Information Integration

Local-as-View: LaV

16.12.2019

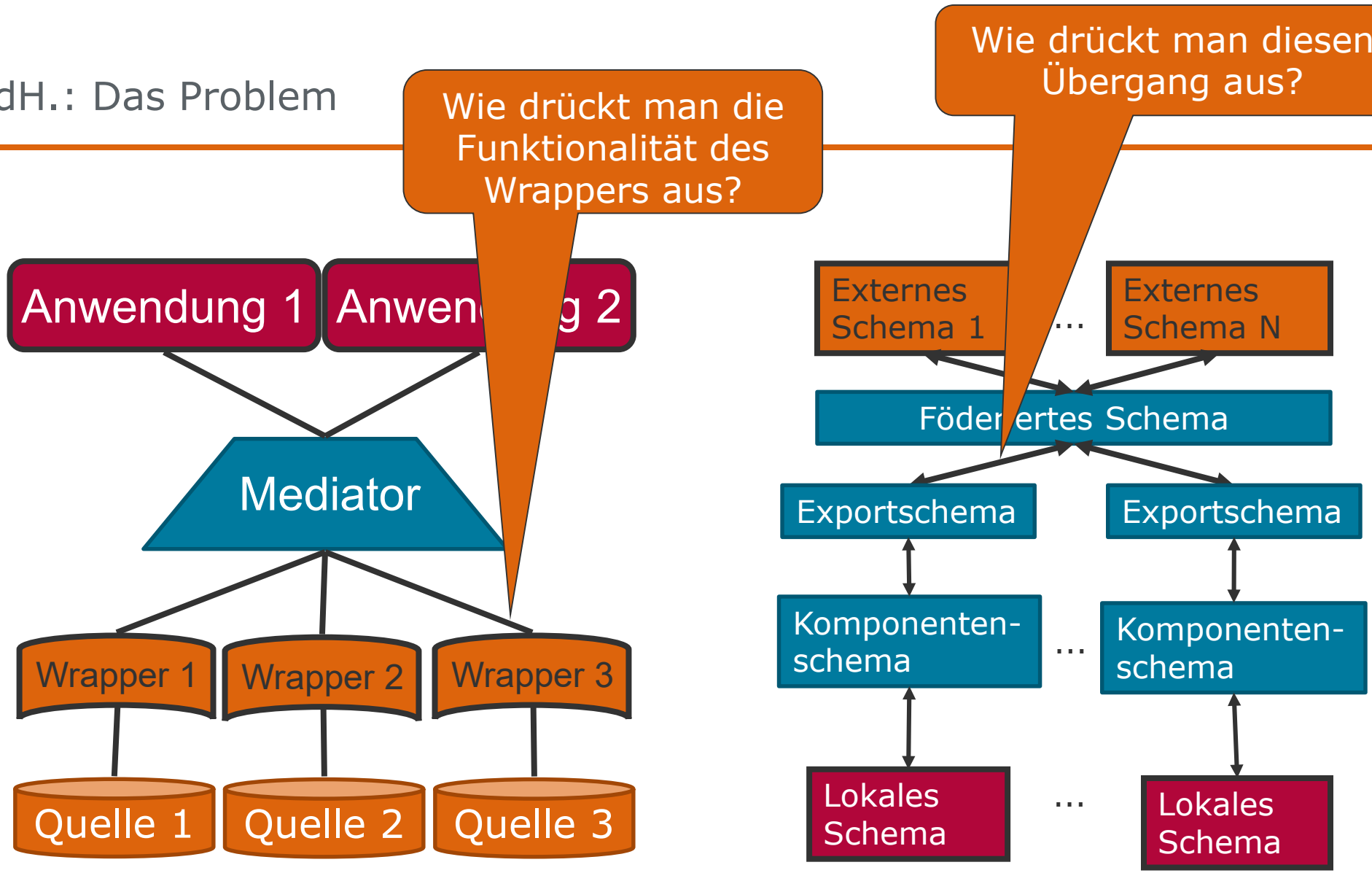
Felix Naumann

Überblick

1. Motivation
2. Korrespondenzen
3. Übersicht Anfrageplanung
4. Global as View (GaV)
5. Local as View (LaV)
   - Modellierung
   - Anwendungen
   - Anfragebearbeitung
   - Containment
6. Global Local as View (GLaV)
7. Vergleich

Felix Naumann
Information Integration
Winter 2019/20

WdH.: Das Problem



Felix Naumann
Information Integration
Winter 2019/20

Modellierung von Datenquellen (Wdh.)

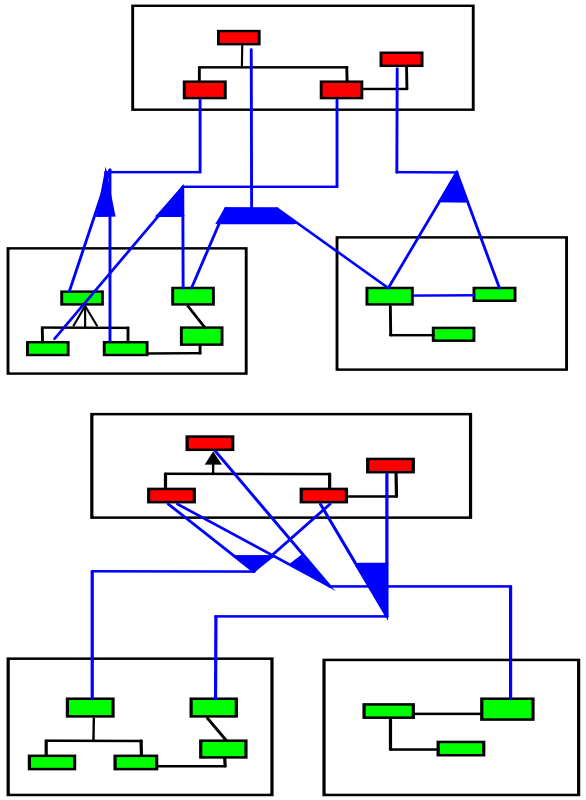
- Kernidee
  - Modellierung strukturell heterogener Quellen in Bezug auf ein globales Schema als Views (Sichten)
  - Relationales Modell

- Allgemein:
  - 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

- Jetzt: Unterscheidung lokales und globales Schema

Global as View / Local as View (Wdh.)

- Global as View
  - Relationen des globalen Schemata werden als Sichten auf die lokalen Schemas der Quellen ausgedrückt.
- Local as View
  - Relationen der Schemata der Quellen werden als Sichten auf das globale Schema ausgedrückt.



