nationalitaet)`	Alle Schauspieler des Filmdienstes	`schauspieler`

Annahme in diesem Beispiel: Jede Quellrelation wird als eigene Quelle betrachtet

Schritt 1: Anfrageplanung

- Globale Anfrage

  „Alle Filme, in denen Hans Albers eine Hauptrolle spielt“

`SELECT`	*titel*, *regisseur*, *rolle*
`FROM`	*film*, *spielt*
`WHERE`	*spielt.schauspieler_name* = 'Hans Albers'
`AND`	*spielt.rolle* = 'Hauptrolle'
`AND`	*spielt.titel* = *film.titel*

- Welche Quellen kommen infrage?
  - *film*: Quellen *imdb.movie* und *fd.film*
  - *spielt*: Quellen *imdb.acts* und *fd.spielt*

- Ergibt vier Anfragepläne

Schritt 1: Anfrageplanung

Plan	Anfrage
p_1	SELECT title, director, role FROM imdb.movie, imdb.acts WHERE imdb.acts.actor_name = 'Hans Albers' AND imdb.acts.role = 'main actor' AND imdb.acts.title = imdb.movie.title;
p_2	SELECT title, director, rolle FROM imdb.movie, fd.spielt WHERE fd.spielt.schauspieler_name = 'Hans Albers' AND fd.spielt.rolle = 'Hauptrolle' AND fd.spielt.titel = imdb.movie.title;
p_3	SELECT titel, regisseur, role FROM fd.film, imdb.acts WHERE imdb.acts.actor_name = 'Hans Albers' AND imdb.acts.role = 'main actor' AND fd.film.titel = imdb.acts.title;
p_4	SELECT titel, regisseur, rolle FROM fd.film, fd.spielt WHERE fd.spielt.schauspieler_name = 'Hans Albers' AND fd.spielt.rolle = 'Hauptrolle' AND fd.film.titel = fd.spielt.titel;

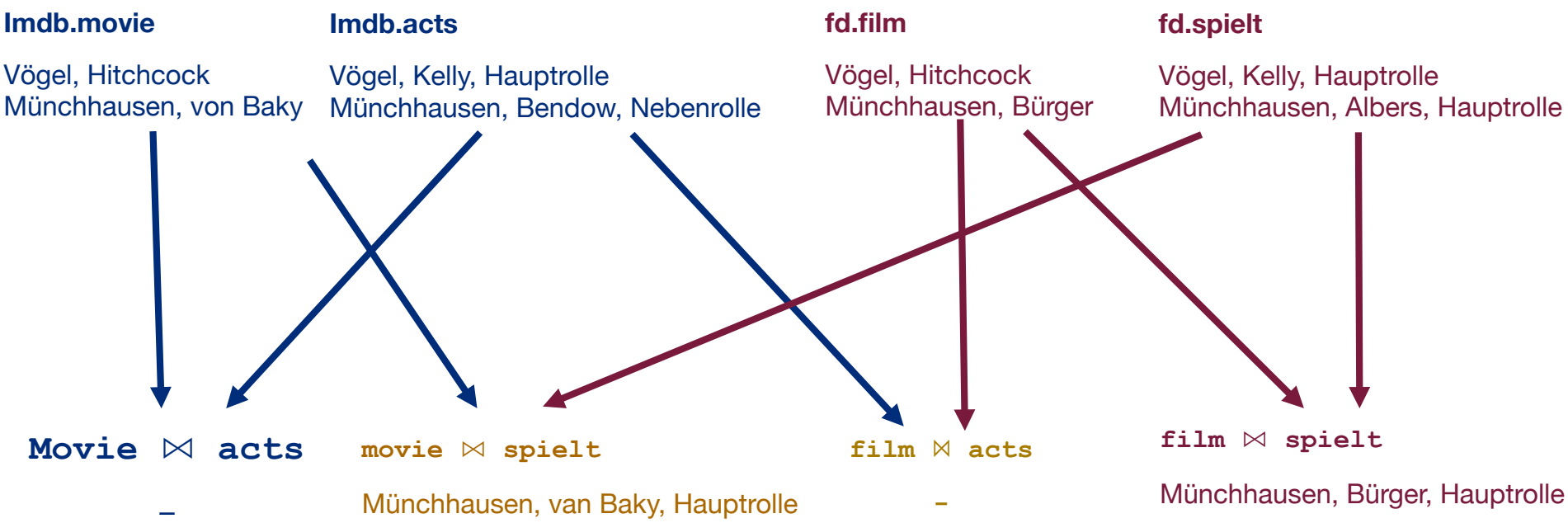
- Jeder Plan besteht aus zwei Teilplänen
  - Auch p_1 und p_4
  - Jeder Teilplan ist quellspezifisch
  - Kann von einer Quelle ausgeführt werden
- Wer führt die Joins zwischen den Teilplänen aus?
- System kann auswählen
  - Alle Pläne? (teuer, vollständig)
  - Schnellster Plan?
  - Billigster Plan?
  - Plan mit dem größten Ergebnis?

Schritt 1: Anfrageplanung

- Warum sollte man imdb.movie und fd.spielt zusammen in einem Plan benutzen?
  - Weil Quellen Fehler (fehlende Werte) enthalten
  - Kombination kann neue (und wichtige!) Tupel produzieren
  - Annahme: Gleiche Werte in den Joinattributen (Hier: Filmnamen, Schauspielernamen)
- Vorsicht vor impliziten Konsistenzannahmen

Beispiel

```
SELECT titel, regisseur, rolle
FROM   film, spielt
WHERE  spielt.schauspieler_name = 'Hans Albers'
AND    spielt.Rolle = 'Hauptrolle'
AND    spielt.titel = film.titel
```



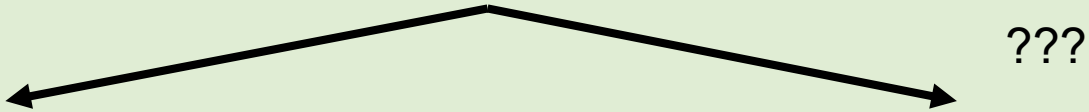
Schritt 2: Anfrageumschreibung

Jeder **Teilplan** kann an genau **eine Quelle** geschickt werden.

- Übersetzung der Schemaelementnamen
- Übersetzung einer SQL Anfrage in einen Web-Service, HTTP-Aufruf, XQuery, ...
- Übersetzung von Konstanten in einer Query
  - Was heißt 'Hauptrolle' in der imdb?

Anfrageübersetzung

```
SELECT titel, rolle
FROM   spielt
WHERE  schauspieler_name=‚Hans Albers‘
AND    rolle=‚Hauptrolle‘
```




```
SELECT title, role
FROM   imdb.acts
WHERE  actor_name=‚Hans Albers‘
AND    role=‚main role‘
```

```
SELECT title, role
FROM   imdb.acts
WHERE  actor_name=‚Hans Albers‘
```

Schritt 3: Anfrageoptimierung

- **Lokale Optimierung:** Jeder Plan einzeln
- **Globale Optimierung:** Gesamtmenge aller Pläne gemeinsam betrachten

Fragen:

- Optimale Reihenfolge der Ausführung der Teilpläne?
- Möglicherweise parallele Ausführung?
- Selektionsbedingungen global oder lokal prüfen?

```
SELECT   titel, director, role
FROM     imdb.movie, imdb.acts
WHERE    imdb.acts.actor_name = 'Hans Albers'
AND      imdb.acts.role = 'main role'
AND      imdb.acts.title = imdb.movie.title
```

Teilpläne

```
TP1:
SELECT   title, director
FROM     imdb.movie;

TP2:
SELECT   title, role
FROM     imdb.acts
WHERE    actor_name=‚Hans Albers‘
AND      role=‚main role‘;
```

Schritt 3: Anfrageoptimierung

Erst TP1, dann TP2, dann Join im Mediator

- Verlangt nach Download aller Filme der IMDB
- Vermutlich sehr teuer

Besser: TP1 und TP2 parallel

- Ergebnis von TP1 ist für TP2 unerheblich
- Immer noch sehr teuer

```
SELECT   titel, director, role
FROM     imdb.movie, imdb.acts
WHERE    imdb.acts.actor_name = 'Hans Albers'
AND      imdb.acts.role = 'main role'
AND      imdb.acts.title = imdb.movie.title
```

Teilpläne

```
TP1:
SELECT   title, director
FROM     imdb.movie;

TP2:
SELECT   title, role
FROM     imdb.acts
WHERE    actor_name=‚Hans Albers‘
AND      role=‚main role‘;
```

Schritt 3: Anfrageoptimierung

Erst TP2, dann TP1 umschreiben, Join im Mediator

- TP2: nur wenige Tupel
- Pushen dieser Tupel an TP1 „WHERE title IN (...)“
- Keine Parallelisierung mehr möglich
- Viel kleinere Zwischenergebnisse, vermutlich viel schneller

```
SELECT   titel, director, role
FROM     imdb.movie, imdb.acts
WHERE    imdb.acts.actor_name = 'Hans Albers'
AND      imdb.acts.role = 'main role'
AND      imdb.acts.title = imdb.movie.title
```

Teilpläne

```
TP1:
SELECT   title, director
FROM     imdb.movie;

TP2:
SELECT   title, role
FROM     imdb.acts
WHERE    actor_name=‚Hans Albers‘
AND      role=‚main role‘;
```

Schritt 4: Anfrageausführung

- Jeder der vier Pläne wurde optimiert und übersetzt
- Teilpläne müssen ausgeführt werden
  - Anfrage abschicken, Ergebnis abwarten, lokal puffern
  - Time-Outs überwachen
  - Müssen alle Teilpläne ausgeführt werden?
    - Wenn wir P1-P4 betrachten und keine Joinbedingungen pushen können
    - Ergibt 8 Teilpläne - aber nur 4 verschiedene Teilpläne
    - Optimierungspotential
    - Entweder global optimieren oder dynamisch durch Caching erkennen
- Fehlende Operationen müssen im Mediator ausgeführt werden
  - Nicht-pushbare Selektionen
  - Joins zwischen Relationen verschiedener Quellen

Schritt 5: Ergebnisintegration

- Jeder Plan liefert eine Menge korrekter Ergebnisse
- Trotzdem kann es Probleme geben
  - Duplikate: Objekte sind in mehreren Quellen vorhanden
  - Konflikte: Objekte sind mehrmals mit widersprüchlichen Angaben vorhanden

Anfrageübersetzung

`Imdb.movie`

Vögel, Hitchcock
Münchhausen, von Baky

`Imdb.acts`

Vögel, Kelly, Hauptrolle
Münchhausen, Bendow, Nebenrolle

`fd.film`

Vögel, Hitchcock
Münchhausen, Bürger

`fd.spielt`

Vögel, Kelly, Hauptrolle
Münchhausen, Albers, Hauptrolle

... ⋈ ... **UNION** ... ⋈ ... **UNION** ...

