Information Integration

SchemaSQL

18.11.2019

Felix Naumann

Überblick

Wiederholung

1. **Strukturelle Heterogenität**
2. Multidatenbanken

SchemaSQL

1. Basis-Syntax
2. Aggregation
3. Umstrukturierung
4. Architektur und Implementierung

Felix Naumann
Information Integration
Winter 2019/20

Strukturelle Heterogenität

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

Schematische Heterogenität

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

SQL

XQuery

SchemaSQL

Felix Naumann
Information Integration
Winter 2019/20

Schematische Heterogenität

```
Männer( Id, Vorname, Nachname)
Frauen( Id, Vorname, Nachname)
```

Relation vs. Attribut

```
Person( Id, Vorname,
        Nachname,männlich,
        weiblich)
```

Relation vs. Wert

```
Person( Id, Vorname,
        Nachname, Geschlecht)
```

Attribut vs. Wert

Schematische Heterogenität - Lösungen

- Problem
  - Einheitlich auf beide Schemata zugreifen
    - Auf Schemaebene: Schema Mapping und Schema-Sprachen
    - Auf Datenebene: Virtuelle Integration
  - Beide Schemata 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, MSQL, CPL
    - Lose Kopplung, Multidatenbanken
  - Schema Mapping
    - Z.B. Clio, RONDO, u.a.
    - Enge Kopplung, föderierte Datenbanken

Schematische Heterogenität – Lösungen

- 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

Überblick

Wiederholung

1. Strukturelle Heterogenität
2. **Multidatenbanken**

SchemaSQL

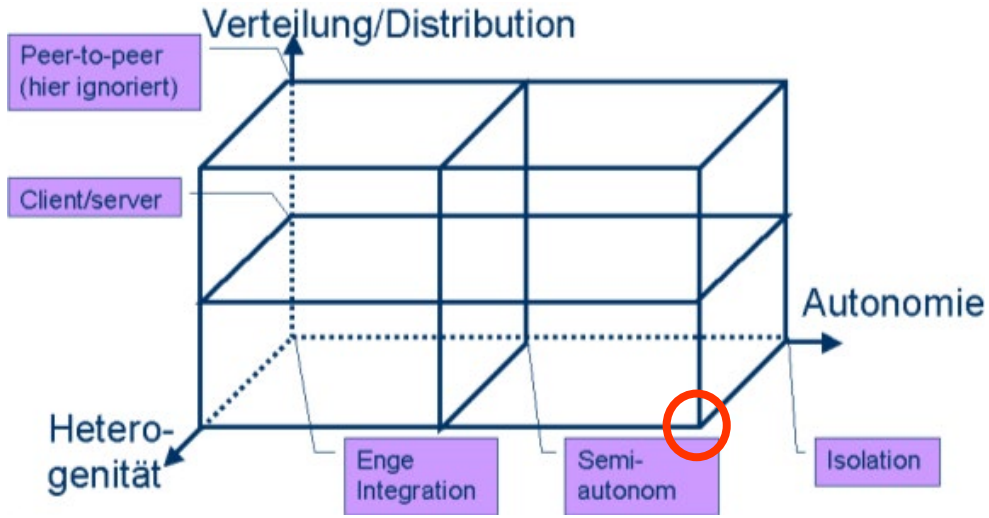
1. Basis-Syntax
2. Aggregation
3. Umstrukturierung
4. Architektur und Implementierung

Felix Naumann
Information Integration
Winter 2019/20

„Aut2, Dist0, Het1“

- Multidatenbanksystem (MDBMS)
- Volle Autonomie
  - Keine bekannte Kooperation
  - Keine Kommunikation untereinander
- Keine Interoperation untereinander möglich
- Integration nur in neuer, integrierender Komponente.
- Z.B. zwei DBMS auf einer Maschine
  - Nicht zur Interoperation entwickelt

- Verteilung wird hier nicht betrachtet, ist aber möglich.