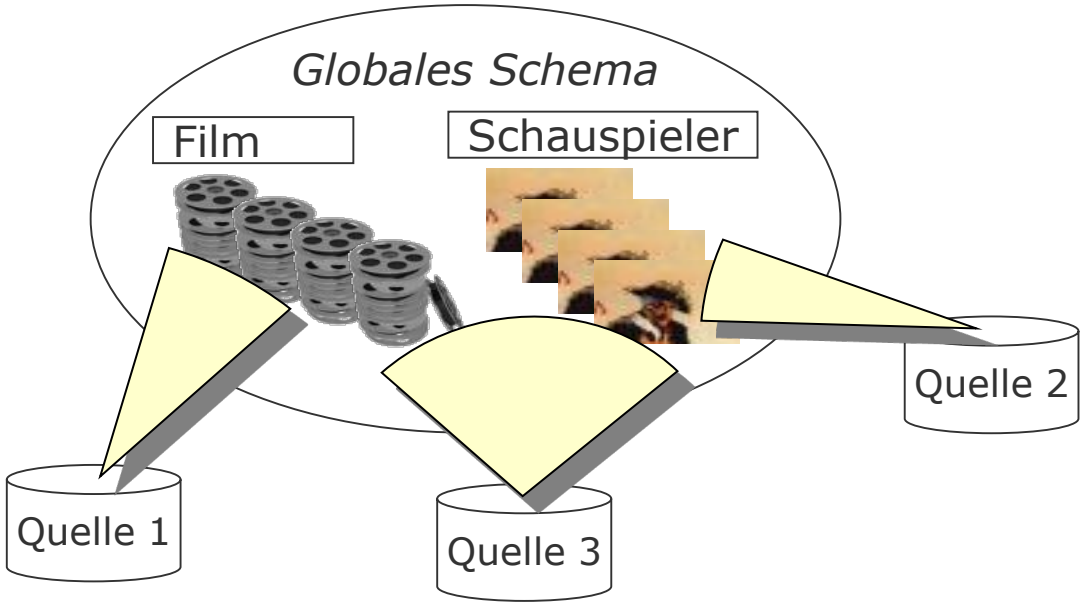
Felix Naumann
Information Integration
Winter 2019/20

Überblick

1. Motivation
2. Korrespondenzen
3. Übersicht Anfrageplanung
4. Global as View (GaV)
5. Local as View (LaV)
   - **Modellierung**
   - Anwendungen
   - Anfragebearbeitung
   - Containment
6. Global Local as View (GLaV)
7. Vergleich

Warum LaV? Eine andere Sichtweise

- Es gibt in der Welt eine Menge von Filmen, Schauspielern, …
- Das globale Schema modelliert diese Welt.
- Theoretisch steht damit die Extension 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.



Local as View (LaV) – Beispiel

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

S1: IMDB(Titel, Regie, Jahr, Genre)
S2: MyMovies(Titel, Regie, Jahr, Genre)
S3: RegieDB(Titel, Regie)
S4: GenreDB(Titel, Jahr, Genre)

```
CREATE VIEW S1 AS
SELECT * FROM Film

CREATE VIEW S2 AS
SELECT * FROM Film

CREATE VIEW S3 AS
SELECT Film.Titel, Film.Regie
FROM Film

CREATE VIEW S4 AS
SELECT Film.Titel, Film.Jahr,
            Film.Genre
FROM Film
```


Quelle: VL „Data Integration“, Alon Halevy, University of Washington, 2002

Local as View (LaV) – Beispiel

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

S9: ActorDB(Titel, Schauspieler, Jahr)

„Verpasste Chance“

```
CREATE VIEW S9 AS
SELECT Titel, NULL, Jahr
FROM Film
```

Felix Naumann
Information Integration
Winter 2019/20

Local as View (LaV) – Beispiel

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

S7: KinoDB(Kino, Genre)

```
CREATE VIEW S7 AS
SELECT Programm.Kino, Film.Genre
FROM Film, Programm
WHERE Film.Titel = Programm.Titel
```

- Assoziationen des globalen Schemas können in der Sicht hergestellt werden.

Local as View (LaV) – Beispiel

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

S9: Filme(Titel, Jahr, Ort, RegieID)
Regie(ID, Regisseur)

```
CREATE VIEW S9Filme AS
SELECT Titel, Jahr, NULL, NULL
FROM Film

CREATE VIEW S9Regie AS
SELECT NULL, Regie
FROM Film
```

- Assoziationen des lokalen Schemas können nicht abgebildet werden.

Local as View (LaV) – Beispiel

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

S8: NeueFilme(Titel, Regie, Genre)
(IC: Jahr > 2000)

```
CREATE VIEW S8 AS
SELECT Titel, Regie, Genre
FROM Film
WHERE Jahr > 2000
```

- IC auf der Quelle kann modelliert werden
  - Wenn das Attribut im globalen Schema existiert
- ICs müssen in der Quelle nicht explizit definiert werden
  - Auch implizite Einschränkungen können in den View aufgenommen werden.

Local as View (LaV) – Globale ICs

Globales Schema
NeuerFilm(Titel, Regie, Jahr, Genre)
Programm(Kino, Titel, Zeit)
Nebenbedingung: Jahr > 2000

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

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

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

- Nebenbedingungen auf dem globalen Schema können wir nicht sinnvoll modellieren.
- Das ging aber bei GaV
- Also hat beides Stärken und Schwächen

Local as View (LaV) – lokale ICs

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 = 2020)

Modellierung zur Optimierung

Modellierung zur Beantwortbarkeit

```
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 = 2020
```

Lokale ICs: Weitere Beispiele

Datenquelle	Beschreibung
`spielfilme(titel, regisseur, laenge)`	Informationen über Spielfilme, die mindestens 80 Minuten Länge haben.
`kurzfilme(titel, regisseur)`	Informationen über Kurzfilme. Kurzfilme sind höchstens 10 Minuten lang.
`filmkritiken(titel, regisseur, schauspieler, kritik)`	Kritiken zu Hauptdarstellern von Filmen
`us_spielfilme(titel, laenge, schauspieler_name)`	Spielfilme mit US-amerikanischen Schauspielern
`spielfilm_kritiken(titel, rolle, kritik)`	Kritiken zu Rollen in Spielfilmen
`kurzfilm_rollen(titel, rolle, schauspieler_name, nationalitaet)`	Rollenbesetzungen in Kurzfilmen

```
film(titel,typ,regisseur,laenge);
schauspieler(schauspieler_name,nationalitaet);
spielt(titel,schauspieler_name,rolle,kritik);
```

```
film(T,Y,R,L),L>79,Y='Spielfilm'                ⊇  spielfilme(T,R,L)
film(T,Y,R,L),L<11,Y='Kurzfilm'                 ⊇  kurzfilme(T,R)
film(T,_,R,_),spielt(T,S,O,K),O='Hauprolle'     ⊇  filmkritiken(T,R,S,K)
film(T,Y,_,L),spielt(T,S,_,_),
schauspieler(S,N), N='US',Y='Spielfilm'         ⊇  us_spielfilm(T,L,S)
film(T,Y,_,_),spielt(T,_,O,K),Y='Spielfilm'     ⊇  spielfilm_kritiken(T,O,K)
film(T,Y,_,_),spielt(T,S,O,_),
schauspieler(S,N), Y='Kurzfilm'                 ⊇  kurzfilm_rollen(T,O,S,N)
```