–

Münchhausen, Bürger, Hauptrolle
Münchhausen, von Baky, Hauptrolle

–

Global as View

Anfragebearbeitung – Global as View

- Gegeben:
  - Anfrage gegen globales Schema
  - GaV *Mapping*: für jede globale Relation genau eine Sicht auf lokale Quellen
- Gesucht:
  - Alle Tupel, die Anfragebedingungen erfüllen
  - Aber: Daten sind in lokalen Quellen gespeichert!
- Idee: Ersetze jede Relation der Anfrage durch ihre Sicht (*View Expansion*, *Query Unfolding*)
- Resultat: geschachtelte Anfrage gegen Quellschemata

Global as View (Beispiel)

Globales Schema

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

S1: IMDB(Titel, Regie, Jahr, Genre)
S2: RegieDB(Titel, Regie)
S3: GenreDB(Titel, Jahr, Genre)

```
SELECT Titel, Jahr
FROM Film
WHERE Jahr = ‚2003‘
```

Globale Sicht für Film

```
SELECT Titel, Jahr
FROM (SELECT * FROM IMDB
      UNION
      SELECT R.Titel, R.Regie, G.Jahr, G.Genre
      FROM RegieDB R, GenreDB G
      WHERE R.Titel = G.Titel
)
WHERE Jahr = ‚2003‘
```

Global as View (Beispiel)

```
SELECT F.Titel, P.Kino
FROM    Film F, Programm P
WHERE F.Titel = P.Titel
AND       P.Zeit > 20:00
```

S1: IMDB(Titel, Regie, Jahr, Genre)
S2: RegieDB(Titel, Regie)
S3: GenreDB(Titel, Jahr, Genre)
S7: KinoDB(Titel, Kino, Genre, Zeit)

```
SELECT F.Titel, P.Kino
FROM (SELECT * FROM IMDB
      UNION
      SELECT R.Titel, R.Regie, G.Jahr, G.Genre
      FROM RegieDB R, GenreDB G
      WHERE R.Titel = G.Titel
     ) AS F,
     (SELECT *
      FROM KinoDB
     ) AS P
WHERE F.Titel = P.Titel
AND   P.Zeit > 20:00
```

Global as View

Geschachtelte Anfragen

- Beliebig tiefe Schachtelung
- Bearbeitung
  - Konzeptionell: Ausführung der Anfragen von innen nach außen und Speicherung in temporären Relationen.
  - Tatsächlich: Optimierungspotential durch Umschreiben der Anfrage (Entschachtelung)

Global as View (Umschreibung)

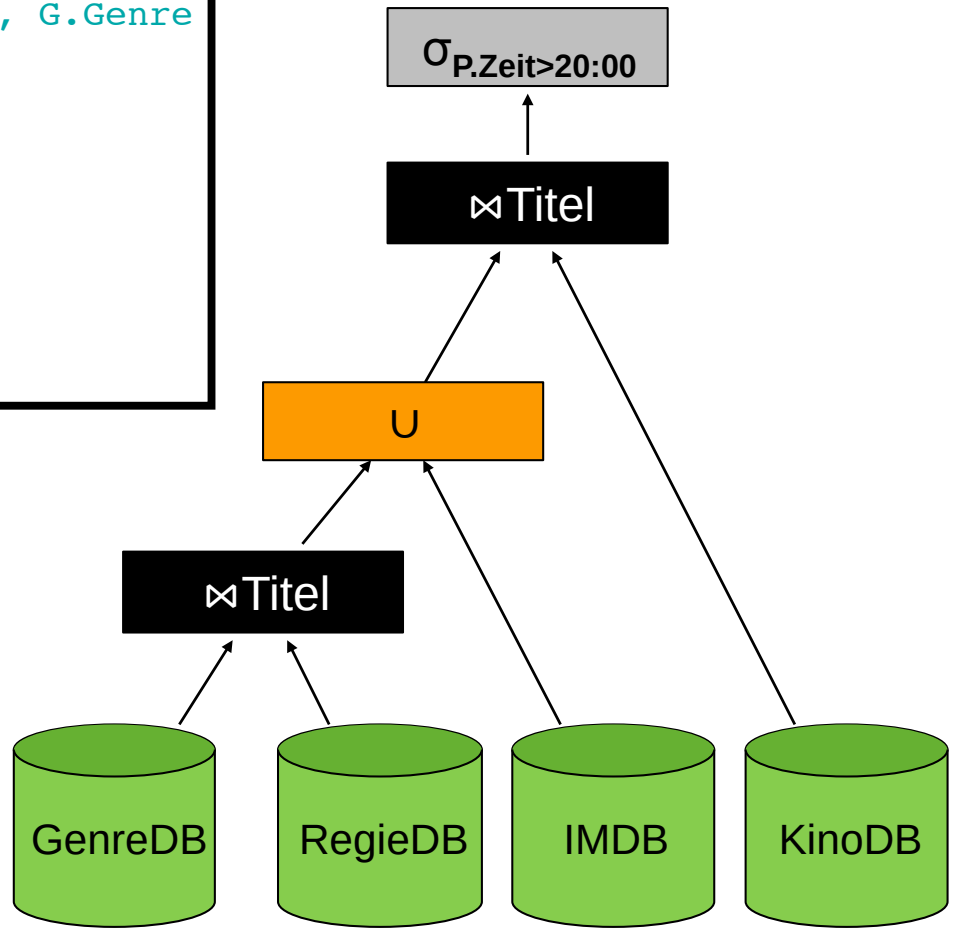
- Entschachtelung

```
SELECT F.Titel, P.Kino
FROM (SELECT * FROM IMDB
      UNION
      SELECT R.Titel, R.Regie, G.Jahr, G.Genre
      FROM RegieDB R, GenreDB G
      WHERE R.Titel = G.Titel
     ) AS F,
     (SELECT *
      FROM KinoDB
     ) AS P
WHERE F.Titel = P.Titel
AND   P.Zeit > 20:00
```

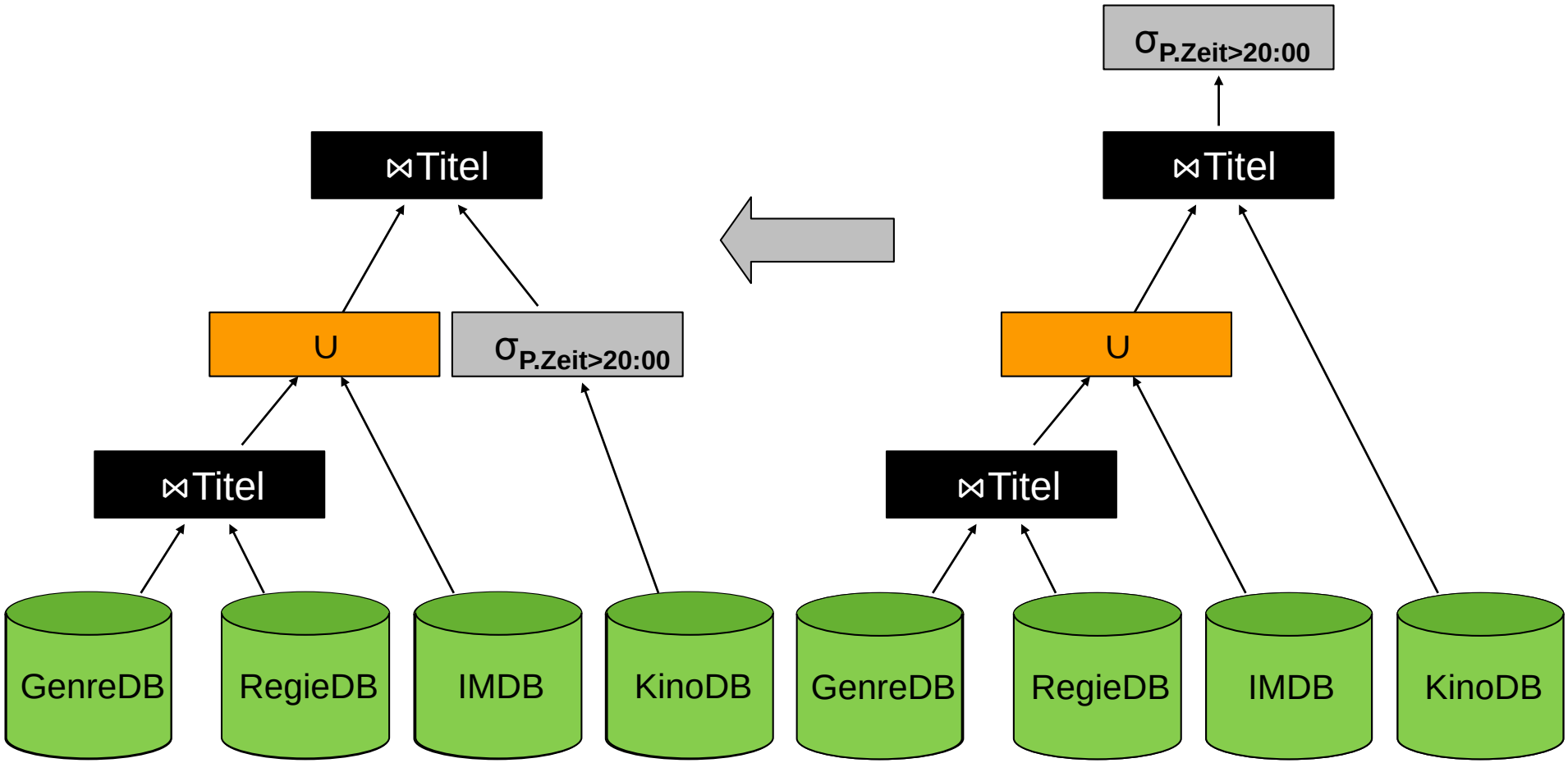
```
SELECT  F.Titel, P.Kino
FROM    IMDB F, KinoDB P
WHERE   F.Titel = P.Titel
AND     P.Zeit > 20:00
UNION
SELECT  F1.Titel, P1.Kino
FROM    RegieDB F1,
        GenreDB F2,
        KinoDB P1
WHERE   F1.Titel = F2.Titel
WHERE   F1.Titel = P1.Titel
AND     P1.Zeit > 20:00
```