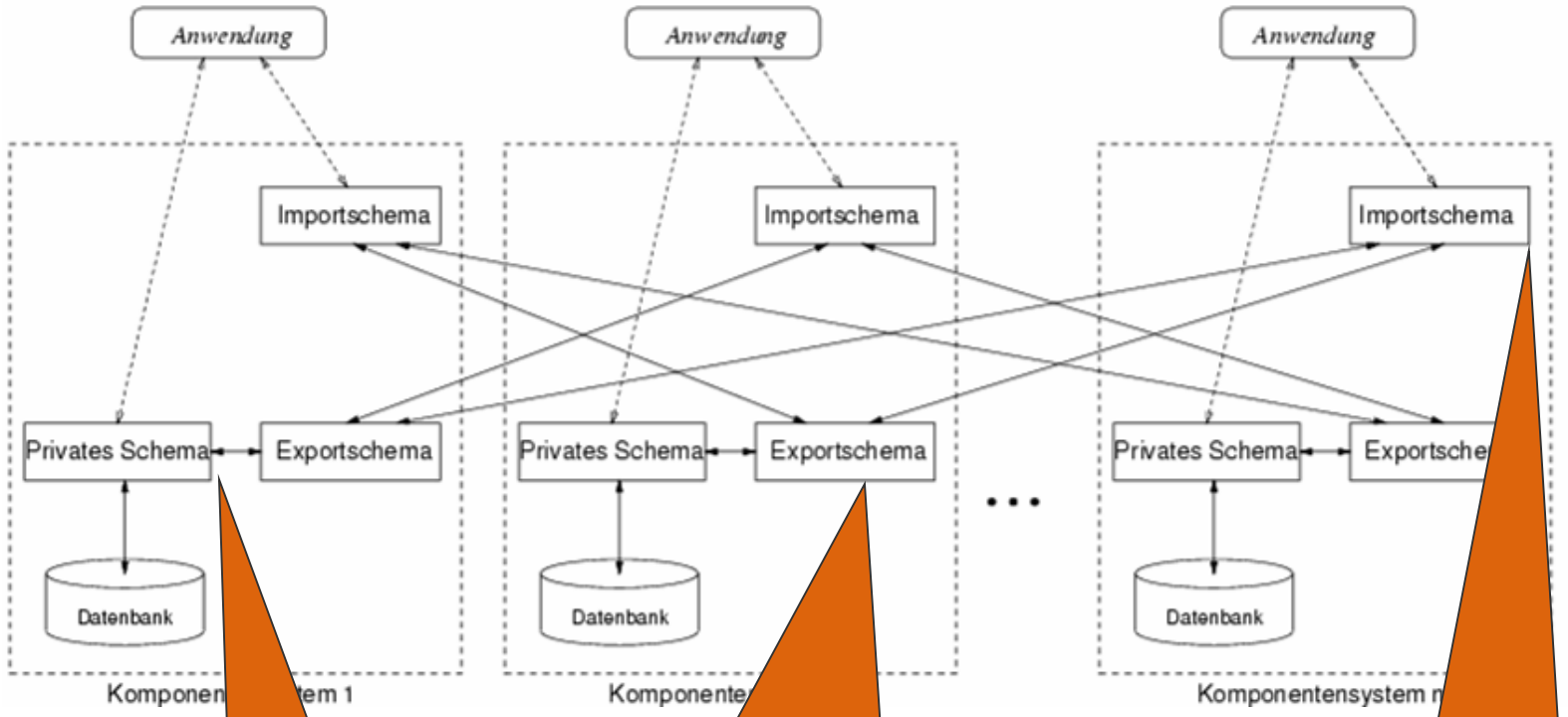


Enge vs. lose Kopplung

- Enge Kopplung
  - Festes, integriertes/föderiertes Schema
    - Modelliert mit Korrespondenzen / Mappings
  - Feste Anfragesprache
  - Kanonisches Datenmodell

- Lose Kopplung
  - Kein festes Schema
    - Nutzer müssen Semantik der Quellen kennen
    - Integrierte Sichten helfen
  - Feste Anfragesprache
  - SchemaSQL [LSS01]
  - Allgemein: Multidatabase query language (MDBQL) [LMR90]