Überblick

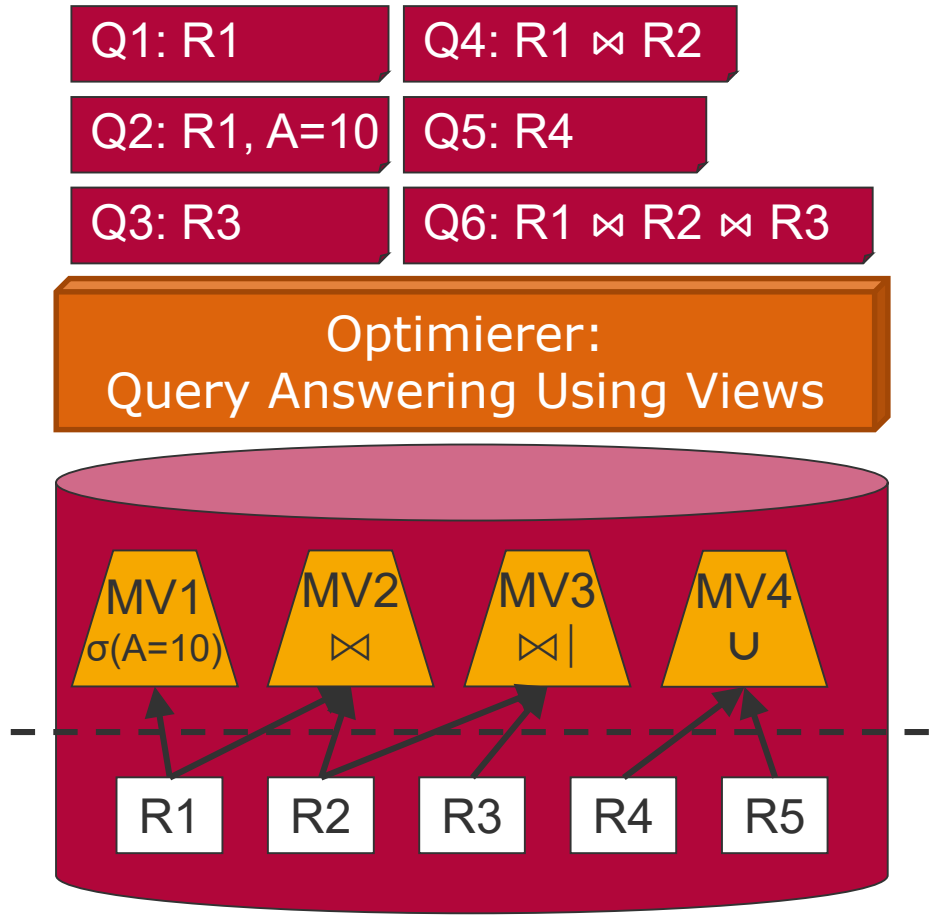
1. Motivation
2. Korrespondenzen
3. Übersicht Anfrageplanung
4. Global as View (GaV)
5. Local as View (LaV)
   - Modellierung
   - **Anwendungen**
   - Anfragebearbeitung
   - Containment
6. Global Local as View (GLaV)
7. Vergleich

LaV – Anwendungen

- Anfrageoptimierung
  - Materialisierte Sichten auf Datenbankschema
- Datawarehouse Design
  - Materialisierte Sichten auf Warehouse-Schema
- Semantisches Caching
  - Materialisierte Daten beim Client
- Datenintegration
  - Datenquellen als Sichten auf globales (Mediator) Schema

LaV Anwendung: Anfrageoptimierung

- Materialisierte Sichten (materialized views, MV) auf Datenbankschema
  - MQT: Materialized Query Table
  - AST: Advanced Summary Table
- Welche Sichten helfen bei der Beantwortung einer Datenbankanfrage durch Vorberechnung von Prädikaten?
- Probleme:
  - Es ist nicht immer besser eine MV zu verwenden (Indizes!).
  - Aktualisierung von MVs
    - Write auf MV
    - Write auf Basisrelation



LaV Anwendung: Datawarehouse Design

- Sichten auf Warehouse-Schema

- Gegeben eine query workload
  - Query workload = Menge von Anfragen plus Häufigkeiten
- Welche Sichten sollte ich materialisieren?
  - Um alle Anfragen der Workload zu beantworten

- Allgemeiner:
  - Gegeben eine query workload, welche Sichten sollte ich materialisieren um die workload optimal zu beantworten.
    - Idee: Alle Kombinationen prüfen
  - Frage: Warum ist dieses Problem eigentlich ganz einfach?
    - Besser: Kosten einbeziehen: Speicherplatz, view updates, Indizes

LaV Anwendung: Semantisches Caching

- In verteilten DBMS
- Materialisierte Daten beim Client
  - Stammen aus (komplexen) Anfragen

- Gegeben eine Anfrage
  - Welche Daten im Cache kann ich zur Beantwortung verwenden?
  - Welche Daten muss ich neu anfragen?

- Auch: Welche Sichten sollte ich vorberechnen?

LaV Anwendung: Datenintegration

- Datenquellen als Sichten auf globales (Mediator) Schema

- Fragen:
  - Wie kann ich Antworten auf eine Anfrage an das globale Schema nur mittels der Sichten beantworten?
    - Unterschied zu Anfrageoptimierung: Keine Basistabellen verfügbar.
  - Kann ich die Anfrage vollständig (extensional) beantworten?

Überblick

1. Motivation
2. Korrespondenzen
3. Übersicht Anfrageplanung
4. Global as View (GaV)
5. Local as View (LaV)
   - Modellierung
   - Anwendungen
   - **Anfragebearbeitung**
   - Containment
6. Global Local as View (GLaV)
7. Vergleich

Anfrageplanung

- Gegeben
  - Eine Anfrage q an das globale Schema
  - Lokale Schemata

- 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 q

Anfrageplan

- Wir nennen $q_1 \bowtie \ldots \bowtie q_n$ einen Anfrageplan.
- Definition
  Gegeben eine globale Anfrage q. Ein Anfrageplan p für q ist eine Anfrage der Form $q_1 \bowtie \ldots \bowtie q_n$, so dass
  - jedes q_i kann von genau einem Wrapper ausgeführt werden,
  - und jedes von p berechnete Tupel ist eine semantisch korrekte Antwort für q.

