

Informationsintegration
SchemaSQL

31.5.2007
Felix Naumann

Überblick

2

- Wiederholung
 - Strukturelle Heterogenität
 - Multidatenbanken
- SchemaSQL
 - Basis-Syntax
 - Aggregation
 - Umstrukturierung
 - Architektur und Implementierung



Strukturelle Heterogenität

3

- Datenmodell-Heterogenität
 - Relationales Modell
 - XML Modell
 - OO Modell
 - Hierarchisches Modell
- Schematische Heterogenität
 - Integritätsbedingungen, Schlüssel, Fremdschlüssel, etc.
 - Struktur (Attribut vs. Relation etc.)

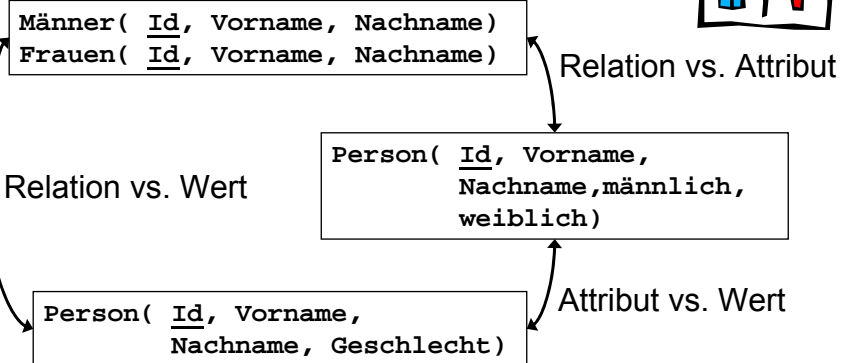
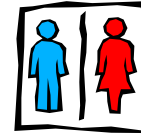
Schematische Heterogenität

4

- Modellierung
 - Relation vs. Attribut
 - Attribut vs. Wert
 - Relation vs. Wert
 - Benennung
 - Relationen
 - Attribute
 - Normalisiert vs. Denormalisiert
 - Geschachtelt vs. Fremdschlüssel
- } SQL
- } SchemaSQL

Schematische Heterogenität

5



Felix Naumann | VL Informationsintegration | SS 2007

Schematische Heterogenität - Lösungen

6

- Problem
 - Einheitlich auf beide Schemas zugreifen
 - Auf Schemaebene: Schema Mapping und Schema-Sprachen
 - Auf Datenebene: Virtuelle Integration
 - Beide Schemas in eine gemeinsames neues Schema integrieren
 - Auf Schemaebene: Schemaintegration
 - Auf Datenebene: Materialisierte Integration
- Für die materialisierte Integration
 - Schemaintegration
 - ETL
- Für die virtuelle Integration
 - Schema-Sprachen
 - Z.B. SchemaSQL, MSOL, CPL
 - Lose Kopplung, Multidatenbanken
 - Schema Mapping
 - Z.B. Clio, RONDO, u.a.
 - Enge Kopplung, föderierte Datenbanken

Felix Naumann | VL Informationsintegration | SS 2007

Schematische Heterogenität – Lösungen

7

SchemaSQL [LSS96, LSS99, LSS01]

- Erweiterung von SQL
- Daten und Metadaten werden gleich behandelt
- Umstrukturierungen innerhalb der Anfrage
- Dynamische Sicht-Definition
- Horizontale Aggregation

```
SELECT RelA
FROM uniA->RelA, uniA::RelA A, uniB::grundgehalt B
WHERE RelA = B.institut
AND A.Kategorie = „Student“
AND A.grundgehalt > B.Student
```

High-order Join

Felix Naumann | VL Informationsintegration | SS 2007

Überblick

8

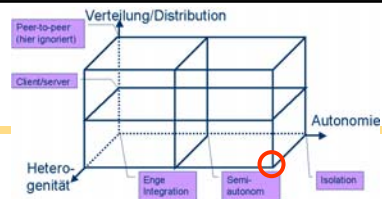
- Wiederholung
 - Strukturelle Heterogenität
 - Multidatenbanken
- SchemaSQL
 - Basis-Syntax
 - Aggregation
 - Umstrukturierung
 - Architektur und Implementierung



Felix Naumann | VL Informationsintegration | SS 2007

Aut2, Dist0, Het1

9



- Multidatenbanksystem (MDBMS)
- Volle Autonomie
 - Keine bekannte Kooperation
 - Keine Kommunikation untereinander
- Keine Interoperation untereinander möglich
- Integration nur in neuer, integrierender Komponente.
- Z.B. DBMS und WWW Server auf einer Maschine
 - Nicht zur Interoperation entwickelt
 - DBMS „spricht“ kein http, WWW „spricht“ kein SQL