Global as View (Umschreibung)

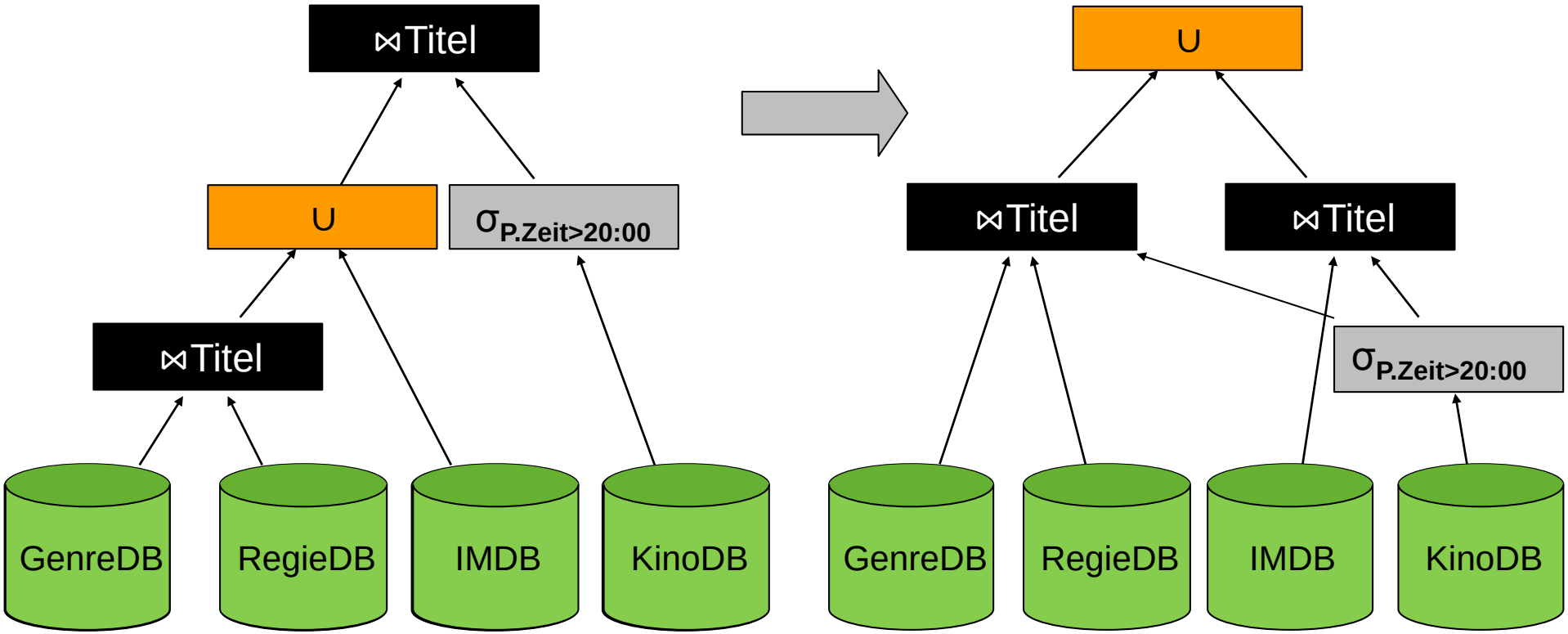
```
SELECT F.Titel, P.Kino
FROM (SELECT * FROM IMDB
      UNION
      SELECT R.Titel, R.Regie, G.Jahr, G.Genre
      FROM RegieDB R, GenreDB G
      WHERE R.Titel = G.Titel
      ) AS F,
      (SELECT *
       FROM KinoDB
      ) AS P
WHERE F.Titel = P.Titel
AND   P.Zeit > 20:00
```



Global as View (Umschreibung)



Global as View (Umschreibung)



Local as View

Anfragebearbeitung – LaV

- Gegeben:
  - Anfrage gegen globales Schema
  - LaV *Mapping*: für jede lokale Relation genau eine Sicht auf globales Schema
- Gesucht:
  - Alle Tupel, die die Anfragebedingungen erfüllen
  - Aber: Daten sind in lokalen Quellen gespeichert!

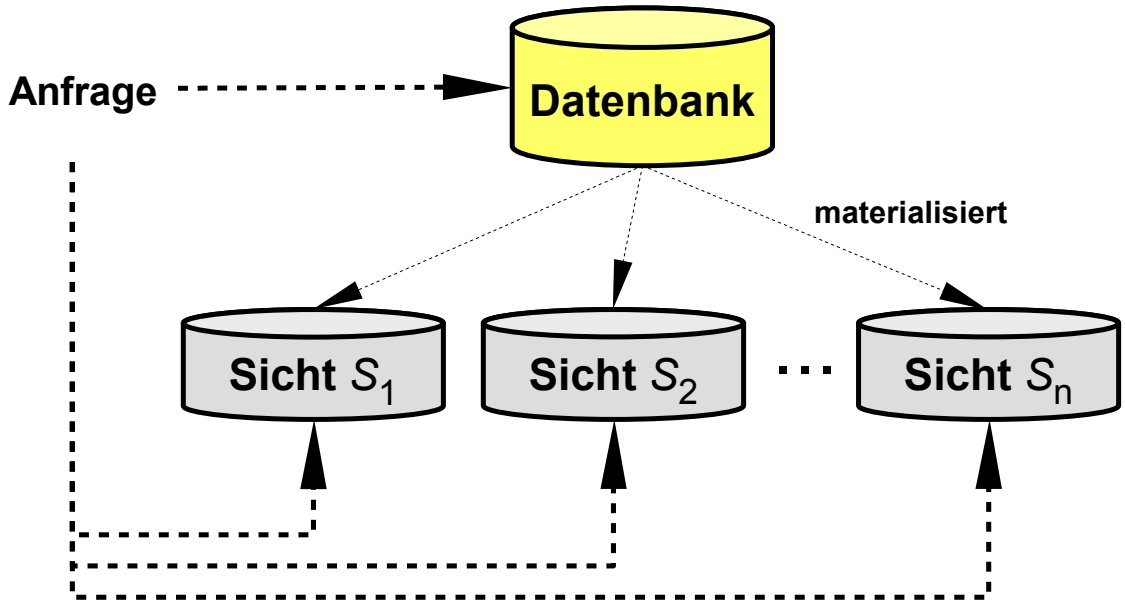
Anfragebearbeitung Local as View

Query Answering Using Views

Idee:

- 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

Query Answering Using Views



- Gegeben:
  - Datenbank
  - Menge an materialisierten Sichten (materialisiert da teuer)
  - Anfrage q gegen Datenbank
- Gewünscht:
  - Performanzoptimierung durch Nutzung der Sichten
  - Anfrage q' gegen Sichten die äquivalent zu q ist

Local as View (Beispiel)

Globales Schema

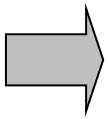
Lehrt(prof,kurs_id, sem, eval, univ)
Kurs(kurs_id, titel, univ)

Globale Anfrage

```
SELECT prof
FROM Lehrt L, Kurs K
WHERE L.kurs_id = K.kurs_id
AND K.titel LIKE „%_Datenbanken“
AND L.univ = „HPI“
```

Quelle 1: Alle Datenbankveranstaltungen

```
CREATE VIEW DB-kurs AS
SELECT K.titel, L.prof, K.kurs_id, K.univ
FROM   Lehrt L, Kurs K
WHERE  L.kurs_id = K.kurs_id
AND    L.univ = K.univ
AND    K.titel LIKE „%_Datenbanken“
```



Umgeschriebene Anfrage

```
SELECT prof
FROM DB-kurs D
WHERE D.univ = „HPI“
```

Quelle 2: Alle HPI-Vorlesungen

```
CREATE VIEW HPI-VL AS
SELECT K.titel, L.prof, K.kurs_id, K.univ
FROM   Lehrt L, Kurs K
WHERE  L.kurs_id = K.kurs_id
AND    K.univ = „HPI“
AND    L.univ =  „HPI“
AND    K.titel LIKE „%VL_%“
```

Warum nicht Quelle 2 einbeziehen?

Local as View (Beispiel)

Globales Schema

Lehrt(prof,kurs_id, sem, eval, univ)
Kurs(kurs_id, titel, univ)

Quelle 1: Alle Datenbankveranstaltungen

```
CREATE VIEW DB-kurs AS
SELECT K.titel, L.prof, K.kurs_id, K.univ
FROM   Lehrt L, Kurs K
WHERE  L.kurs_id = K.kurs_id
AND    L.univ = K.univ
AND    K.titel LIKE „%_Datenbanken“
```

Quelle 2: Alle HPI-Vorlesungen

```
CREATE VIEW HPI-VL AS
SELECT K.titel, L.prof, K.kurs_id, K.univ
FROM   Lehrt L, Kurs K
WHERE  L.kurs_id = K.kurs_id
AND    K.univ = „HPI“
AND    L.univ =  „HPI“
AND    K.titel LIKE „%VL_%“
```

Globale Anfrage

```
SELECT titel, kurs_id
FROM Kurs K
WHERE L.univ = „HPI“
```

Umgeschriebene Anfrage

```
SELECT titel, kurs_id
FROM DB-kurs D
WHERE D.univ = „HPI“
UNION
SELECT titel, kurs_id
FROM HPI-VL
```

Warum hier doch Quelle 2 einbeziehen?