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

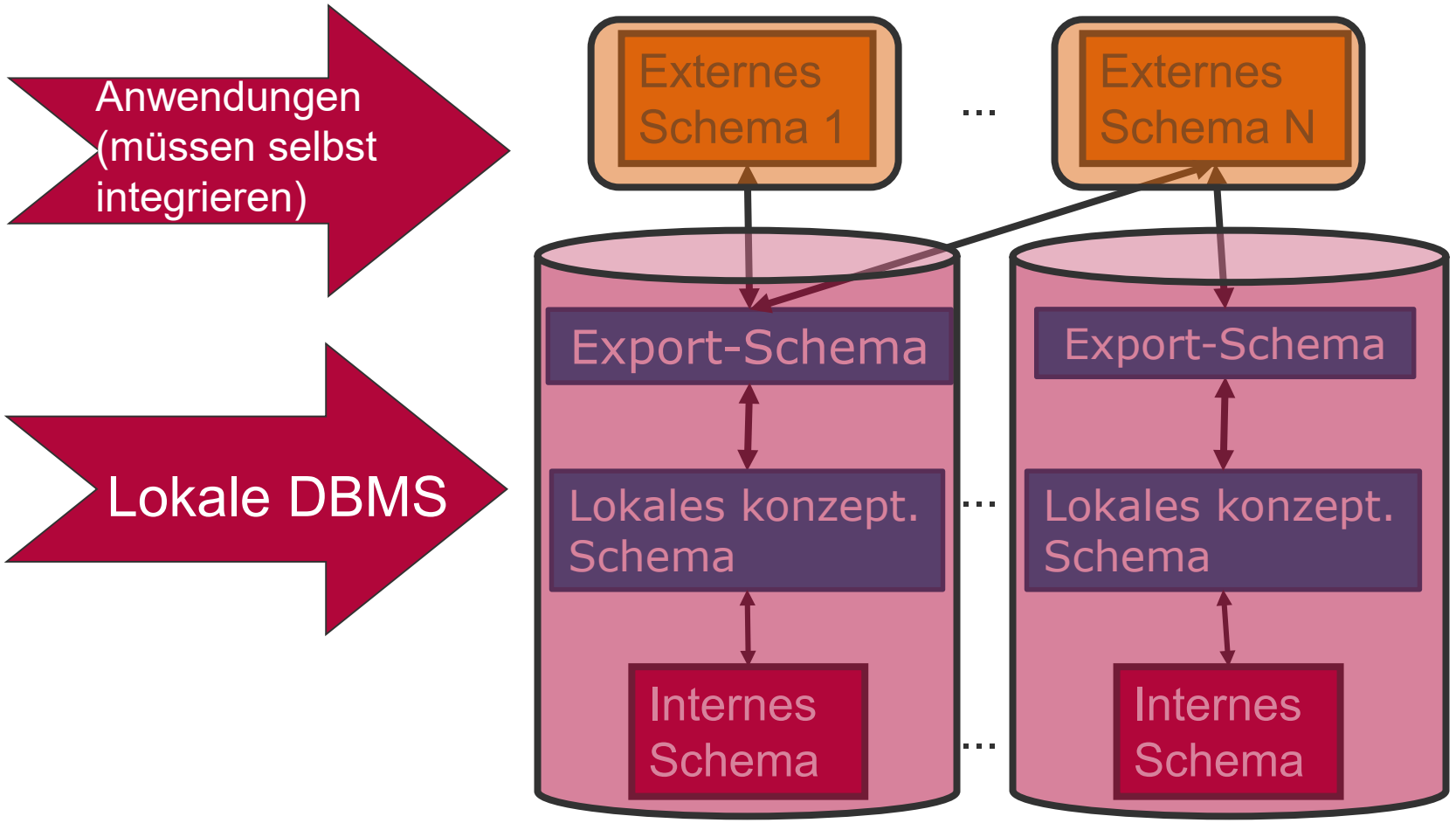


= lokales konzeptionelles Schema

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

Idee: Nur Teilmengen der Exportschemas sollen verwendet werden.