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

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
- Definition
  - Sei S ein Datenbankschema, I eine Instanz von S und q1, q2 Anfragen gegen I.
  - Sei q(I) das Ergebnis einer Anfrage q angewandt auf I.
  - Dann ist

    q1 enthalten in q2, geschrieben $q1 \subseteq q2$

    $\Leftrightarrow$

    $q1(I) \subseteq q2(I)$ für alle möglichen I
- Bemerkung
  - Der wichtige Teil ist der letzte: „für alle möglichen Instanzen I von S“
  - Die können wir natürlich nicht alle ausprobieren.

Semantische Korrektheit

- Damit können wir definieren, wann ein Plan semantisch korrekt ist (aber wir können das noch nicht testen)
- Definition
  - Sei S ein globales Schema, q eine Anfrage gegen S,
  - und p eine Verknüpfung von Views v1,…,vn gegen S, die als linke Seite von LaV Korrespondenzen definiert sind.
  - Dann ist

    p semantisch korrekt für q,

    gdw.

    $p \subseteq q$
- Bemerkung
  - Also ist die Extension von p in der von q enthalten
  - Die Extension von q gibt es natürlich nicht (nur virtuell)

Viele Anfragepläne

- Definition
  Gegeben eine globale Anfrage q. Seien p_1, …, p_n die Menge aller semantisch korrekten Anfragepläne für q. Dann ist das Ergebnis von q definiert als

$$result(q) = \bigcup_{i=1..n} result(p_i)$$

- Bemerkungen
  - Der UNION Operator entfernt Duplikate
    - Dahinter verbirgt sich das Problem der Ergebnisintegration.
  - Wie das Ergebnis berechnet wird, ist Sache der Anfrageoptimierung.
    - Pläne können sich in Teilanfragen überlappen.
  - Das Ergebnis von q hängt ab von den definierten Korrespondenzen.

Answering Queries 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

Felix Naumann
Information Integration
Winter 2019/20

LaV – Beispiel

Globales Schema

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

Quelle 1: Alle Datenbankveranstalt.

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

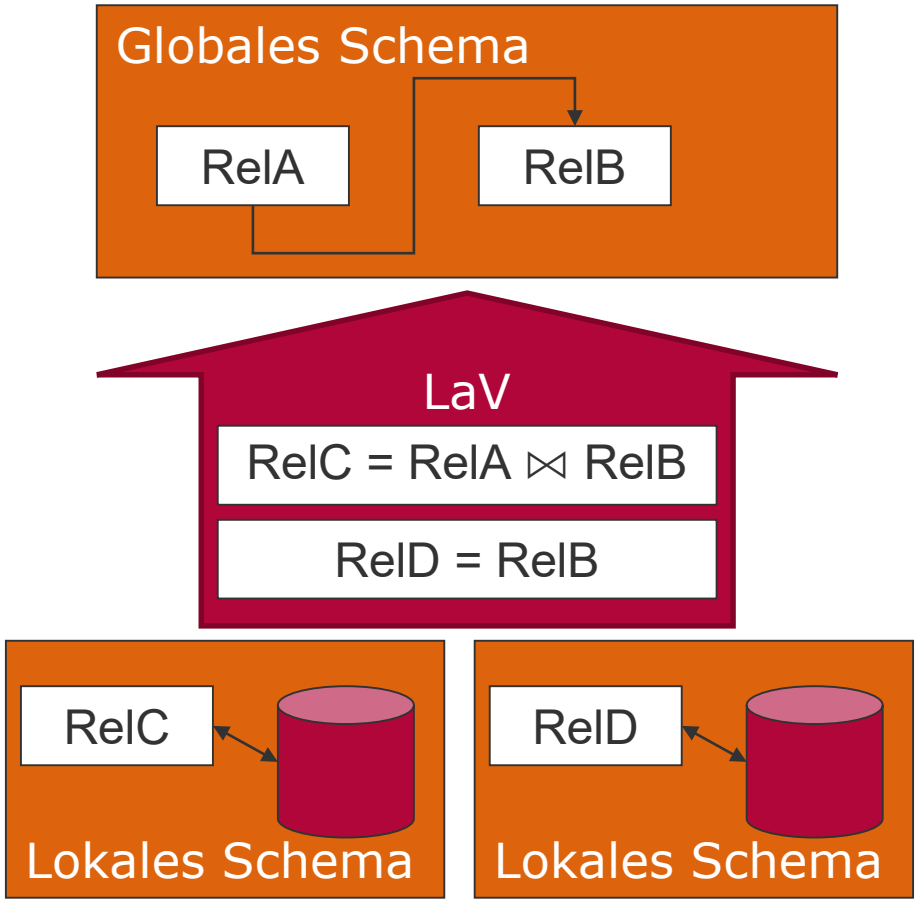
Umgeschriebene Anfrage

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

Frage: Warum nicht Quelle 2 einbeziehen?

Antwort: Weil Quelle 1 ja schon ALLE DB-Veranstaltungen liefert.

LaV Visualisierung (OWA)



Nutzer-Anfrage

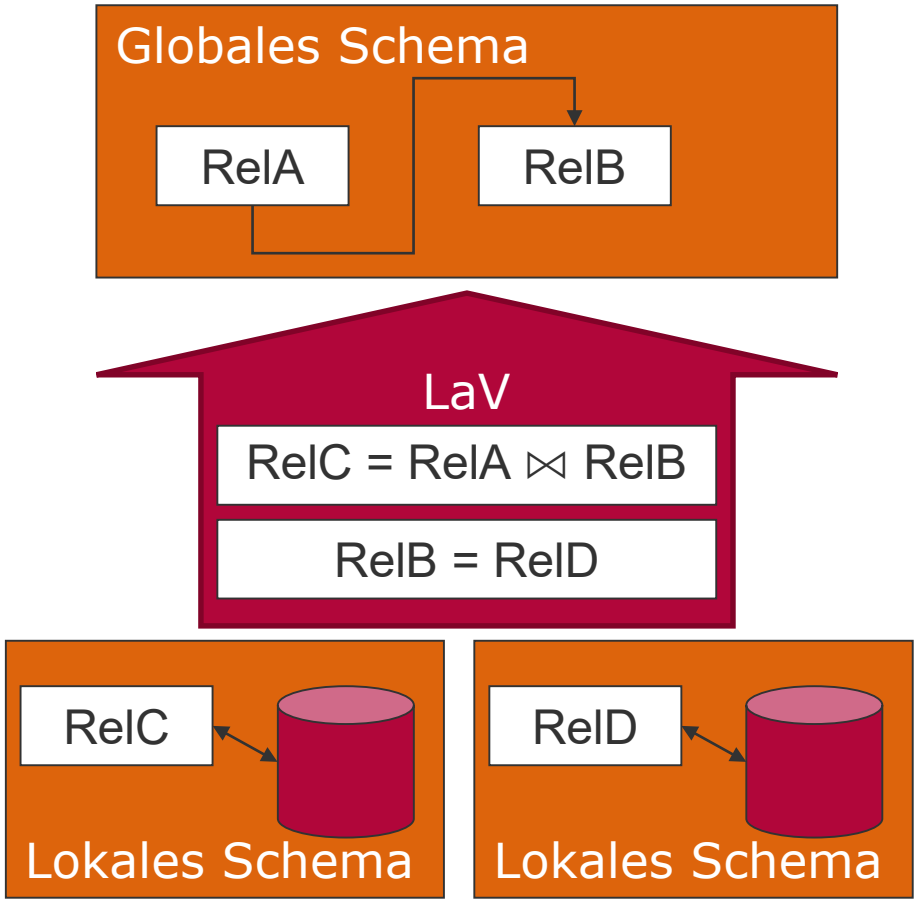
```
SELECT  ???
FROM    RelB
WHERE   ???
```