Anfrageplan

- $q_1 \bowtie \ldots \bowtie q_n$ wird als Anfrageplan bezeichnet

Anfrageplan

Gegeben eine globale Anfrage q. Eine **Anfrageplan** p für q ist eine Anfrage der Form $q_1 \bowtie \ldots \bowtie q_n$, so dass

- jedes q_i von genau einem Wrapper ausgeführt werden kann,
- und jedes von p berechnete Tupel eine **semantisch korrekte** Antwort für q ist

- Bemerkungen
  - 'Semantisch korrekt' haben wir noch nicht definiert
  - In der Regel gibt es viele Anfragepläne
  - Die q_i heißen **Teilanfragen** oder **Teilpläne**

Anfrageplanung

- Gegeben: Anfrage q an das globale Schema
- Gesucht: Sequenz von Anfragen $q_1 \diamond \ldots \diamond q_n$

- Jedes q_i kann von einem Wrapper ausgeführt werden
- Die geeignete Verknüpfung von $q_1, \ldots, q_n$ beantwortet q
  - Innerhalb eines Plans durch Joins: $\diamond \rightarrow \bowtie$
  - Verschiedene Pläne werden durch UNION zusammengefasst: $\diamond \rightarrow \cup$
- Von $q_1 \bowtie \ldots \bowtie q_n$ berechnete Tupel sind korrekte Antworten auf Anfrage q

Query Containment

Intuitiv wollen wir das Folgende:

- Eine View v liefert (nur) semantisch korrekte Anfragen auf eine globale Anfrage q, wenn ihre Extension im Ergebnis von q **enthalten** ist.

Query Containment

Sei S ein Datenbankschema, I eine Instanz von S und q_1, q_2 Anfragen gegen I. Sei $q(I)$ das Ergebnis einer Anfrage q angewandt auf I.
Dann ist q_1 **enthalten** in q_2, geschrieben $q_1 \subseteq q_2$ genau dann wenn $q_1(I) \subseteq q_2(I)$ **für alle möglichen** I.

- Bemerkungen
  - Der wichtige Teil ist der letzte: „für alle möglichen Instanzen von S“
  - Die können wir natürlich nicht alle ausprobieren

Query Containment (Beispiel)

```
SELECT K.titel, L.prof, K.kurs_id, K.univ
FROM   Lehrt L, Kurs K
WHERE  L.kurs_id = K.kurs_id
AND    K.univ = „Humboldt“
AND    L.univ =  „Humboldt“
AND    K.titel LIKE „%VL_%“
```

$\subseteq$

```
SELECT  K.titel, L.prof, K.kurs_id, K.univ
FROM    Lehrt L, Kurs K
WHERE   L.kurs_id = K.kurs_id
AND     K.univ = „Humboldt“
AND     L.univ =  „Humboldt“
```

Query Containment (Beispiel)

Quelle 3:

```
CREATE VIEW Hum-Kurse AS
SELECT K.titel, K.univ
FROM   Lehrt L, Kurs K
WHERE  L.kurs_id = K.kurs_id
AND    K.univ = „Humboldt“
AND    L.univ =  „Humboldt“
```

$\subseteq$

Quelle 5:

```
CREATE VIEW Kurse2 AS
SELECT K.titel, K.univ
FROM   Kurs K
AND    K.univ = „Humboldt“
```

Query Containment (Beispiel)

q_1:

```
SELECT K.titel, K.kurs_id
FROM    Kurs K
AND       K.univ = „Humboldt“
AND       K.titel LIKE „%VL_%“
```

q_2:

```
SELECT K.titel, K.univ
FROM    Kurs K
AND       K.univ = „Humboldt“
AND       K.titel LIKE „%VL_%“
```

q_3:

```
SELECT K.titel, K.kurs_id
FROM    Kurs K
AND       K.univ = „Humboldt“
```

q_4:

```
SELECT K.titel
FROM    Kurs K
AND       K.univ = „Humboldt“
```

Welche der Anfragen beinhaltet welche Andere?

- $q_1 \subseteq q_3$
- ansonsten kein Containment Verhältnis präsent

Query Containment

- Prüfung von Containment durch Prüfung aller möglichen Datenbanken?
  - Zu komplex!
- Prüfung von Containment durch Existenz eines **containment mapping**
  - Hängt von betrachteter Anfrageklasse ab
  - Verschiedene Algorithmen (siehe [CM77, DHI12])
  - Besprechen wir hier nicht

Semantische Korrektheit

Mit Query Containment können wir definieren, wann ein Plan **semantisch korrekt** ist

Semantische Korrektheit

Sei S ein globales Schema, q eine Anfrage gegen S, und p eine Verknüpfung von Views $v_1, \ldots, v_n$ gegen S, die als linke Seite von LaV Korrespondenzen definiert sind.
Dann ist p semantisch korrekt für q, genau dann wenn $p \subseteq q$.

- Bemerkungen
  - Also ist die Extension von p in der von q enthalten.
  - Die Extension von q gibt es natürlich nicht.

LaV – Anfragebearbeitung

- Gegeben: Anfrage Q und Sichten $V_1, V_2, \ldots, V_n$ (über dasselbe Schema!)
- Gesucht: Umgeschriebene Anfrage Q', die
  - bei Optimierung: mit *materialized views*, d.h. 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* von Anfrageergebnissen für alle möglichen Datenbankinstanzen D, also $Q'(D) \subseteq Q(D)$?
- Lösung: Query Containment Check [CM77] (Prädikate abbildbar, Ergebnisvariablen abbildbar, Selektionsprädikate kompatibel)

Anfrageumschreibung

- Prinzipiell:
  - Umgeschriebene Anfrage:
    Konjunktive Anfrage (Kombination) aus Sichten
  - Prüfe **jede Kombination** aus Sichten auf Query Containment
  - **Unendlich viele** Kombinationen, da eine Sicht auch mehrfach in eine Umschreibung eingehen kann.
- Verbesserung:
  - Satz: Umschreibung besitzt maximal so vielen Sichten wie die Anfrage Relationen [LRO96]
  - Geschickte Vorauswahl der Sichten: Nutzbarkeit
  - Bucket Algorithmus

Bucket Algorithmus

- Problem:
  - Man muss exponentiell viele Kombinationen auf containment prüfen
  - Schon die Prüfung selbst ist ein NP-vollständiges Problem
  - In der Regel sind die Kombinationen jedoch nicht groß (Anzahl der Relationen)
- Idee zur weiteren Verbesserung:
  - Reduktion der Anzahl der Kombinationen durch geschickte Vorauswahl
  - Jede Relation der Anfrage erhält einen bucket (Eimer)
  - Schritt 1: Füge in jeden bucket alle Sichten, die für die Relation nutzbar sind
  - Schritt 2: Prüfe alle Anfrageumschreibungs-Kombinationen, die aus jedem bucket genau eine Sicht enthalten auf Containment

Bucket Algorithmus (Beispiel)

Quelle 1:

```
CREATE VIEW V1 AS
SELECT E.stud, K.titel, K.sem, K.kurs_id
FROM   Eingeschrieben E, Kurs K
WHERE  E.kurs_id = K.kurs_id
AND    K.kurs_id ≥ 500
AND    E.sem ≥  WS98
```

Quelle 2:

```
CREATE VIEW V2 AS
SELECT E.stud, L.prof, E.sem, L.kurs_id
FROM   Eingeschrieben E, Lehrt L
WHERE  E.kurs_id = L.kurs_id
AND    E.sem = L.sem
```

Quelle 3:

```
CREATE VIEW V3 AS
SELECT E.stud, E.kurs_id
FROM   Eingeschrieben E
WHERE  E.sem ≤  WS94
```

Quelle 4:

```
CREATE VIEW V4 AS
SELECT L.prof, K.kurs_id, K.titel, E.sem
FROM   Eingeschrieben E, Kurs K, Lehrt L
WHERE  E.kurs_id = K.kurs_id
AND    E.kurs_id = L.kurs_id
AND    L.sem = E.sem
AND    E.sem ≤  WS97
```

```
V1(stud, titel, sem, kurs_id) :- E(stud,kurs_id,sem), K(kurs_id,titel),
                                 kurs_id ≥ 500, sem ≥  WS98

V2(stud, prof, sem, kurs_id) :- E(stud, kurs_id, sem), L(prof, kurs_id, sem)

V3(stud, kurs_id) :- E(stud, kurs_id, sem), sem ≤  WS94

V4(prof, kurs_id, titel, sem) :- L(prof, kurs_id, sem), K(kurs_id, titel),
                                 E(stud, kurs_id, sem), sem ≤  WS97
```

Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :- L(prof, kurs_id, sem),
                          E(stud,kurs_id,sem),
                          K(kurs_id, titel),
                          sem ≥  WS95, kurs_id ≥ 300
```

Anfrage Q (in SQL)

```
SELECT E.stud, K.kurs_id, L.prof
FROM   Lehrt L, Eingeschrieben E, Kurs K
WHERE  E.kurs_id = L.kurs_id
AND    E.sem = L.sem
AND    K.kurs_id = E.kurs_id
AND    E.sem ≥  WS95 AND kurs_id ≥ 300
```

```
V1(stud, titel, sem, kurs_id) :- E(stud,kurs_id,sem), K(kurs_id,titel),
                                 kurs_id ≥ 500, sem ≥  WS98

V2(stud, prof, sem, kurs_id) :- E(stud, kurs_id, sem), L(prof, kurs_id, sem)