4-Schichten Architektur



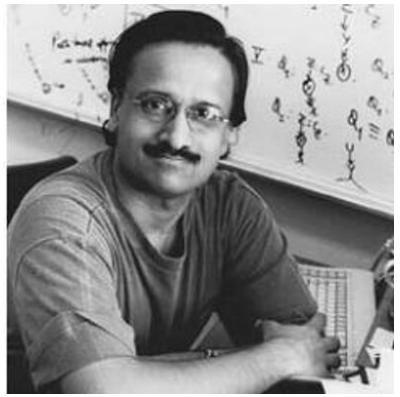
Felix Naumann
Information Integration
Winter 2019/20

Multidatenbanksprachen: 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

SchemaSQL – Features [LSS01]

- 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



Laks V.S. Lakshmanan

Überblick

Wiederholung

1. Strukturelle Heterogenität
2. Multidatenbanken

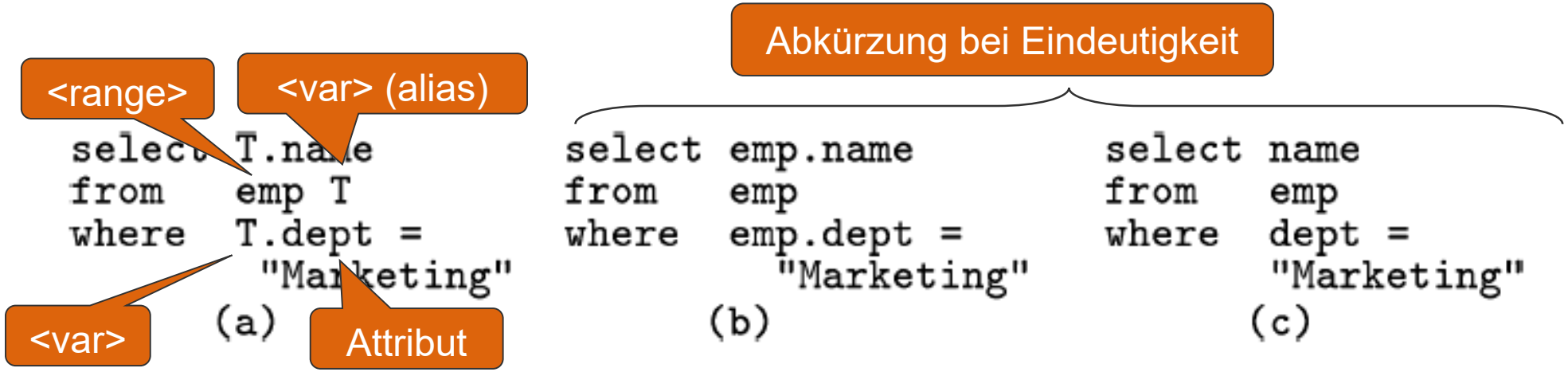
SchemaSQL

1. **Basis-Syntax**
2. Aggregation
3. Umstrukturierung
4. Architektur und Implementierung

Felix Naumann
Information Integration
Winter 2019/20

SchemaSQL – Syntax

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



Felix Naumann
Information Integration
Winter 2019/20

SchemaSQL – Syntax

- 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
  - Z.B.: alle Tupel aller Relationen einer Datenbank
  - Oder: alle Tupel aller Relationen aller Datenbanken
  - Gegebenenfalls mit geeigneten Einschränkungen
    - In WHERE Klausel

Frage: Welcher der 5 ist Standard SQL?

Standard SQL

Felix Naumann
Information Integration
Winter 2019/20

SchemaSQL – Syntax

- Variablendeklaration: <range> <var>
- <range>
  - `->` Iteration über alle Datenbanken
  - `db->` Iteration über alle Relationen in Datenbank db
  - `db::rel->` Iteration über alle Attribute in Relation rel (in db)
  - `db::rel` Iteration über alle Tupel in Relation 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)

SchemaSQL – Beispiel

- Multidatenbank über mehrere Universitäten
  - univ-A, univ-B, univ-C, univ-D

- Information über Angestellte
  - Kategorie (*category*)
  - Gehalt (*salInfo, salFloor* - Grundgehalt)
  - Abteilung (*dept*)

SchemaSQL – Beispiel

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

gleiche Domäne

univ-D

salInfo

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

Felix Naumann
Information Integration
Winter 2019/20

SchemaSQL – Beispiel

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

gleiche Semantik (*dept*)

univ-D

salInfo

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

SchemaSQL – Beispiel

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

gleiche Semantik (*category*)

univ-D

salInfo

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

SchemaSQL – Anfragen

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?