Felix Naumann | VL Informationsintegration | SS 2007

Enge vs. lose Kopplung



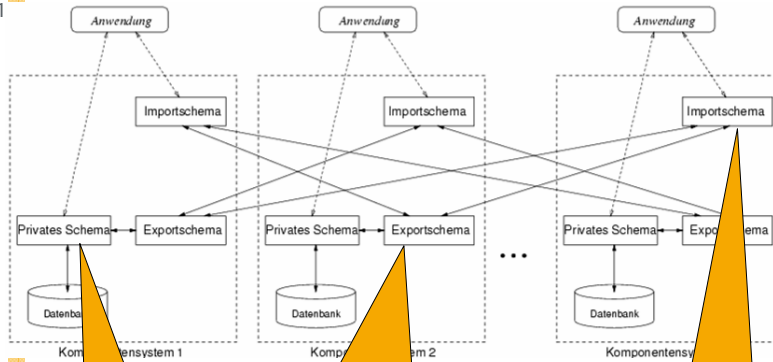
10

- Enge Kopplung
 - Festes, integriertes/föderiertes Schema
 - Modelliert mit Korrespondenzen
 - Feste Anfragesprache
- Lose Kopplung
 - Kein festes Schema
 - Nutzer müssen Semantik der Quellen kennen
 - Integrierte Sichten helfen
 - Feste Anfragesprache
 - SchemaSQL [LSS01]
 - (Multidatabase query language (MDBQL) [LMR90])

Felix Naumann | VL Informationsintegration | SS 2007

Import-/Export-Schema-Architektur nach [HM85]

11



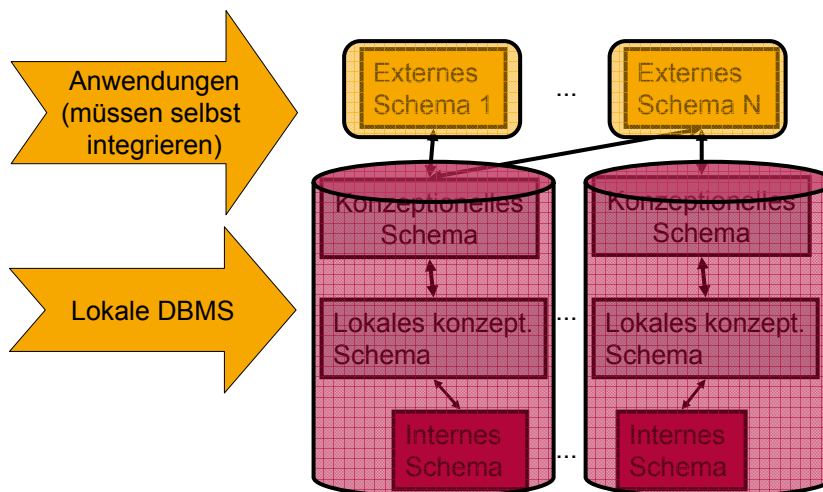
= lokales konzeptionelles Schema

Idee: Nur Teilmenge des lokalen konzeptionellen Schemas wird der Föderation zur Verfügung gestellt.

Idee: Nur Teilmengen der Exportschemas sollen verwendet werden.

4-Schichten Architektur

12



Multidatenbanksprachen

13

Anforderungen

- Schemaunabhängigkeit
 - Struktur darf nicht Ausdrucksfähigkeit beeinflussen.
- Umstrukturierung
 - Anfrageergebnisse müssen neue Struktur erhalten können.
- Verständlichkeit und doch Ausdrucksfähigkeit
- Abwärtskompatibilität mit SQL
- Implementierbar
 - Ohne Veränderung des DBMS
 - Bzw. mit nur minimalen Veränderungen des DBMS

Felix Naumann | VL Informationsintegration | SS 2007

SchemaSQL [LSS01]

14

Features

- Erweiterung von SQL
- Daten und Metadaten werden gleich behandelt
- Umstrukturierungen innerhalb der Anfrage
 - Daten zu Metadaten und umgekehrt
 - Daten: Tupel und Attributwerte
 - Metadaten: Attributnamen, Relationennamen, Datenbanknamen
- Dynamische Sicht-Definition
 - Struktur des Ergebnisses abhängig von aktuellem Zustand der Datenbank
- Horizontale Aggregation
 - Über mehrere Spalten hinweg
- Unterstützung für Multidatenbanken