V3(stud, kurs_id) :- E(stud, kurs_id, sem), sem ≤  WS94

V4(prof, kurs_id, titel, sem) :- L(prof, kurs_id, sem), K(kurs_id, titel),
                                 E(stud, kurs_id, sem), sem ≤  WS97
```

Bucket Algorithmus (Beispiel)

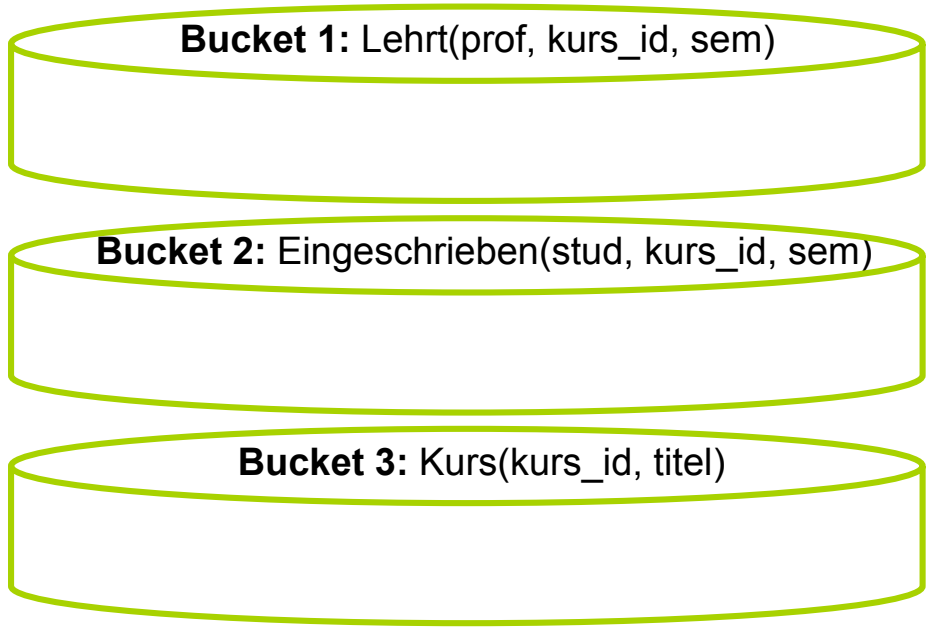
Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :- L(prof, kurs_id, sem),
                          E(stud,kurs_id,sem),
                          K(kurs_id, titel),
                          sem ≥  WS95, kurs_id ≥ 300
```

Teilziele (*subgoals*) von Q

Erzeuge für jedes Teilziel einen *bucket*.

Fülle Sichten in die *buckets*.



Bucket Algorithmus (en detail)

Eine Sicht wird einem bucket hinzugefügt, wenn mindestens ein Teilziel der Sicht 'geeignet' ist

Für jeden bucket:

Für jede Sicht:

Für jedes Teilziel der Sicht:

Prüfung auf 'Eignung' [DHI12]:

1. Beide Ziele betreffen dieselbe Relation
2. Kompatibilität: Prädikate sind passend
3. Alle Kopfvariablen werden exportiert
(d.h. alle Ausgabeattribute der Anfrage müssen Ausgabeattribute der Sicht sein)

Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

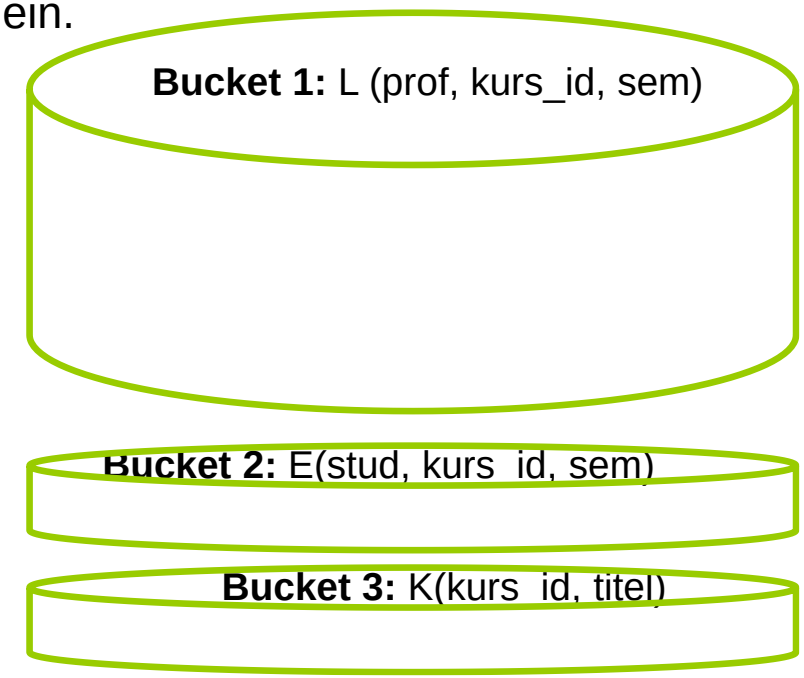
```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V1(stud, titel, sem, kurs_id) :-
    E(stud,kurs_id,sem),
    K(kurs_id,titel),
    kurs_id ≥ 500, sem ≥  WS98
```

Relation Lehrt nicht vorhanden!



Bucket Algorithmus (Beispiel)

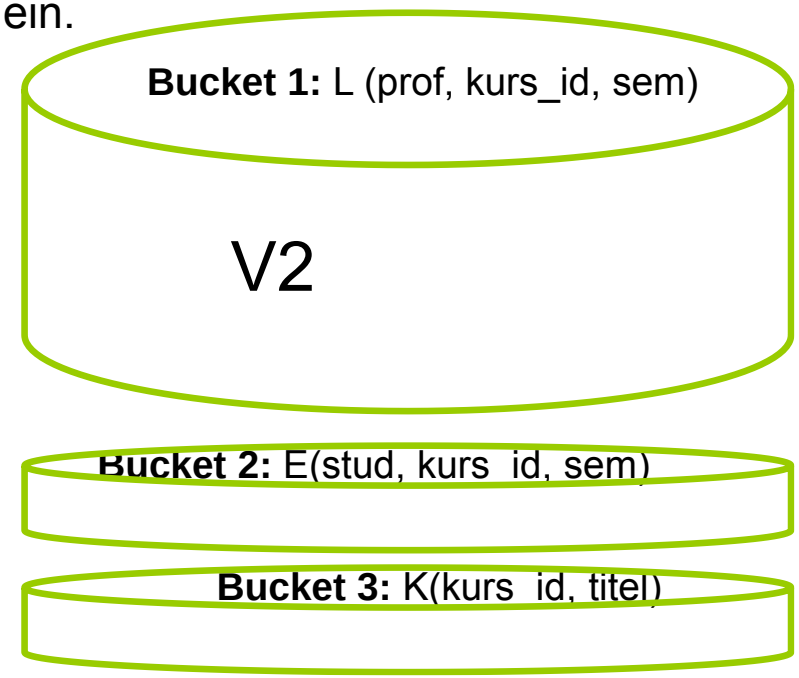
Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V2(stud, prof, sem, kurs_id) :-
    E(stud, kurs_id, sem),
    L(prof, kurs_id, sem)
```



Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

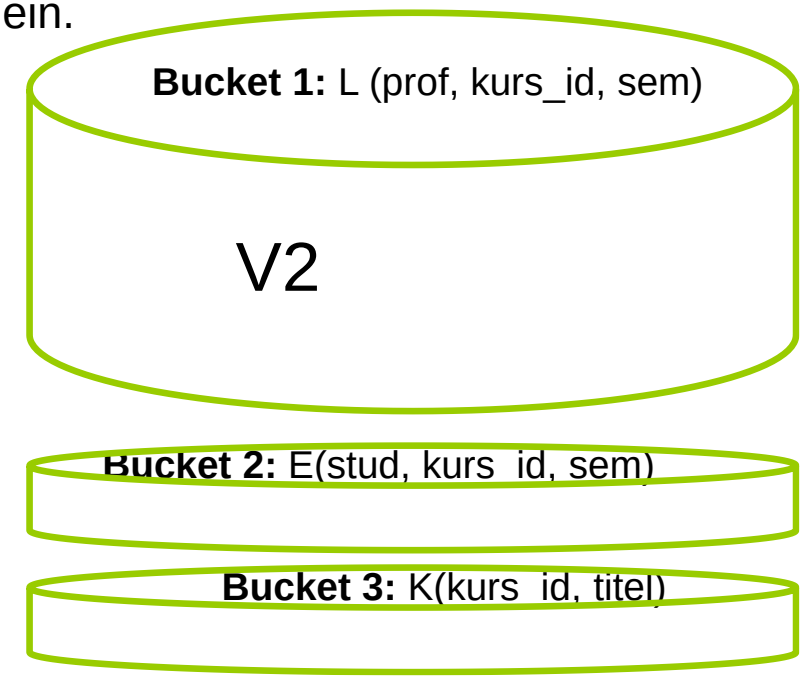
```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V3(stud, kurs_id) :-
    E(stud, kurs_id, sem),
    sem ≤  WS94
```

Relation Lehrt nicht vorhanden!



Bucket Algorithmus (Beispiel)

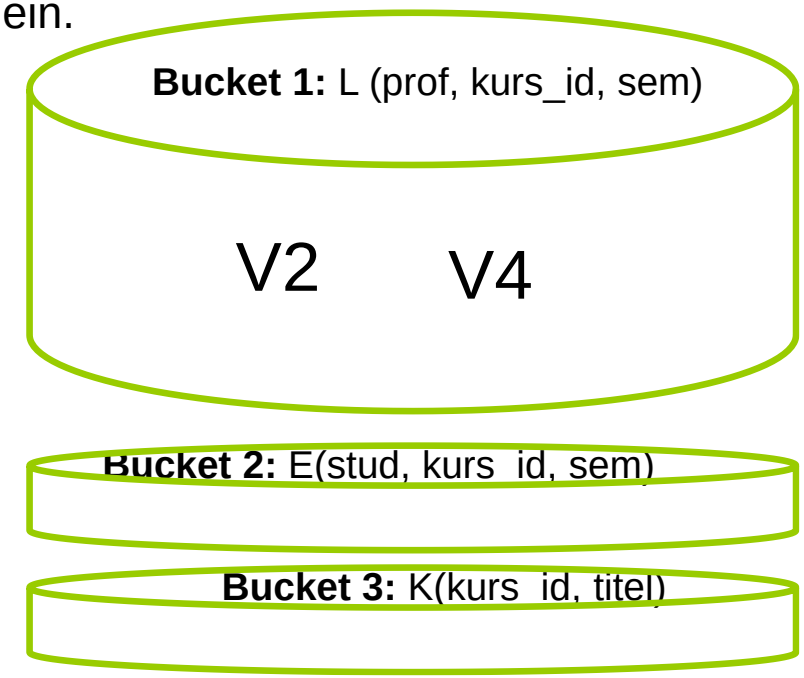
Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V4(prof, kurs_id, titel, sem) :-
    L(prof, kurs_id, sem),
    K(kurs_id, titel),
    E(stud, kurs_id, sem),
    sem ≤  WS97
```