SchemaSQL – Anfragen

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

Join zwischen Attributnamen und Spaltenwerten

- 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

Felix Naumann
Information Integration
Winter 2019/20

SchemaSQL – Anfragen

```
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

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

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**
- 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

```
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

Überblick

Wiederholung

1. Strukturelle Heterogenität
2. Multidatenbanken

SchemaSQL

1. Basis-Syntax
2. **Aggregation**
3. Umstrukturierung
4. Architektur und Implementierung

Felix Naumann
Information Integration
Winter 2019/20

SQL – Herkömmliche Aggregation

- **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

SchemaSQL – Aggregation

- Gesucht
  - Durchschnittliches Gehalt für alle Gruppierungen über alle Abteilungen hinweg.
- Anforderungen
  - Durchschnitt bilden 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
```

Iteration über alle Attribute

SchemaSQL – Aggregation

- Gesucht
  - Durchschnittliches Gehalt aller Gruppierungen über alle Abteilungen hinweg.
- Anforderungen
  - Durchschnittbildung über alle Werte zweier Spalten in zwei Relationen
  - Horizontale (und vertikale) Aggregation

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

SchemaSQL Anfrage

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

Iteration über alle Tupel aller Relationen

Überblick

Wiederholung

1. Strukturelle Heterogenität
2. Multidatenbanken

SchemaSQL

1. Basis-Syntax
2. Aggregation
3. **Umstrukturierung**
4. Architektur und Implementierung

Felix Naumann
Information Integration
Winter 2019/20

SchemaSQL – Umstrukturierung

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**
  - Ähnlich wie Schema Mapping
- 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`
```

SchemaSQL – Umstrukturierung

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
    - Ergebnis-Attribute sind im Voraus nicht bekannt.
- SchemaSQL Anfrage
  -
```
CREATE VIEW AtoB::salInfo(category, D) AS
SELECT A.category, A.salFloor
FROM   univ-A::salInfo A, A.dept D
```

Variable im Schema

Iteration über Attributwerte (also CS und Math)

SchemaSQL – Umstrukturierung

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

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

AtoB

salInfo (allocated)

category	CS	Math
Prof	65,000	null
AssocProf	50,000	null
Technician	45,000	null
Prof	null	60,000
AssocProf	null	55,000
Technician	null	45,000

AtoB

salInfo

category	CS	Math
Prof	65,000	60,000
AssocProf	50,000	55,000
Technician	45,000	45,000

univ-B

salInfo

category	CS	Math
Prof	55,000	65,000
AssocProf	50,000	55,000
Technician	43,000	44,000

SchemaSQL – Aggregation

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 Blöcke (pro Fakultät möglicherweise mehrere Zeilen)
- 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

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 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
    ```
- Outputschema
  - `salInfo(faculty, Prof, AssocProf, Technician)`

Felix Naumann
Information Integration
Winter 2019/20
    ```

SchemaSQL – Umstrukturierung & Aggregation

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
- Anforderungen
  - 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

```
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
```

		category type$_1$	...	*category type*$_n$
	dept	**<categories>**		
faculty$_1$		faculty$_1$ & category type$_1$	... ...	faculty$_1$ & category type$_n$
...		...	...	...
faculty$_k$		faculty$_k$ & category type$_1$	... ...	faculty$_k$ & category type$_n$