Felix Naumann | VL Informationsintegration | SS 2007

Überblick

15

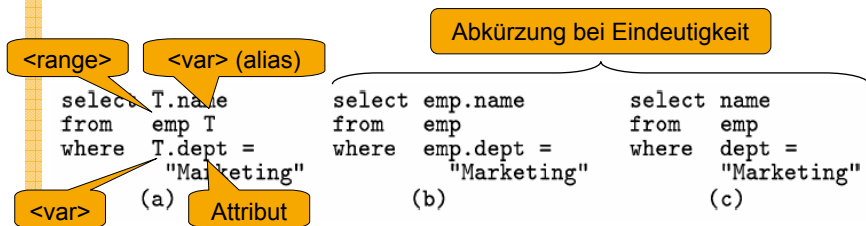
- Wiederholung
 - Strukturelle Heterogenität
 - Multidatenbanken
- SchemaSQL
 - Basis-Syntax
 - Aggregation
 - Umstrukturierung
 - Architektur und Implementierung



SchemaSQL – Syntax

16

- Erweiterung von SQL
- Standard SQL
 - Variablendeklaration in FROM Klausel
 - Variablenverwendung in SELECT und WHERE Klauseln



SchemaSQL – Syntax

17

- Erweiterung von SQL
- Anforderungen
 - Unterscheidung von mehreren DBs, jeweils mit mehreren Relationen
 - Metadaten: Variablendeklarationen nicht nur für Tupelmengen
 - Aggregation nicht nur vertikal über ein Attribut
 - Sondern auch horizontal

SchemaSQL – Syntax

18

- Variablendeklaration über
 1. Datenbanknamen
 2. Relationen in einer Datenbank
 3. Attributnamen einer Relation
 4. Tupel einer Relation
 5. Werte eines Attributs
- Deklaration durch <range> <var>
- Wichtiger Unterschied: Geschachtelte Deklarationen
 - Alle Tupel aller Relationen einer Datenbank

Frage: Welcher der 5 ist Standard SQL?

Standard SQL

SchemaSQL – Syntax

19

- Variablendeklaration: `<range> <var>`
- `<range>`
 - `->` Iteration über alle Datenbanken
 - `db->` Iteration über alle Relationen in db
 - `db::rel->` Iteration über alle Attribute in rel (in db)
 - `db::rel` Iteration über alle Tupel in rel (in db)
 - `db::rel.attr` Iteration über alle Werte von Attribut attr (in rel und db)
- Präfixe können bei Eindeutigkeit weggelassen werden.
- `<var>`
 - Konstante (ein beliebiger Name)
 - Variable, falls wiederum als Variablendeklaration definiert