Bucket Algorithmus (Beispiel)

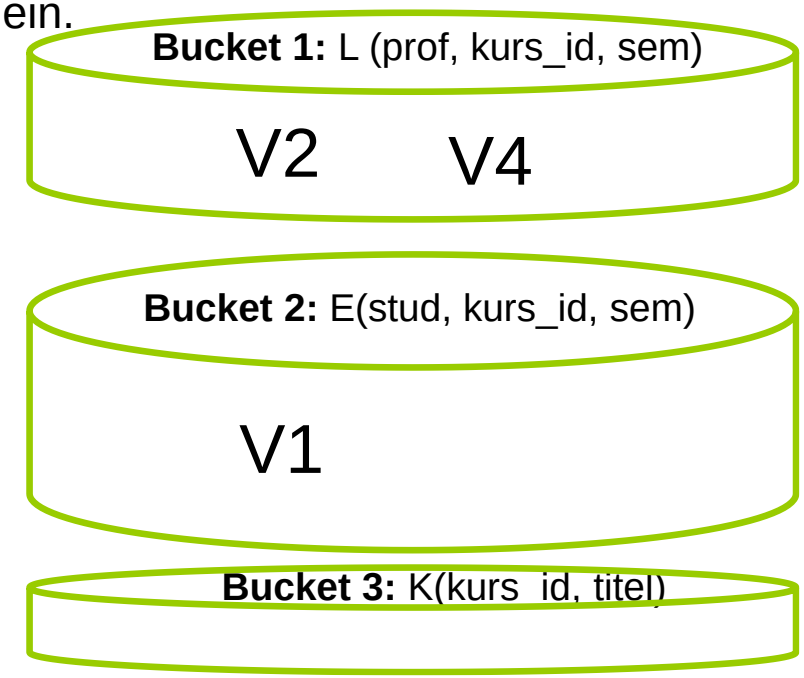
Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V1(stud, titel, sem, kurs_id) :-
    E(stud,kurs_id,sem),
    K(kurs_id,titel),
    kurs_id ≥ 500, sem ≥  WS98
```



Bucket Algorithmus (Beispiel)

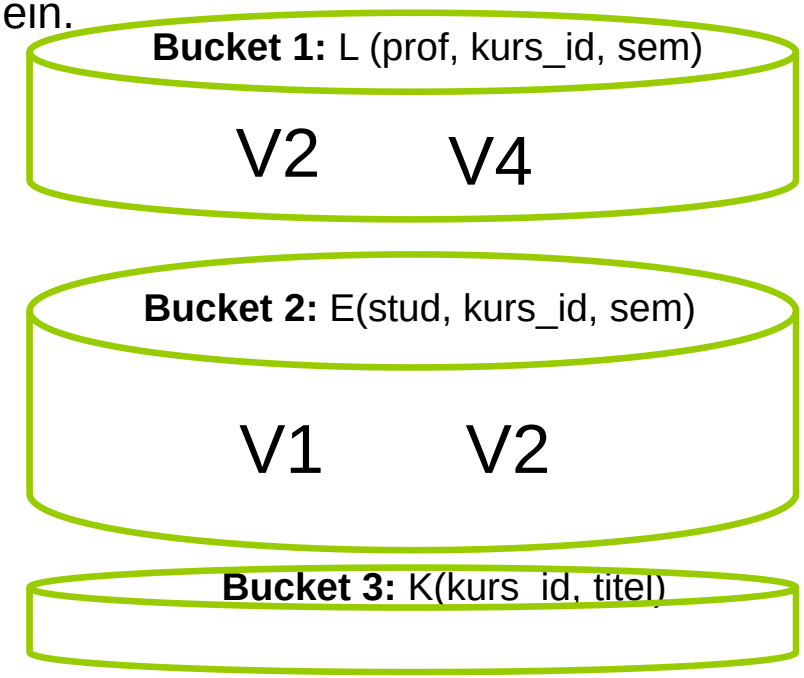
Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V2(stud, prof, sem, kurs_id) :-
    E(stud, kurs_id, sem),
    L(prof, kurs_id, sem)
```



Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

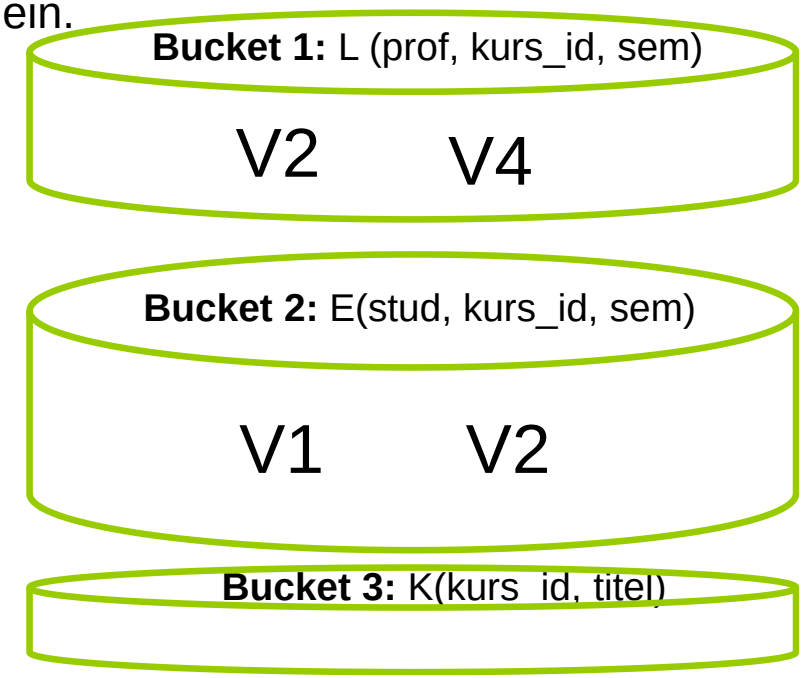
```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V3(stud, kurs_id) :-
    E(stud, kurs_id, sem),
    sem ≤  WS94
```

sem ≥ WS95 vs. sem ≤ WS94



Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

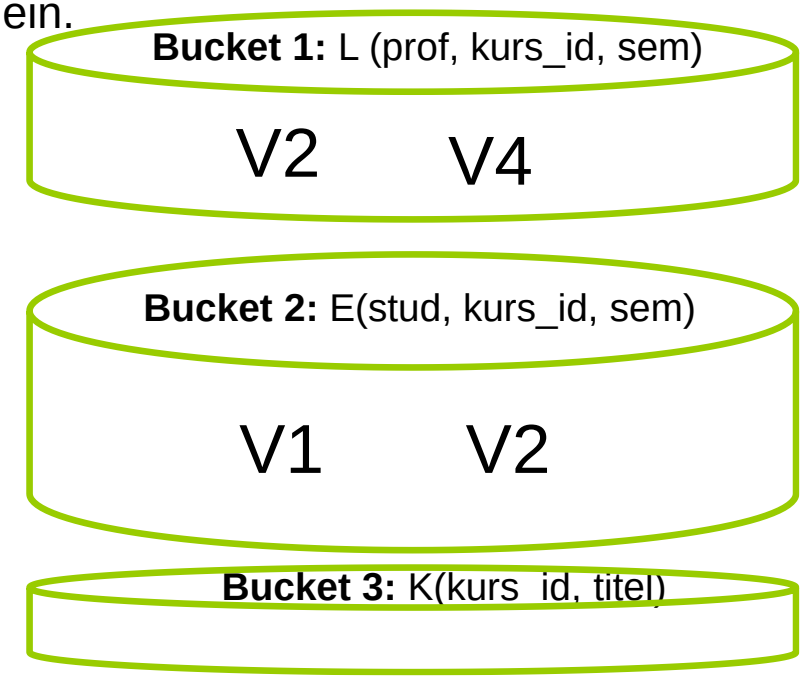
```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V4(prof, kurs_id, titel, sem) :-
    L(prof, kurs_id, sem),
    K(kurs_id, titel),
    E(stud, kurs_id, sem),
    sem ≤  WS97
```

stud wird nicht exportiert!



Bucket Algorithmus (Beispiel)