Überblick

Wiederholung

1. Strukturelle Heterogenität
2. Multidatenbanken

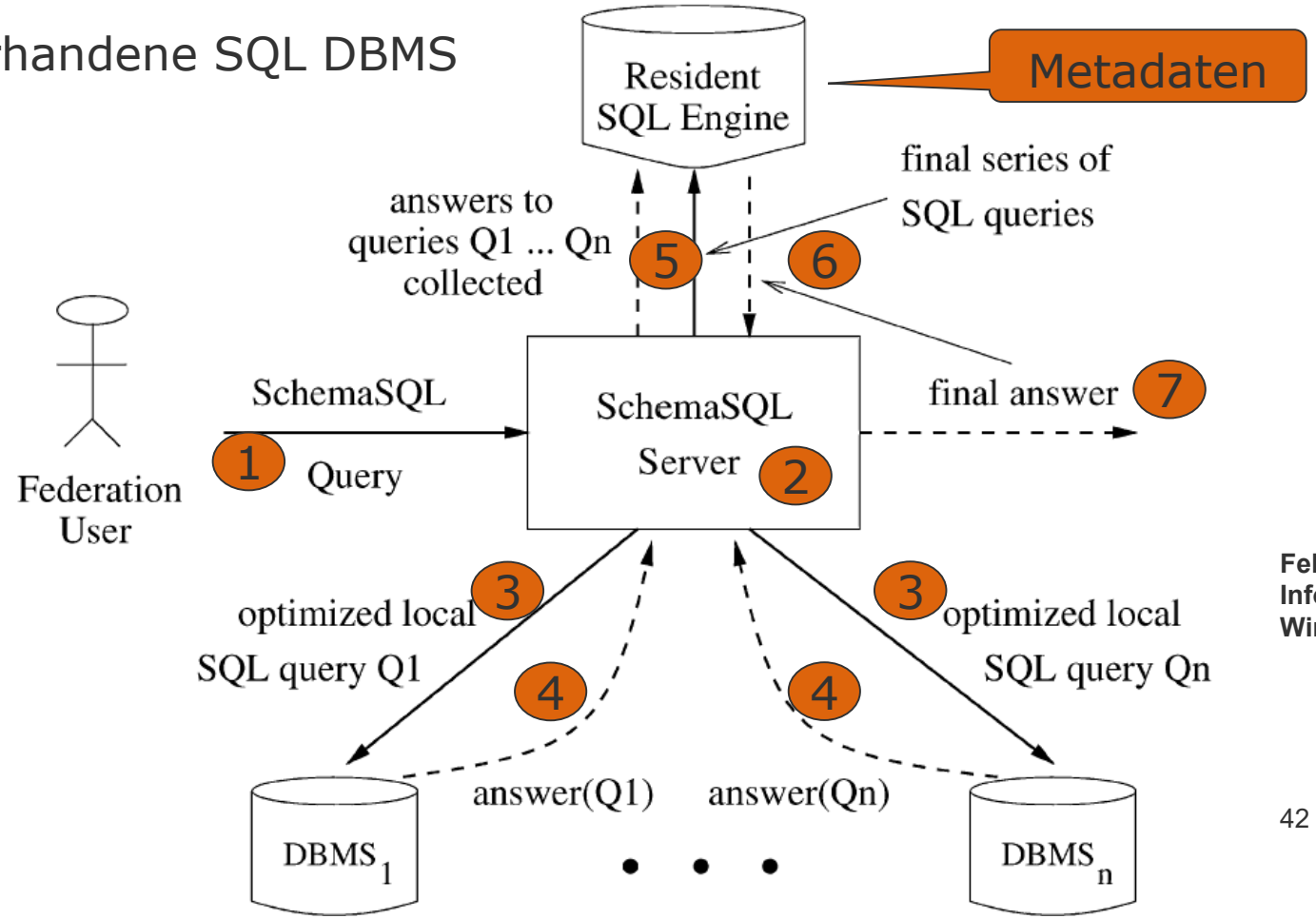
SchemaSQL

1. Basis-Syntax
2. Aggregation
3. Umstrukturierung
4. **Architektur und Implementierung**

Felix Naumann
Information Integration
Winter 2019/20

Implementierung

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



Anfragebearbeitung

- Phase 1
  - Variablen der FROM Klausel instanziieren
    - 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