Felix Naumann | VL Informationsintegration | SS 2007

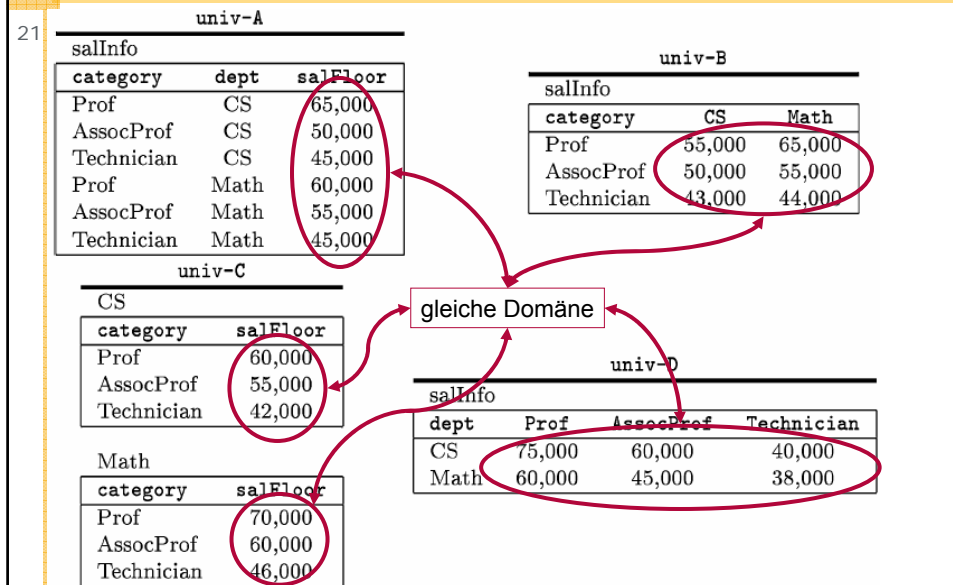
SchemaSQL – Beispiel

20

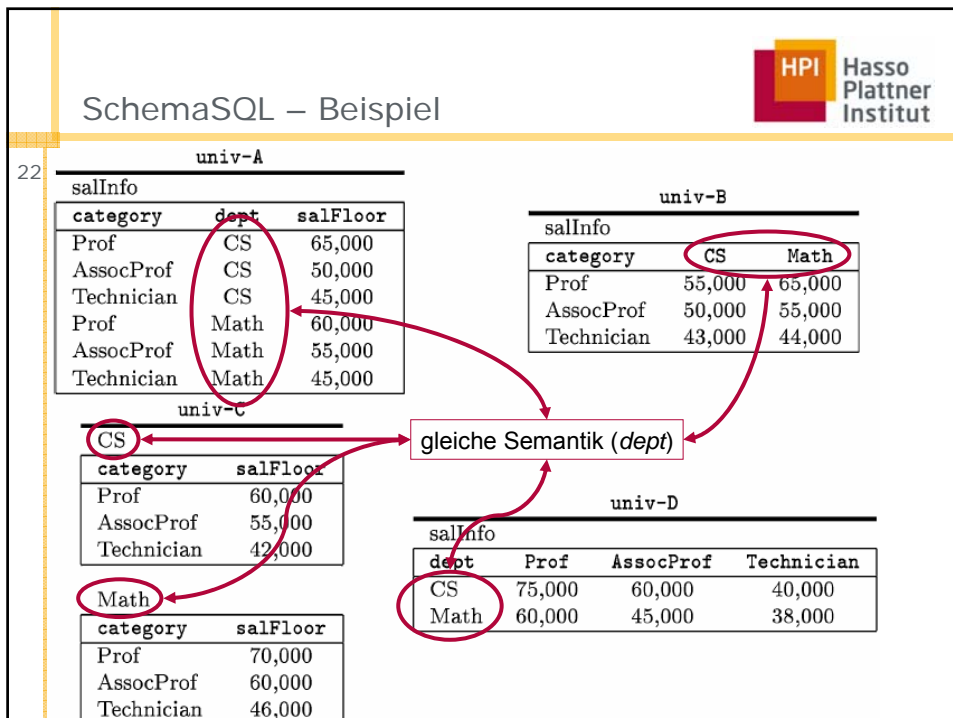
- Multidatenbank über mehrere Universitäten
 - univ-A, univ-B, univ-C, univ-D
- Information über Angestellte
 - Kategorie (*category*)
 - Gehalt (*salInfo*, *salFloor*)
 - Abteilung (*dept*)

Felix Naumann | VL Informationsintegration | SS 2007

SchemaSQL – Beispiel



SchemaSQL – Beispiel



SchemaSQL – Beispiel

23

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

univ-B		
salInfo		
category	CS	Math
Prof	55,000	65,000
AssocProf	50,000	55,000
Technician	43,000	44,000

univ-C	
CS	
category	salFloor
Prof	60,000
AssocProf	55,000
Technician	42,000
Math	
category	salFloor
Prof	70,000
AssocProf	60,000
Technician	46,000

univ-D			
salInfo			
dept	Prof	AssocProf	Technician
CS	75,000	60,000	40,000
Math	60,000	45,000	38,000

gleiche Semantik (category)

SchemaSQL – Anfragen

24

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

univ-B		
salInfo		
category	CS	Math
Prof	55,000	65,000
AssocProf	50,000	55,000
Technician	43,000	44,000

Gesucht

- Alle Abteilungen in **univ-A**, die Technikern mehr zahlen als in gleichen Abteilungen von **univ-B**

Anforderungen

- Selektionen jeweils auf `Technician``
- Vergleich der Gehälter
- Join zwischen beiden Tabellen
 - Verschiedene DBs
 - Über welches Attribut?

Felix Naumann | VL Informationsintegration | SS 2007

25

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

univ-B		
salInfo		
category	CS	Math
Prof	55,000	65,000
AssocProf	50,000	55,000
Technician	43,000	44,000

Gesucht

- Alle Abteilungen in univ-A, die Technikern mehr zahlen als in gleichen Abteilungen von univ-B

SchemaSQL Anfrage

```

■ SELECT  A.dept
FROM      univ-A::salInfo A,
          univ-B::salInfo B,
          univ-B::salInfo-> AttB
WHERE     AttB <> `category`
AND       A.dept = AttB
AND       A.category = `Technician`
AND       B.category = `Technician`
AND       A.salFloor > B.AttB
    
```

Alle Attributnamen

Join zwischen Attributnamen und Spaltenwerten

Frage: Ist dies im Sinne GaV oder LaV?