Anfrage-umschreibung

Umgeschriebene Anfrage

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

LaV Visualisierung (CWA)



Nutzer-Anfrage

```
SELECT ???
FROM   RelA NATURAL JOIN RelB
WHERE  ???
```

Anfrage-umschreibung

Umgeschriebene Anfrage

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

LaV – Beispiel

Ausschnitt Globales Schema

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

Quelle 1: Alle Datenbankveranstalt.

```
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“
```

Globale Anfrage

```
SELECT titel, kurs_id
FROM Kurs K
WHERE L.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_%“
```

Umgeschriebene Anfrage

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

Frage:
Warum hier doch Quelle 2 einbeziehen?

LaV – Beispiel Vergleich

Globale Anfrage

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

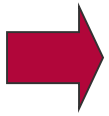
Umgeschriebene Anfrage

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

} Vollständige Antwort (CWA)

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
```

} Maximale Antwort

Frage:
Was fehlt?

Felix Naumann
Information Integration
Winter 2019/20

CWA / OWA

- Closed World Assumption (CWA)
  - Vereinigung aller Daten der Basisrelationen entspricht der Menge aller relevanten Daten.
  - Beispiele: Data Warehouse (und traditionelle DBMS)

- Open World Assumption (OWA)
  - Vereinigung aller Daten der Datenquellen ist eine Teilmenge aller relevanten Daten.
  - Probleme
    - Inhalt der globalen Relation nicht fest: Anfrageergebnisse können sich ändern
    - Definition der Vollständigkeit der Ergebnisse: Welchen Anteil an der *world* hat das Ergebnis?
    - Negation in Anfragen

CWA / OWA – Beispiel

- Relation R(A,B)
- View 1
  - CREATE VIEW V1 AS
    SELECT A FROM R
  - Extension: a
- View 2
  - CREATE VIEW V2 AS
    SELECT B FROM R
  - Extension: b
- Anfrage: SELECT * FROM R
  - CWA: (a,b) muss in der Extension von R sein.
    - Wenn es irgend ein anderes (a, x) gäbe müsste x in der Extension von V2 sein
  - OWA: (a,b) muss nicht in der Extension von R sein.

R = (a,b)

R = (a,b)
oder
R = (a,x)
(y,b)
o.ä.

Felix Naumann
Information Integration
Winter 2019/20

LaV – Anfragebearbeitung

- Gegeben: Anfrage Q und Sichten V_1, ..., V_n
- Gesucht: Umgeschriebene Anfrage Q‘, die
  - bei Optimierung: äquivalent ist (Q = Q‘).
  - bei Integration: maximal enthalten ist.
    - D.h. $Q \supseteq Q'$ und
    - es existiert kein Q‘‘ mit $Q \supseteq Q'' \supset Q'$

- Problem:
  - Wie definiert und testet man Äquivalenz und *maximal containment*?

Überblick

1. Motivation
2. Korrespondenzen
3. Übersicht Anfrageplanung
4. Global as View (GaV)
5. Local as View (LaV)
   - Modellierung
   - Anwendungen
   - Anfragebearbeitung
   - **Containment**
6. Global Local as View (GLaV)
7. Vergleich

LaV – Anfrageumschreibungen

- Gegeben
  - Anfrage Q (query)
  - Sicht V (view) bzw. Plan
- Fragen
  - Ist Ergebnis von V identisch dem Ergebnis von Q?
  - Kurz: Ist V äquivalent zu Q, $V = Q$?

- Rückführung auf „Enthalten sein“ (*containment*)
  - Ist das Ergebnis von V in Q enthalten?
  - Kurz: Ist V in Q enthalten, $V \subseteq Q$?
- Denn
  - $V \subseteq Q$, $Q \subseteq V \Rightarrow V = Q$

Felix Naumann
Information Integration
Winter 2019/20

LaV – Anfrageumschreibungen (WdH)

- 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,
    $\Leftrightarrow$ Q'(D) $\subseteq$ Q(D) für jedes mögliche D.

```
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“
```

LaV – Beispiele

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

≠

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

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

≠

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

Felix Naumann
Information Integration
Winter 2019/20

LaV – Beispiele