- 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

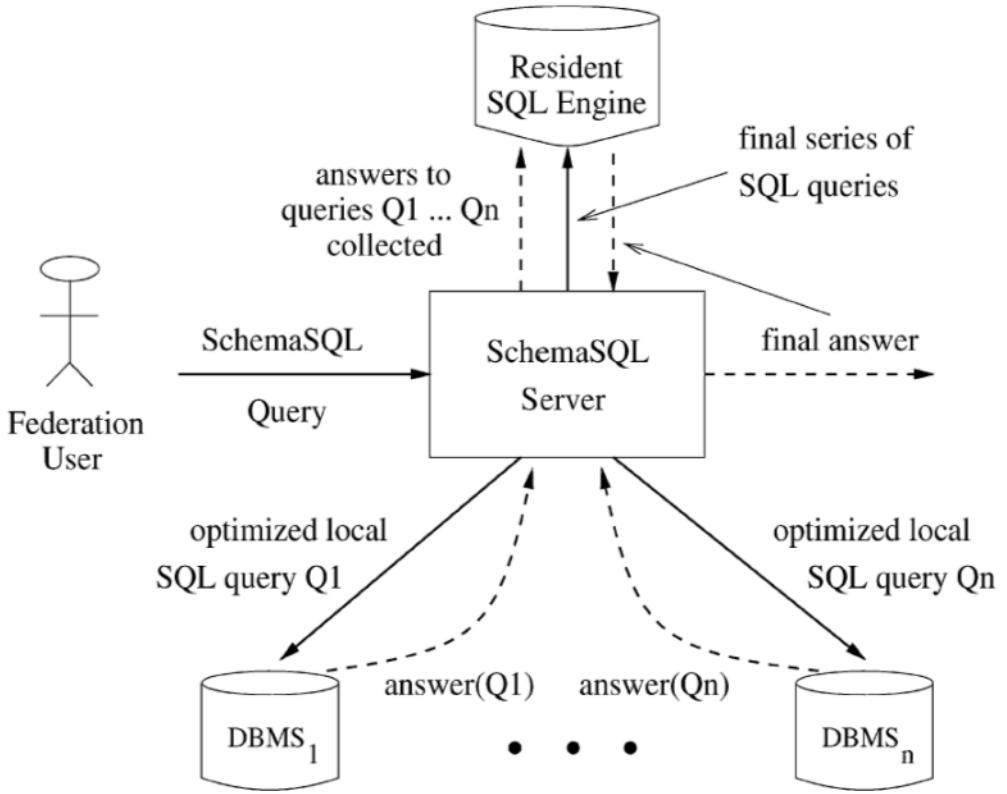
salInfo

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

```
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
```

Anfragebearbeitung – Beispiel

- Phase 2: Idee
  - VITs sind in internem SQL Server materialisiert.
  - SchemaSQL Anfrage umschreiben, so dass Ergebnis nur mittels der VITs erzeugt werden kann.



Anfragebearbeitung – Beispiel

VIT_{RelC}

RelC
CS
Math

VIT_C

RelC	CsalFloor
CS	42,000
Math	46,000

VIT_D

Ddept	Dtechnician
CS	40,000
Math	38,000

- Erzeuge „JoinedVIT“
  - Natural Join über alle VITs: Damit Tupel der gleichen DB zusammenbleiben
  -
```
CREATE VIEW JVIT(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
```

VIT_{RelC} hier nicht nötig

```
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

Anfragebearbeitung – Beispiel

- Nochmal die JVIT
  -
```
CREATE VIEW JVIT(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
```

- Erzeuge endgültige Anfrage
  - Projektionen, Sortierungen, etc.
  -
```
SELECT RelC, CsalFloor
FROM   JVIT
```

Felix Naumann
Information Integration
Winter 2019/20

Rückblick

Wiederholung

1. Strukturelle Heterogenität
2. Multidatenbanken

SchemaSQL

1. Basis-Syntax
2. Aggregation
3. Umstrukturierung
4. Architektur und Implementierung

Felix Naumann
Information Integration
Winter 2019/20

Literatur

- Wichtigste Literatur
  - [LSS01] Laks V. S. Lakshmaman, 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] Lakshmaman, Sadri, Subramanian: SchemaSQL – A Language for Interoperability in Relational Multidatabase Systems, in VLDB 1996
  - [LSS99] Lakshmaman, Sadri, Subramanian: On Efficiently Implementing SchemaSQL on a SQL Database System, in VLDB 1999
  - [LMR90] Litwin, Mark, Roussopoulos: Interoperability of Multiple Autonomous Databases, in ACM Comp. Surveys, 1990