26

```

SELECT A.dept
FROM  univ-A::salInfo A,
      univ-B::salInfo B,
      univ-B::salInfo-> AttB
WHERE AttB <> `category`
AND   A.dept = AttB
AND   A.category = `Technician`
AND   B.category = `Technician`
AND   A.salFloor > B.AttB
    
```

-> alle Datenbanknamen
db-> alle Relationen in db
db::rel-> alle Attribute in rel (in db)
db::rel alle Tupel in rel (in db)
db::rel.attr alle Werte von Attribut attr

SchemaSQL – Anfragen

27

univ-C

CS	
category	salFloor
Prof	60,000
AssocProf	55,000
Technician	42,000

Math

category	salFloor
Prof	70,000
AssocProf	60,000
Technician	46,000

univ-D

salInfo			
dept	Prof	AssocProf	Technician
CS	75,000	60,000	40,000
Math	60,000	45,000	38,000

Gesucht (wie eben)

- Alle Abteilungen in **univ-C**, die Technikern mehr zahlen als in gleichen Abteilungen von **univ-D**

Anforderungen

- Selektionen jeweils auf `Technician``
 - Aber auch auf Attributebene
- Vergleich der Gehälter
- Join zwischen beiden Tabellen
 - Verschiedene DBs
 - Über welches Attribut?

SchemaSQL – Anfragen

28

univ-C

CS	
category	salFloor
Prof	60,000
AssocProf	55,000
Technician	42,000

Math

category	salFloor
Prof	70,000
AssocProf	60,000

univ-D

salInfo			
dept	Prof	AssocProf	Technician
CS	75,000	60,000	40,000
Math	60,000	45,000	38,000

Gesucht (wie eben)

- Alle Abteilungen in **univ-C**, die Technikern mehr zahlen als in gleichen Abteilungen von **univ-D**

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 Variable

Iteration über Tupel beider Tabellen in univ-C

Join zwischen Relationennamen und Spaltenwerten

SchemaSQL – Anfragen

29

```
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
```

Alle Relationen
in univ-C

Alle Tupel in allen
Relationen

-> alle Datenbanknamen
db-> alle Relationen in db
db::rel-> alle Attribute in rel (in db)
db::rel alle Tupel in rel (in db)
db::rel.attr alle Werte von Attribut attr

Felix Naumann | VL Informationsintegration | SS 2007

Überblick

30

- Wiederholung
 - Strukturelle Heterogenität
 - Multidatenbanken
- SchemaSQL
 - Basis-Syntax
 - Aggregation
 - Umstrukturierung
 - Architektur und Implementierung



Felix Naumann | VL Informationsintegration | SS 2007

SchemaSQL – Herkömmliche Aggregation

31

- **AVG, COUNT, SUM, MIN, MAX, (STDDEV, VARIANCE)**
- **SELECT AVG(Budget) FROM projekt**
- **SELECT SUM(p.Budget), MAX(p.Budget)**
FROM mitarbeiter m, projekt p
WHERE m.p_id = p.p_id
AND m.Nachname = "Schmidt"
- **SELECT COUNT(*)**
FROM mitarbeiter
 - Aggregation ist vertikal: Alle Werte einer Spalte werden zusammengefasst
- **SELECT m.name, SUM(p.Budget), MAX(p.Budget)**
FROM mitarbeiter m, projekt p
WHERE m.p_id = p.p_id
GROUP BY m.id
 - Aggregation ist vertikal: Teilmengen (Gruppen) von Werten einer Spalte werden zusammengefasst

Felix Naumann | VL Informationsintegration | SS 2007

SchemaSQL – Aggregation

32

Gesucht

- Durchschnittliches Gehalt aller Gruppierungen über alle Abteilungen hinweg.

Anforderungen

- Durchschnittsbildung über alle Werte zweier Spalten
 - Aber in einer Anfrage
- Horizontale (und vertikale) Aggregation

univ-B		
salInfo		
category	CS	Math
Prof	55,000	65,000
AssocProf	50,000	55,000
Technician	43,000	44,000

SchemaSQL Anfrage

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

Felix Naumann | VL Informationsintegration | SS 2007

SchemaSQL – Aggregation

33

Gesucht

- Durchschnittliches Gehalt aller Gruppierungen über alle Abteilungen hinweg.

univ-C			
CS		Math	
category	salFloor	category	salFloor
Prof	60,000	Prof	70,000
AssocProf	55,000	AssocProf	60,000
Technician	42,000	Technician	46,000

Anforderungen

- Durchschnittsbildung über alle Werte zweier Spalten in zwei Relationen
- Horizontale (und vertikale) Aggregation