```
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“
```

$\subseteq$

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

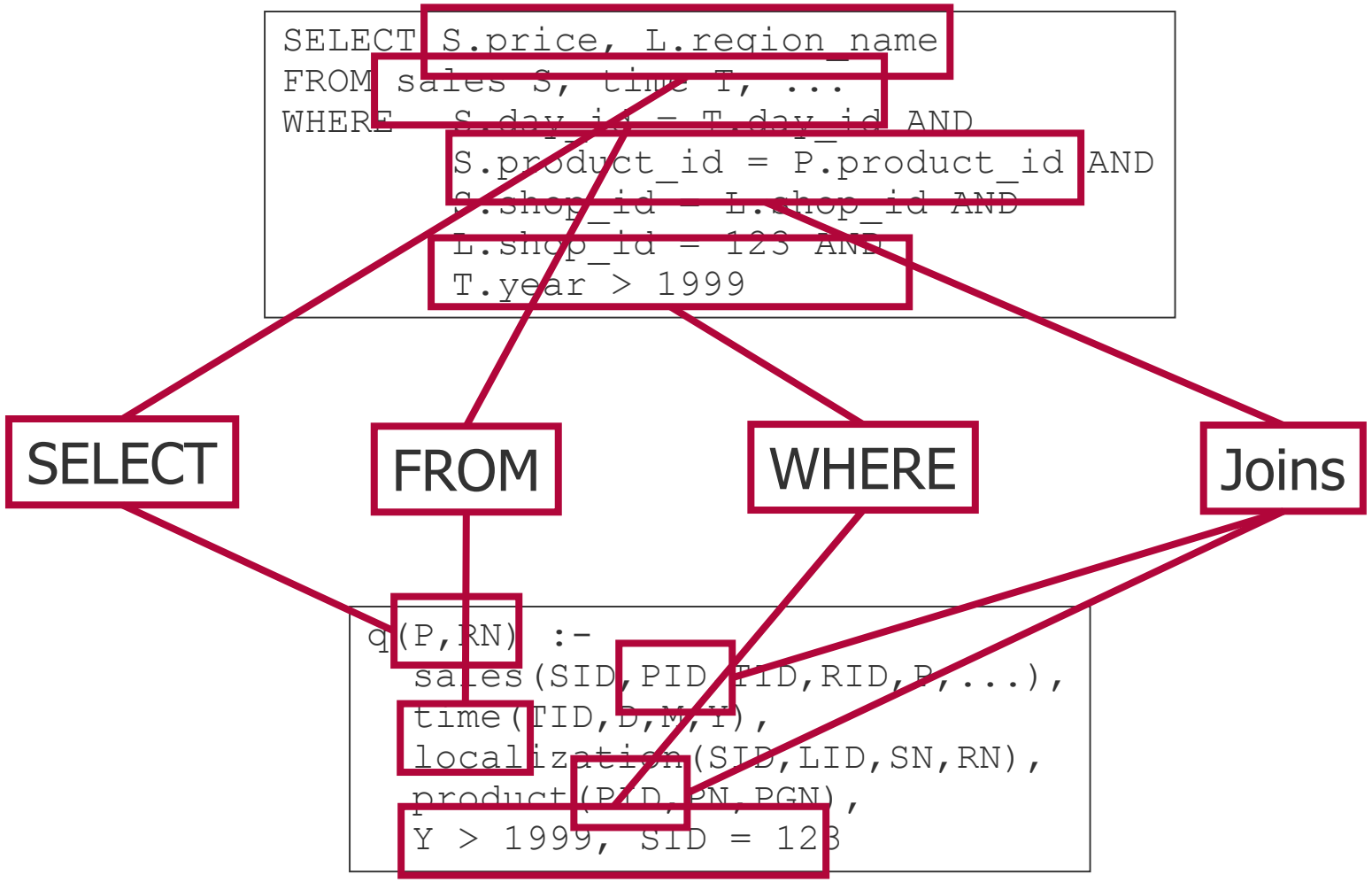
- Prüfung von containment durch Prüfung aller möglichen Datenbanken?
  - Zu komplex!
- Prüfung von containment durch Existenz eines *containment mapping*.
  - NP-vollständig in |Q|+|Q‘| nach [CM77]
  - Mehrere Algorithmen

Datalog Notation

- Im Folgenden: Nur konjunktive Anfragen
  - Nur Equijoins und Selektionsbedingungen mit =,<,> zwischen Attribut und Konstante
  - Kein NOT, EXISTS, GROUP BY, ≠, X>Y, ...
- Schreibweise: Datalog / Prolog
  - SELECT Klausel
    - Regelkopf, Exportierte Attribute
  - FROM Klausel
    - Relationen werden zu Prädikaten
  - WHERE Klausel
    - Joins werden durch gleiche Attributnamen angezeigt
    - Bedingungen werden explizit angegeben

Felix Naumann
Information Integration
Winter 2019/20

SQL – Datalog



Felix Naumann
Information Integration
Winter 2019/20

Definition 2.2

Sei V eine Menge von Variablensymbolen und C eine Menge von Konstanten. Eine *konjunktive Datalog-Anfrage* q ist eine Anfrage der Form:

$$q(v_1, v_2, \ldots, v_n) \quad :- \quad r_1(w_{1,1}, \ldots, w_{1,n_1}), r_2(w_{2,1}, \ldots w_{2,n_2}), \ldots, r_m(w_{m,1}, \ldots, w_{m,n_m}), k_1, \ldots, k_l;$$

mit extensionalen Prädikaten $r_1, r_2, \ldots r_m$, $v_i \in V$, $w_{i,j} \in V \cup C$ und $\forall v \in V : \exists i, j : w_{i,j} = v$ und $\forall c \in C : \exists i, j : w_{i,j} = c$. Alle k_i haben für beliebige $v_1, v_2 \in V$ und $c \in C$ die Form $v_1 < c$, $v_1 > c$, $v_1 = c$ oder $v_1 = v_2$. Dann ist:

- $head(q) = q(v_1, v_2, \ldots, v_n)$ der Kopf von q,
- $body(q) = r_1(w_{1,1}, \ldots), r_2(w_{2,1}, \ldots), \ldots, r_m(w_{m,1}, \ldots)$ der Rumpf von q,
- $exp(q) = \{v_1, v_2, \ldots, v_n\}$ die Menge der *exportierten Variablen* von q,
- $var(q) = V$ die Menge aller Variablen von q,
- $const(q) = C$ die Menge aller Konstanten von q,
- $sym(q) = C \cup V$ die Menge aller Symbole von q,
- $r_1, r_2, \ldots r_m$ sind die *Literale* von q, und
- $cond(q) = k_1, \ldots k_l$ sind die Bedingungen von q. ■

Query Containment

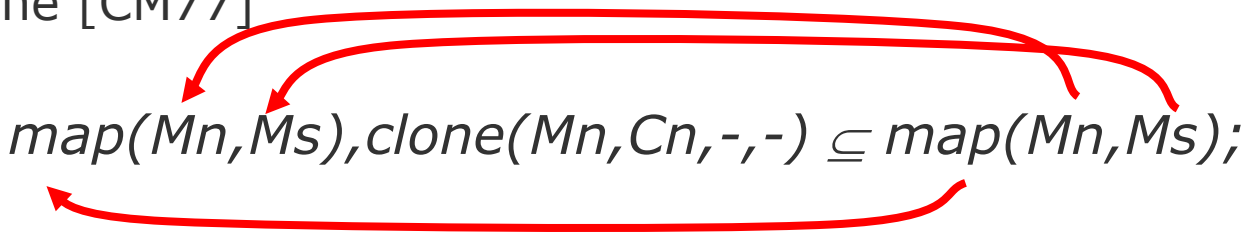
- A query *p* is contained in a query *u* ($p \subseteq u$) iff all tuples computed by *p* are also computed by *u* for every DB.

- Beispiele (Abkürzende Schreibweise: Regelkopf weggelassen)
  - map(Mn,Ms) $\subseteq$ map(Mn,Ms);
  - map(Mn,Ms), Mn = 'HGM' $\subseteq$ map(Mn,Ms);
  - map(Mn,Ms), Ms<500 $\subseteq$ map(Mn,Ms);
  - map(Mn,Ms), clone(Mn,Cn,-) $\subseteq$ map(Mn,Ms);
  - clone(Mn,Cn,Cs), clone(Mn,Cn,Cs) $\subseteq$ clone(Mn,Cn,Cs);

Herleitung von Query Containment

- p ⊆ u ⇔ es existiert ein *containment mapping* von u nach p