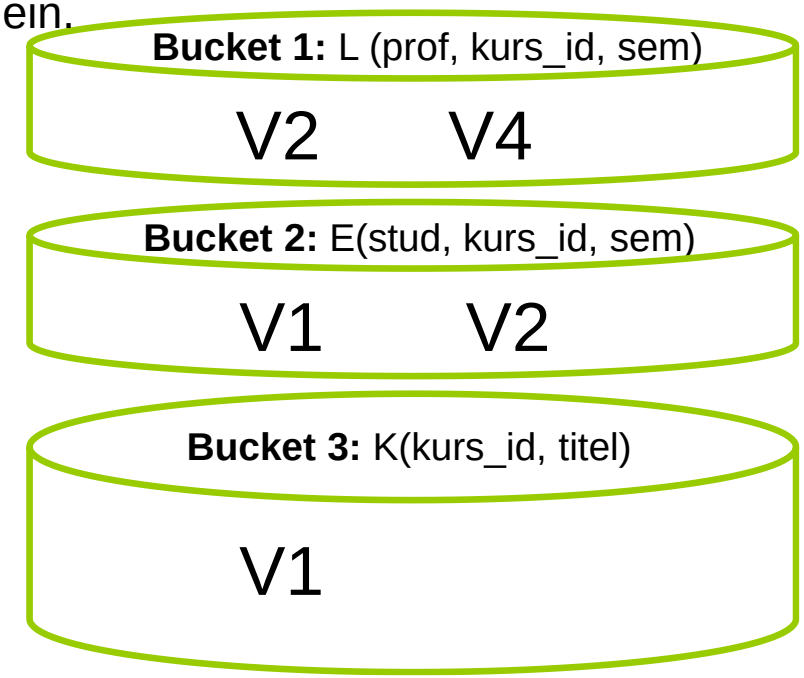
Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V1(stud, titel, sem, kurs_id) :-
    E(stud,kurs_id,sem),
    K(kurs_id,titel),
    kurs_id ≥ 500, sem ≥  WS98
```



Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

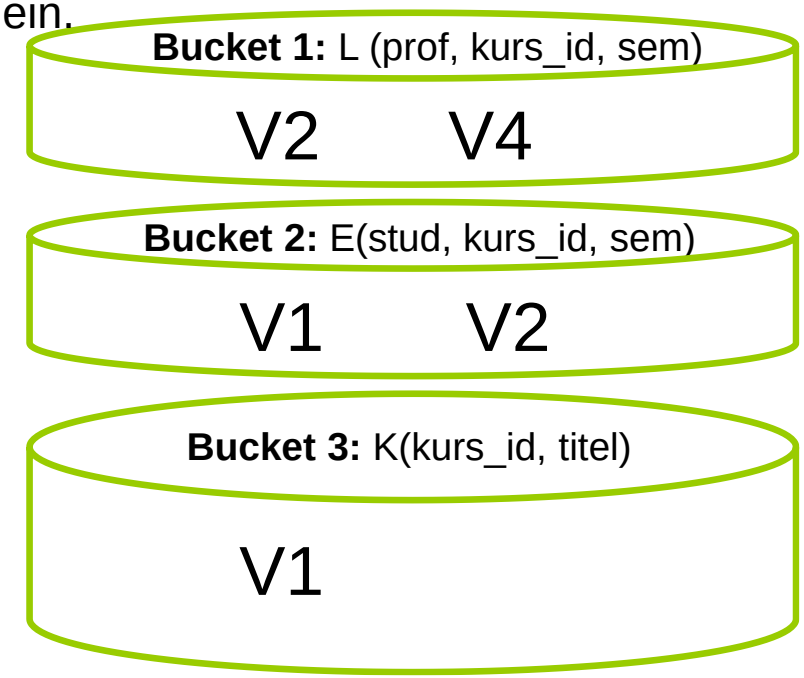
```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V2(stud, prof, sem, kurs_id) :-
    E(stud, kurs_id, sem),
    L(prof, kurs_id, sem)
```

Relation Kurs nicht vorhanden!



Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

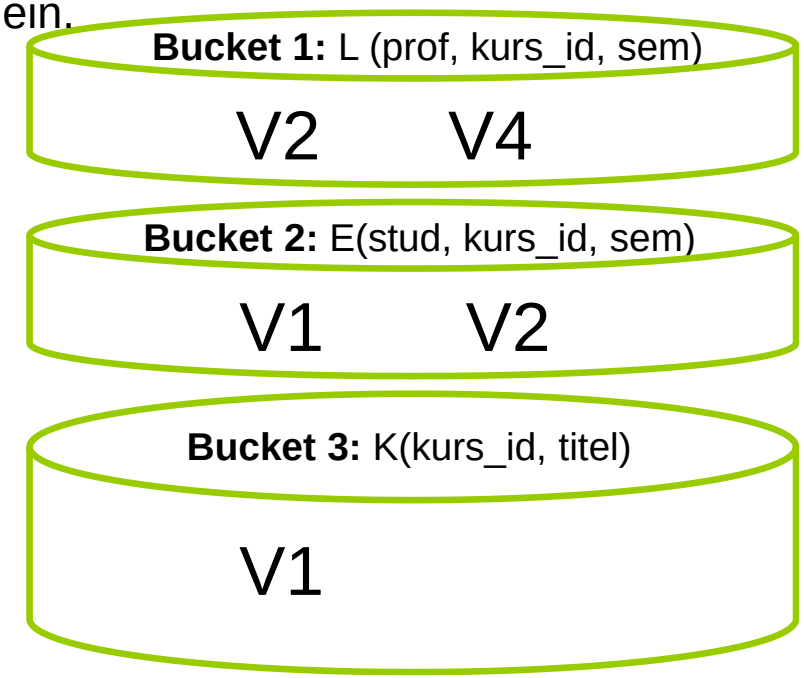
```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V3(stud, kurs_id) :-
    E(stud, kurs_id, sem),
    sem ≤  WS94
```

Relation Kurs nicht vorhanden und sem ≤ WS94!



Bucket Algorithmus (Beispiel)

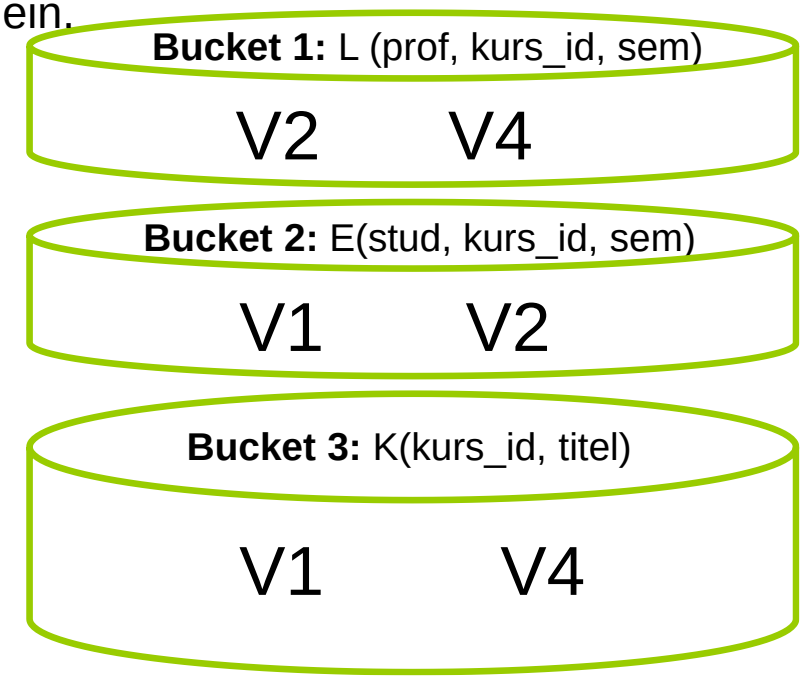
Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```

- Für jede Sicht: Betrachte deren Teilziele.
- Falls mind. ein Teilziel geeignet ist, füge Sicht in bucket ein.

Prüfung auf „Eignung“:
1. Selbe Relation.
2. Kompatibilität: Prädikate sind passend.
3. Alle Kopfvariablen werden exportiert

```
V4(prof, kurs_id, titel, sem) :-
    L(prof, kurs_id, sem),
    K(kurs_id, titel),
    E(stud, kurs_id, sem),
    sem ≤  WS97
```

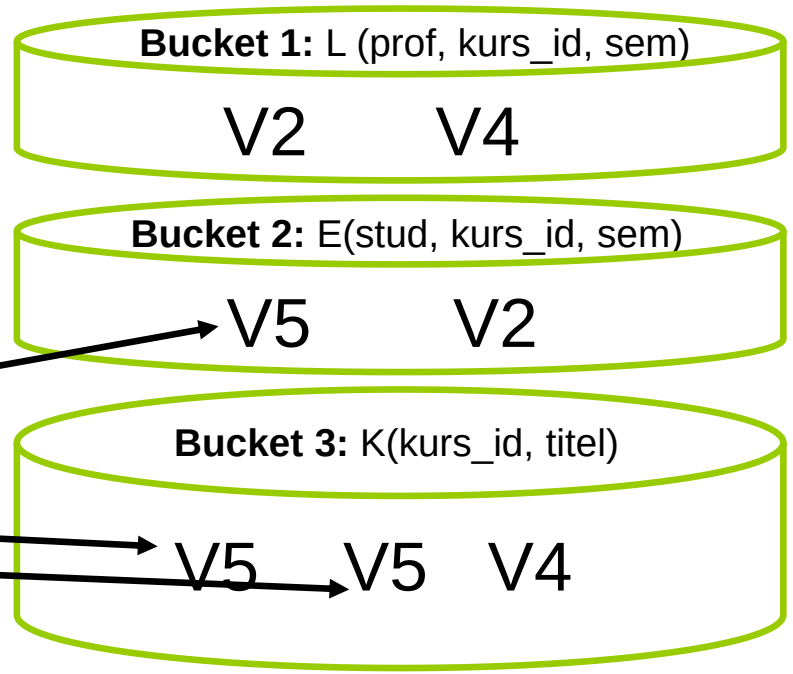


Bucket Algorithmus (Beispiel)

- Eine Sicht kann in verschiedenen Buckets auftauchen
  - Verschiedene Rollen
- Eine Sicht kann mehrmals in einem Bucket auftauchen
  - Wenn mehrere Teilziele passen

```
Q (stud, kurs_id, prof) :-
    L(prof, kurs_id, sem),
    E(stud,kurs_id,sem),
    K(kurs_id,titel),
    sem ≥  WS95,
    kurs_id ≥ 300
```

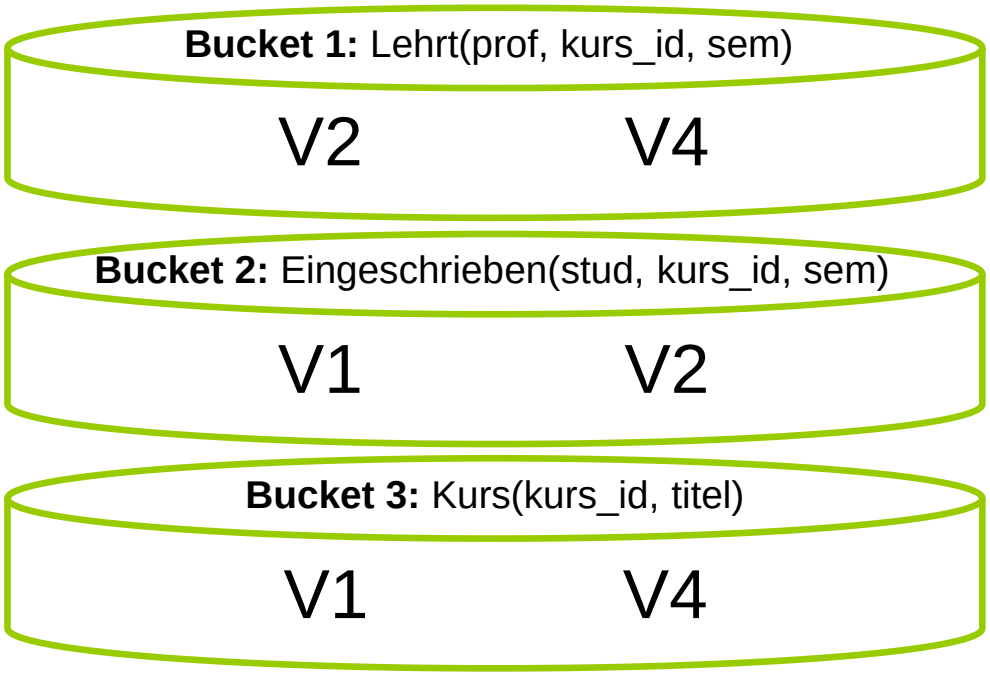
```
V5(stud,titel,kurs_id,kurs_id2) :-
    E(stud,kurs_id,sem),
    K(kurs_id,titel),
    K(kurs_id2,titel),
    kurs_id ≥ 500,
    sem ≥  WS98
```



Bucket Algorithmus (Beispiel)

Anfrage Q (in Datalog)