Iteration über alle Tupel aller Relationen

SchemaSQL Anfrage

- ```
SELECT T.category,
 avg(T.salFloor)
FROM univ-C -> D,
 univ-C::D T
GROUP BY D.category
```

## Überblick

34

- Wiederholung
  - Strukturelle Heterogenität
  - Multidatenbanken
- SchemaSQL
  - Basis-Syntax
  - Aggregation
  - (Umstrukturierung)
  - Architektur und Implementierung



35

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

| univ-B     |        |        |
|------------|--------|--------|
| salInfo    |        |        |
| category   | CS     | Math   |
| Prof       | 55,000 | 65,000 |
| AssocProf  | 50,000 | 55,000 |
| Technician | 43,000 | 44,000 |

Gesucht

- Umstrukturierung der Daten aus univ-B in das Schema von univ-A

Anforderung

- Trennung
  - Definition des Outputschemas
  - Umstrukturierung der Daten

SchemaSQL Anfrage

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

36

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

| univ-B     |        |        |
|------------|--------|--------|
| salInfo    |        |        |
| category   | CS     | Math   |
| Prof       | 55,000 | 65,000 |
| AssocProf  | 50,000 | 55,000 |
| Technician | 43,000 | 44,000 |

Gesucht

- Umgekehrt: Umstrukturierung der Daten aus univ-A in das Schema von univ-B

Anforderung

- Dynamische Schemaerzeugung
  - Ich weiß nicht im Voraus welche Attribute das Ergebnis haben wird.

SchemaSQL Anfrage

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

## SchemaSQL – Aggregation

27

univ-D

| salInfo |        |           |            |
|---------|--------|-----------|------------|
| dept    | Prof   | AssocProf | Technician |
| CS      | 75,000 | 60,000    | 40,000     |
| Math    | 60,000 | 45,000    | 38,000     |

faculty

| dname   | fname             |
|---------|-------------------|
| math    | Arts and sciences |
| physics | Arts and sciences |
| cs      | Engineering       |

Durchschnittliches Gehalt aller Angestellten pro Fakultät

Anforderung

- Aggregation über einen Block

SchemaSQL

- ```
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
```

SchemaSQL – Umstrukturierung & Aggregation

28

univ-D

salInfo			
dept	Prof	AssocProf	Technician
CS	75,000	60,000	40,000
Math	60,000	45,000	38,000

faculty

dname	fname
math	Arts and sciences
physics	Arts and sciences
cs	Engineering

Durchschnittliches Gehalt aller Angestellten pro Fakultät und Kategorie

Anforderung

- Aggregation über Block
- Umstrukturierung

SchemaSQL

- ```
CREATE VIEW
average::salInfo(faculty, C) AS
SELECT U.fname, AVG(T.C)
FROM univ-D::salInfo -> C,
 univ-D::salInfo T,
 univ-D::faculty U
WHERE C <> „dept“
AND T.dept = U.dname
GROUP BY U.fname
```
- Outputschema
  - salInfo(faculty, Prof, AssocProf, Technician)

## SchemaSQL – Umstrukturierung & Aggregation

39

univ-D

| salInfo |        |           |            |
|---------|--------|-----------|------------|
| dept    | Prof   | AssocProf | Technician |
| CS      | 75,000 | 60,000    | 40,000     |
| Math    | 60,000 | 45,000    | 38,000     |

### faculty

| dname   | fname             |
|---------|-------------------|
| math    | Arts and sciences |
| physics | Arts and sciences |
| cs      | Engineering       |

### empType

| category   | type           |
|------------|----------------|
| Prof       | Teaching       |
| AssocProf  | Teaching       |
| Technician | Technical      |
| Secretary  | Administrative |

- Durchschnittliches Gehalt aller Angestellten pro Fakultät und Type
  - Anforderung
    - Aggregation über mehrere Blöcke
      - Vertikal über dept
      - Horizontal über category
    - Umstrukturierung
  - SchemaSQL
    - `create view averages::salInfo(faculty, Y) as`

```
select U.fname, avg(T.C)
from univ-D::salInfo-> C,
univ-D::salInfo T,
univ-D::faculty U,
univ-D::empType E,
E.type Y
where C <> "dept" and
T.dept = U.dname and
E.category = C
group by U.fname
```
  - Outputschema
    - salInfo(faculty, Teaching, Technical, Administrative)