- Containment mapping:
  - h: sym(u) → sym(p) (Abbildung der Symbole)
  - CM1: Jede Konstante in u wird auf die gleiche Konstante in p abgebildet.
  - CM2: Jede exportierte Variable in u wird auf eine exportierte Variable in p abgebildet.
  - CM3: Jedes Literal (Relation) in u wird auf mindestens ein Literal in p abgebildet
  - CM4: Die Bedingungen von p implizieren die Bedingungen von u
- Beweis: siehe [CM77]



Finden von Containment Mappings

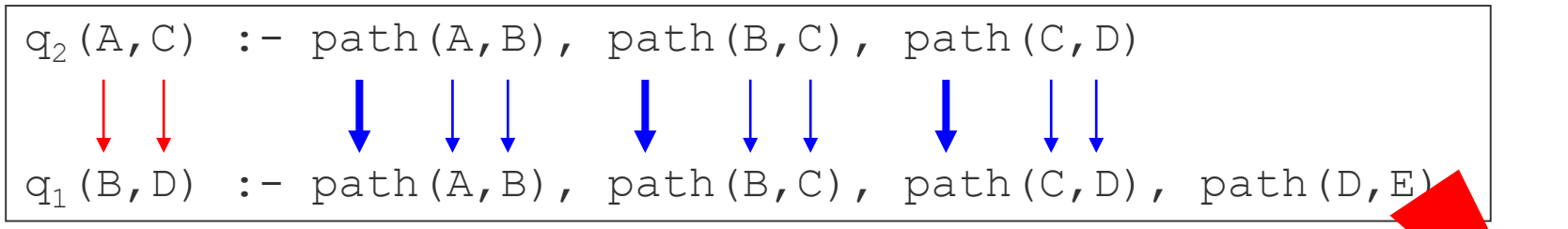
- Ursprüngliche Motivation: Anfrageminimierung
  - Vorstufe zur Optimierung
- Problem ist NP vollständig
  - Suchraum ist exponentiell in der Anzahl der Literale
  - Beweis: Reduktion auf „Exakt Cover“
- Also: Alles ausprobieren
  - Aufbau eines Suchbaums
    - Jede Ebene entspricht einem Literal
    - Auffächerung nach möglichen CMs
- Algorithmus
  - Nicht hier
  - Siehe Lehrbuch

Weitere Containment Beispiele

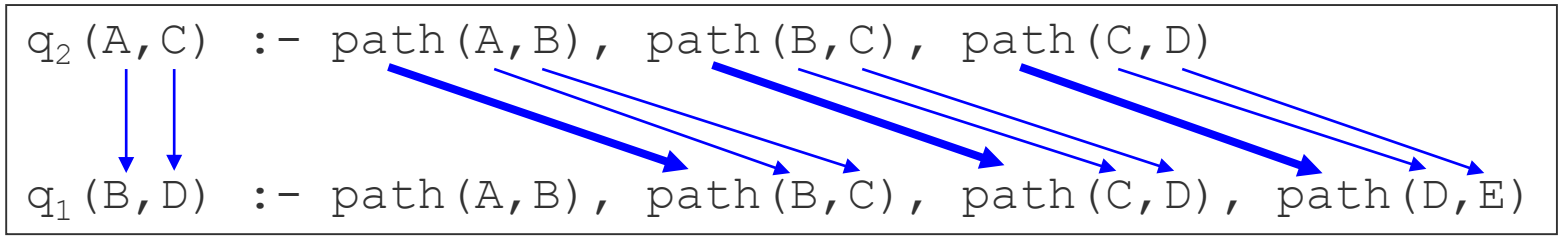
- product(PID, PN, PGID, PGN), PGN=`Wasser´ ⊆ product(PID, PN, PGID, PGN)
- product(PID, PN, PGID, PGN) ⊈ product(PID, PN, PGID, PGN), PGN=`Wasser´
- product(PID, PN, PGID, PGN) ⊈ localization(SID, SN, RID, RN)
- sales(SID, PID, …,P,…), P>80, P<150 ⊈ sales(SID, PID, …,P,…), P>100, P<150
- sales(SID, PID, …,P,…), product(PID,PN,…)
  ⊆ sales(SID, PID, …,P,…) bei Projektion auf sales-Attribute

Beispiel

$q_1 \subseteq q_2$?

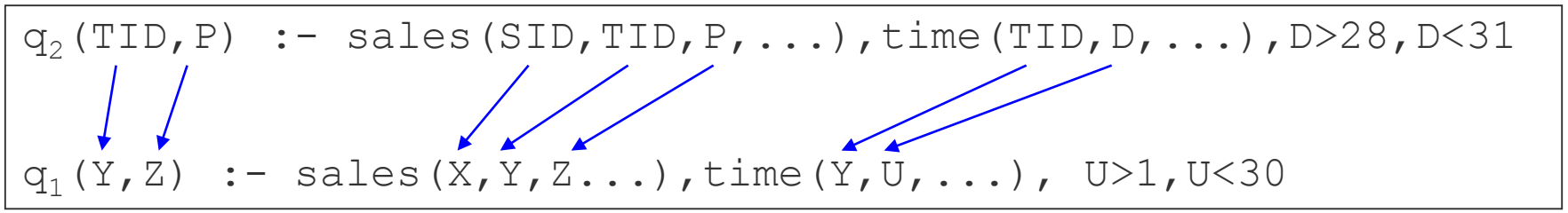


Mapping: A→A, B→B, C→C, D→D



Mapping: A→B, B→C, C→D, D→E

Beispiel



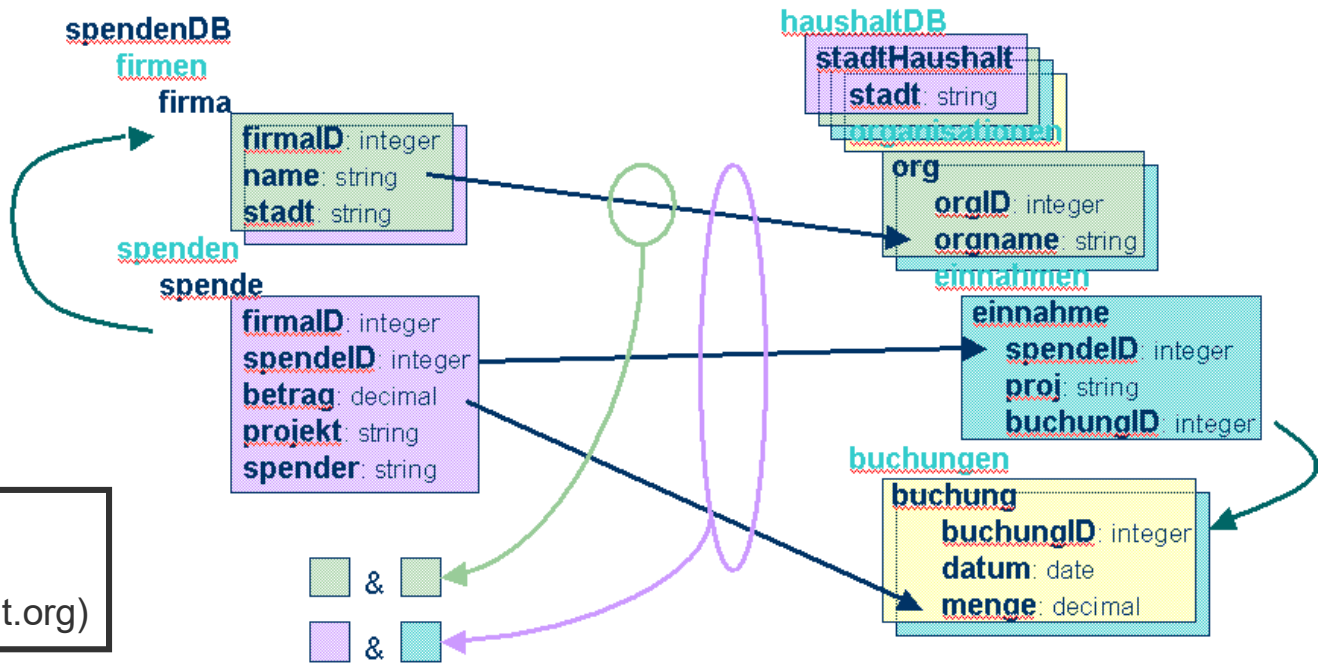
CM: SID→X, TID→Y, P→Z, D→U
h(D)=U

Aber: U>1∧U<30 !→ h(D)>28∧h(D)<31

$$q_1 \not\subseteq q_2$$

Erzeugung der Anfragen

Beobachtung:
Interpretation als
containment



$\pi_{name}(spendenDB.firmen) \subseteq \pi_{orgname}(haushaltDB.stadtHaushalt.org)$

$\pi_{name,spendeID,betrag}(spendenDB.firmen \bowtie spendenDB.spenden) \subseteq \pi_{orgname,spendeID,menge}(haushaltDB.stadtHaushalt.org.einnahmen \bowtie haushaltDB.stadtHaushalt.buchungen)$

Felix Naumann
Information Integration
Winter 2019/20

Anfrageumschreibung

- Umschreibung von globaler Anfrage auf Menge an lokalen Anfragen

- Prinzipiell:
  - Prüfe jede Kombination an Sichten auf Containment
  - Unendlich viele Kombinationen, da eine Sicht auch mehrfach in eine Umschreibung eingehen kann.

- Verbesserungen:
  - Satz: Umschreibung mit maximal so vielen Sichten wie Relationen in Anfrage (ohne range-Prädikate).
  - Geschickte Vorauswahl der Sichten: Nutzbarkeit
    - Bucket Algorithmus (nächster Foliensatz)

Überblick

1. Motivation
2. Korrespondenzen
3. Übersicht Anfrageplanung
4. Global as View (GaV)
5. Local as View (LaV)
   - Modellierung
   - Anwendungen
   - Anfragebearbeitung
   - Containment