```
Q(stud, kurs_id, prof) :-  L(prof, kurs_id, sem),
                           E(stud,kurs_id,sem),
                           K(kurs_id, titel),
                           sem ≥  WS95, kurs_id ≥ 300
```



Kombinationen

V2, V1, V1
V4, V1, V4
V2, V2, V4
V2, V2, V1
V2, V1, V4
V4, V1, V1
V4, V2, V4
V4, V2, V1

Frage: Welche Kombinationen (Pläne) sind zulässig?

Bucket Algorithmus (Kombinationen)

- Wieviele Kombinationen?
  - $|B_1| \times |B_1| \times \ldots \times |B_n|$ $(n = |Q|)$
  - Falls m = Anzahl Sichten: m^n
  - Wichtig: Jede der exponentiell vielen Kombinationen liefert potentiell einen Teil des Ergebnisses
- Eine Kombination $Q' = V_1, \ldots, V_k$ führt zu einer Anfrageumschreibung von Q, falls
  - $Q' \subseteq Q$
  - oder $Q', \{\text{Prädikate}\} \subseteq Q$
  - oder es existiert ein Homomorphismus[1] ψ, so dass gilt $\psi(Q', \{\text{Prädikate}\}) \subseteq Q$

[1] Ein Homomorphismus bildet die Elemente aus der einen Menge so in die andere Menge ab, dass sich ihre Bilder dort hinsichtlich der Struktur ebenso verhalten, wie sich deren Urbilder in der Ausgangsmenge verhalten.

Bucket Algorithmus (Kombinationen)

```
V1 (stud, titel, sem, kurs_id) :- E(stud,kurs_id,sem), K(kurs_id,titel), kurs_id ≥ 500, sem ≥  WS98
V2 (stud, prof, sem, kurs_id) :- E(stud, kurs_id, sem), L(prof, kurs_id, sem)
V3 (stud, kurs_id) :- E(stud, kurs_id, sem), sem ≤  WS94
V4(prof, kurs_id, titel, sem) :- L(prof, kurs_id, sem),K(kurs_id, titel),E(stud, kurs_id, sem), sem ≤  WS97
```

Anfrage: `Q (stud, kurs_id, prof) :- L(prof, kurs_id, sem), E(stud,kurs_id,sem), K(kurs_id, titel), sem ≥ WS95, kurs_id ≥ 300`

Kombinationen

V2, V1, V1
V4, V1, V4
V2, V2, V4
V2, V2, V1
V2, V1, V4
V4, V1, V1
V4, V2, V4
V4, V2, V1

Einzelne Sichten nützlich (in Bezug auf Anfrage), aber zusammen Widerspruch: sem ≥ WS98 (V1) vs. sem ≤ WS97 (V4)

Bucket Algorithmus (Kombinationen)

```
V1 (stud, titel, sem, kurs_id) :- E(stud,kurs_id,sem), K(kurs_id,titel), kurs_id ≥ 500, sem ≥  WS98
V2 (stud, prof, sem, kurs_id) :- E(stud, kurs_id, sem), L(prof, kurs_id, sem)
V3 (stud, kurs_id) :- E(stud, kurs_id, sem), sem ≤  WS94
V4(prof, kurs_id, titel, sem) :- L(prof, kurs_id, sem),K(kurs_id, titel),E(stud, kurs_id, sem), sem ≤  WS9
```

Anfrage: `Q (stud, kurs_id, prof) :- L(prof, kurs_id, sem), E(stud,kurs_id,sem), K(kurs_id, titel), sem ≥ WS95, kurs_id ≥ 300`

Kombinationen

V2, V1, V1
~~V4, V1, V4~~
V2, V2, V4
V2, V2, V1
~~V2, V1, V4~~
~~V4, V1, V1~~
V4, V2, V4
~~V4, V2, V1~~

```
Q‘(stud, kurs_id, prof):-
  V2(stud‘, prof, sem, kurs_id),
  V1 (stud, titel‘, sem, kurs_id),
  V1 (stud‘, titel, sem‘, kurs_id)
```

x‘ markiert nicht gebrauchte Attribute.

Bucket Algorithmus (Kombinationen)

- Prüfe:

```
Q ⊇ Q'(stud, kurs_id, prof):-
                V2(stud', prof, sem, kurs_id),
                V1 (stud, titel', sem, kurs_id),
                V1 (stud', titel, sem', kurs_id)
```

- Wie kann man das prüfen wenn Q globale Relationen verwendet und Q' auf Sichten basiert
- Query Unfolding:

```
Q' = Q''(stud, kurs_id, prof):-
     E(stud', kurs_id, sem), L(prof, kurs_id, sem),
     E(stud,kurs_id,sem), K(kurs_id,titel'), kurs_id ≥ 500, sem ≥  WS98
```

- Finden eines **containment mappings**

Bucket Algorithmus (Kombinationen)

- Endgültiges Ergebnis: Kombination aller Kombinationen durch UNION:
- $V_1, V_2 \cup V_2, V_4$

Umgeschriebene Anfrage Q‘:

```
SELECT V1.stud, V1.kurs_id, V2.prof
FROM   V1, V2
WHERE  V1.sem = V2.sem
AND    V1.kurs_id = V2.kurs_id

UNION

SELECT V2.stud, V2.kurs_id, V2.prof
FROM   V2, V4
WHERE  V2.sem = V4.sem
AND    V2.kurs_id = V4.kurs_id
AND    V2.prof = V4.prof
```

Bucket Algorithmus (Analyse)

- Vollständigkeit: Der BA findet eine Anfrageumschreibung falls eine Umschreibung existiert
- BA verwirft nur Kombinationen, die unzulässig sind.
- BA garantiert nach [LMSS95] maximal enthaltene Umschreibungen, wenn die betreffende Anfrage keine arithmetischen Vergleiche mit $\geq, \leq, >, <$ oder $\neq$ beinhaltet

Weitere Verfahren:

- MiniCon Algorithmus
- Inverse Rules Algorithmus
- Improved Bucket Algorithmus

Global Local as View

Anfragebearbeitung – GLaV

- Gegeben:
  - Anfrage gegen globales Schema
  - GLaV *Mapping*: jede lokale Relation ist Teil genau einer Sicht auf globales Schema
- Gesucht:
  - Alle Tupel, die die Anfragebedingungen erfüllen
  - Aber: Daten sind in lokalen Quellen gespeichert!

Anfragebearbeitung Global Local as View

Eine GLaV-Regel kann als Komposition einer LaV-Regel und einer GaV-Regel betrachtet werden

Beispiel:

$\text{Kino(kid,k)}, \text{Genre(kid,g)} \subseteq \text{Film(t,r,j,g)}, \text{Prog(k,t,z)}$

wobei gilt:

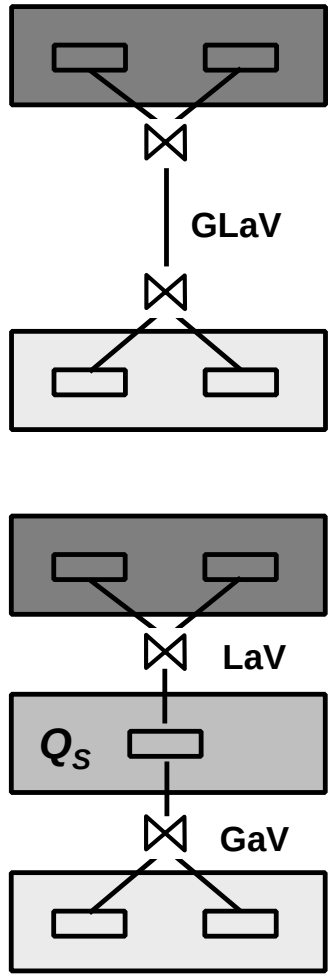
$Q_S = \text{Kino(kid,k)}, \text{Genre(kid,g)}$

$Q_G = \text{Film(t,r,j,g)}, \text{Prog(k,t,z)}$

Daher Komposition in:

LaV-Regel: $Q_S \subseteq \text{Film(t,r,j,g)}, \text{Prog(k,t,z)}$

GaV-Regel: $\text{Kino(kid,k)}, \text{Genre(kid,g)} \subseteq Q_S$



Anfragebearbeitung Global Local as View

Vorgehensweise:

- Schritt 1: Betrachte die linke Seite der GLaV-Regeln als einzelne **logische** Relation Q_{S_i}
  ($\Rightarrow$ GLaV-Regeln werden auf LaV-Regeln reduziert)
- Schritt 2: Verwende einen LaV-Algorithmus um einen logischen Anfrageplan zu erstellen
- Schritt 3: Benutze Query Unfolding um die logischen Relationen $Q_{S_1}, \ldots, Q_{S_k}$ durch die tatsächlichen Sichtendefinitionen zu ersetzen

Literatur

[CM77] Ashok K. Chandra and Philip M. Merlin.
Optimal implementation of conjunctive queries in relational data bases.
In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing*, 1977.

[DHI12] Anhai Doan, Alon Halevy, and Zachary Ives.
Principles of Data Integration.
Morgan Kaufmann, 2012.

[LMSS95] A.Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava.
Answering queries using views.
In *Proc. of the Symposium on Principles of Database Systems (PODS)*, 1995.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille.
Query-answering algorithms for information agents.
In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 1996.