## SchemaSQL – Umstrukturierung & Aggregation

40

```
create view averages::salInfo(faculty, Y) as
select U.fname, avg(T.C)
from univ-D::salInfo-> C,
univ-D::salInfo T,
univ-D::faculty U,
univ-D::empType E,
E.type Y
where C <> "dept" and
T.dept = U.dname and
E.category = C
group by U.fname
```

|                      | dept         | category type <sub>1</sub>                        | ... | category type <sub>n</sub>                        |
|----------------------|--------------|---------------------------------------------------|-----|---------------------------------------------------|
|                      | <categories> |                                                   |     |                                                   |
| faculty <sub>1</sub> |              | faculty <sub>1</sub> & category type <sub>1</sub> | ... | faculty <sub>1</sub> & category type <sub>n</sub> |
| ...                  |              | ...                                               | ... | ...                                               |
| faculty <sub>k</sub> |              | faculty <sub>k</sub> & category type <sub>1</sub> | ... | faculty <sub>k</sub> & category type <sub>n</sub> |

## Überblick

41

- Wiederholung
  - Strukturelle Heterogenität
  - Multidatenbanken
- SchemaSQL
  - Basis-Syntax
  - Aggregation
  - Umstrukturierung
  - Architektur und Implementierung

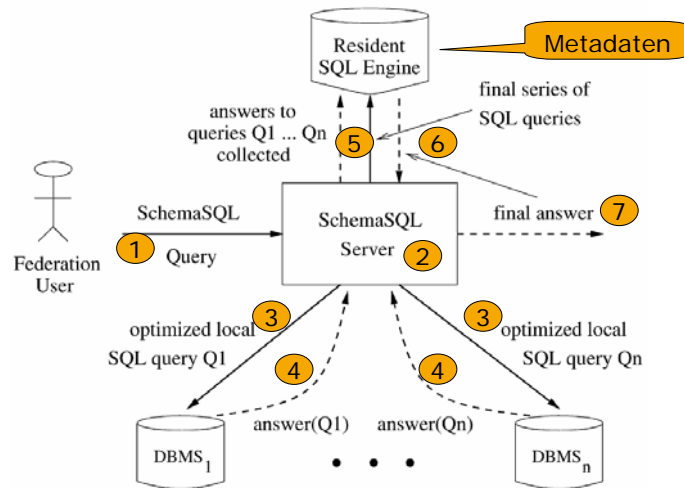


## Implementierung

42

### Anforderungen

- *Non-intrusive*
- Minimaler Eingriff in vorhandene SQL DBMS
- (Optimierung)
- Metadatenverwaltung



- Phase 1
  - Variablen der FROM Klausel instanzieren
    - VITs (Variable instantiation table)
  - Verwendung der Metadatenbank
    - FST (Federation System Table)
    - Schema:
 

```
FST(dbname, relationname, attributename)
```
- Phase 2
  - SchemaSQL Anfrage umschreiben
  - Umgeschriebene Anfrage auf instanziierten Variablen ausführen