6. **Global Local as View (GLaV)**
7. Vergleich

Global-Local-as-View (GLAV)

- Kombination GaV und LaV
  - 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
- Auch „BaV“: Both-as-View

Anfrageplanung

- GaV:
  - View unfolding
  - Standardtechniken aus relationalen Datenbanken
- LaV
  - Query Containment und Answering queries using views
  - Mehrere Algorithmen, auch andere Anwendungen
  - Schwierige Planung
- GLaV
  - Erst Anfrageplanung mit den Sichten auf das globale Schema
    - Die linken Seiten von Korrespondenzen
  - Dann Unfolding mit den Sichten auf die lokalen Schemata
    - Die rechten Seiten von Korrespondenzen
- Dann verteilte Optimierung

Überblick

1. Motivation
2. Korrespondenzen
3. Übersicht Anfrageplanung
4. Global as View (GaV)
5. Local as View (LaV)
   - Modellierung
   - Anwendungen
   - Anfragebearbeitung
   - Containment
6. Global Local as View (GLaV)
7. **Vergleich**

Vergleich GaV / LaV

Modellierung

- GaV
  - Jede globale Relation definiert als Sicht auf eine oder mehr Relationen aus einer oder mehr Quellen.
  - Meist UNION über mehrere Quellen
  - Nebenbedingungen auf lokalen Quellen können nicht modelliert werden.
- LaV
  - Globale Relationen werden durch mehrere Sichten definiert.
  - Definition oft nur in Kombination mit anderen globalen Relationen
  - Nebenbedingungen auf globale Relationen können nicht definiert werden.
- GLaV
  - Globale und lokale Nebenbedingungen sind möglich
  - Konzepte können lokal oder global eingeschränkt werden

Vergleich GaV / LaV

- Anfragebearbeitung
  - GaV:
    - Anfrageumschreibung: Einfaches View unfolding
    - Eine große, umgeschriebene Anfrage
    - Interessante Optimierungsprobleme
  - LaV
    - Anfrageumschreibung: Answering queries using views
    - UNION über viele mögliche umgeschriebene Anfragen
    - Mehrere Algorithmen
    - Viele Anwendungen
    - Noch interessantere Optimierungsprobleme
- Flexibilität
  - GaV: Views setzen Relationen mehrerer Quellen in Beziehung
  - LaV: Jede View bezieht sich nur auf eine Quelle

Literatur

Gute Zusammenfassung für LaV und weiterführende Literatur:

- [Levy01] Alon Y. Halevy: Answering queries using views: A survey, in VLDB Journal 10: 270-294, 2001.

Eher theoretisch

- [Ull00] Jeffrey D. Ullman: Information Integration Using Logical Views. TCS 2000: 189-210
- [Hull97] Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. Richard Hull. PODS 1997 tutorial
- [Ull97] Jeffrey D. Ullman: Information Integration Using Logical Views. ICDT 1997: 19-40
- [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, pages 77-90, Boulder, Colorado, 2-4 May 1977.
- [LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, Divesh Srivastava: Answering Queries Using Views. PODS 1995: 95-104