## Anfragebearbeitung – Beispiel

| univ-C     |          |
|------------|----------|
| CS         |          |
| category   | salFloor |
| Prof       | 60,000   |
| AssocProf  | 55,000   |
| Technician | 42,000   |
| Math       |          |
| category   | salFloor |
| Prof       | 70,000   |
| AssocProf  | 60,000   |
| Technician | 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, salFloor
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
```

FST(dbname, relname, attname)

Phase 1: Anfragen direkt an einzelne DBMS

$VIT_{RelC}(RelC)$ :

- `SELECT DISTINCT relname FROM FST WHERE dbname = ,univ-C``

- Anfrage an Metadaten

$VIT_C(RelC, CsalFloor)$ :

- Bindings für  $r_i$ :  
`SELECT RelC FROM  $VIT_{RelC}$`
- `SELECT , $r_1$ ` AS RelC, salFloor AS CsalFloor FROM  $r_1$  WHERE category = ,Technician` UNION ... UNION SELECT , $r_n$ ` AS RelC, salFloor AS CsalFloor FROM  $r_n$  WHERE category = ,Technician``

- Anfrage direkt an univ-C!

$VIT_D(Ddept, Dtechnician)$

- `SELECT dept AS Ddept, technician AS Dtechnician FROM salInfo`

- Anfrage direkt an univ-D!

## Anfragebearbeitung – Beispiel

46

- $VIT_{RelC}(RelC)$ 
  - {(Math), (CS)}
- $VIT_C(RelC, CsalFloor)$ 
  - {(CS, 42.000), (Math, 46.000)}
- $VIT_D(Ddept, Dtechnician)$ 
  - {(CS, 40.000), (Math, 38.000)}

| univ-C     |          |
|------------|----------|
| CS         |          |
| category   | salFloor |
| Prof       | 60,000   |
| AssocProf  | 55,000   |
| Technician | 42,000   |

| Math       |          |
|------------|----------|
| category   | salFloor |
| Prof       | 70,000   |
| AssocProf  | 60,000   |
| Technician | 46,000   |

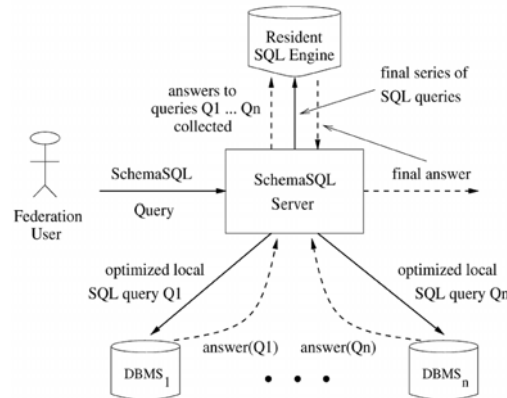
| univ-D  |        |           |            |  |
|---------|--------|-----------|------------|--|
| salInfo |        |           |            |  |
| dept    | Prof   | AssocProf | Technician |  |
| CS      | 75,000 | 60,000    | 40,000     |  |
| Math    | 60,000 | 45,000    | 38,000     |  |

## Anfragebearbeitung – Beispiel

47

### Phase 2: Idee

- VITs sind in internem SQL Server materialisiert
- SchemaSQL Anfrage umschreiben, so dass Ergebnis nur mittels der VITs erzeugt werden kann.



Felix Naumann | VL Informationsintegration | SS 2007

## Anfragebearbeitung – Beispiel

48

| <i>VIT<sub>RelC</sub></i> |  | <i>VIT<sub>C</sub></i> |           | <i>VIT<sub>D</sub></i> |             |
|---------------------------|--|------------------------|-----------|------------------------|-------------|
| RelC                      |  | RelC                   | CsalFloor | Ddept                  | Dtechnician |
| CS                        |  | CS                     | 42,000    | CS                     | 40,000      |
| Math                      |  | Math                   | 46,000    | Math                   | 38,000      |

Erzeuge „JoinedVIT“

- Natural Join über alle VITs: Damit Tupel der gleichen DB zusammenbleiben
- `CREATE VIEW JVIEW(RelC, CsalFloor, Ddept, Dtechnician) AS  
SELECT VITRelC.RelC, VITC.CsalFloor, VITD.Ddept,  
VITD.Dtechnician  
FROM VITRelC, VITC, VITD  
WHERE VITRelC.RelC = VITD.Ddept  
AND VITC.CsalFloor > VITD.Dtechnician  
AND VITRelC.RelC = VITC.RelC`

```
SELECT RelC, salFloor
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
```

Schon bei Erzeugung der VITs

Felix Naumann | VL Informationsintegration | SS 2007



## Anfragebearbeitung – Beispiel

49

Nochmal die JVIT

```

■ CREATE VIEW JVIT(ReIC, CsalFloor, Ddept, Dtechnician) AS
 SELECT VITRelC.ReIC, VITC.CsalFloor, VITD.Ddept,
 VITD.Dtechnician
 FROM VITRelC, VITC, VITD
 WHERE VITRelC.ReIC = VITD.Ddept
 AND VITC.CsalFloor > VITD.Dtechnician
 AND VITRelC.ReIC = VITC.ReIC

```

Erzeuge endgültige Anfrage

```

■ Projektionen, Sortierungen, etc.
■ SELECT ReIC, CsalFloor
 FROM JVIT

```

Felix Naumann | VL Informationsintegration | SS 2007

## Rückblick

50

Wiederholung

- Strukturelle Heterogenität
- Multidatenbanken

SchemaSQL

- Basis-Syntax
- Aggregation
- Umstrukturierung
- Architektur und Implementierung



Felix Naumann | VL Informationsintegration | SS 2007

## Wichtigste Literatur

- [LSS01] Laks V. S. Lakshmanan, Fereidoon Sadri, Subbu N. Subramanian: SchemaSQL: An extension to SQL for multidatabase interoperability. ACM Trans. Database Syst. 26(4): 476-519 (2001)
- Dies ist eine Zusammenfassung der beiden unten genannten paper.

## Weitere Literatur

- [LSS96] Lakshaman, Sadri, Subramanian: SchemaSQL – A Language for Interoperability in Relational Multidatabase Systems, in VLDB 1996
- [LSS99] Lakshaman, Sadri, Subramanian: On Efficiently Implementing SchemaSQL on a SQL Database System, in VLDB 1999