



Informationsintegration

Schema SQL: Multidatenbanksprachen am Beispiel

Ulf Leser

Wo sind wir?

- Architekturen und Kriterien
 - Verteilung, Autonomie, Heterogenität, Transparenz
 - Architekturen
 - Mediator-basierte Systeme
 - Data Warehouses
 - [Multidatenbanksprachen](#), [SchemaSQL](#)
- Techniken zur Anfrageplanung in föderierten Systemen
- Verteilte Anfrageoptimierung
- Schemamanagement
- Datenintegration
- Semantische Integration

Buch

I Grundlagen	1		
1 Einleitung	3		
1.1 Integrierte Informationssysteme	4		
1.2 Grundlegende Begriffe	6		
1.3 Szenarien der Informationsintegration	9		
1.4 Adressaten und Aufbau des Buches	12		
2 Repräsentation von Daten	17		
2.1 Datenmodelle	18		
2.1.1 Das relationale Datenmodell	19		
2.1.2 XML-Daten	23		
2.1.3 Semistrukturierte Daten, Texte und andere Formate	27		
2.1.4 Überführung von Daten zwischen Modellen	32		
2.2 Anfragesprachen	34		
2.2.1 Relationale Algebra	34		
2.2.2 SQL	37		
2.2.3 Datalog	38		
2.2.4 SQL/XML	41		
2.2.5 XML-Anfragesprachen	44		
2.3 Weiterführende Literatur	46		
3 Verteilung, Autonomie und Heterogenität	49		
3.1 Verteilung	51		
3.2 Autonomie	54		
3.3 Heterogenität	58		
3.3.1 Technische Heterogenität	62		
3.3.2 Syntaktische Heterogenität	64		
3.3.3 Heterogenität auf Datenmodellebene	65		
3.3.4 Strukturelle Heterogenität	66		
3.3.5 Schematische Heterogenität	70		
3.3.6 Semantische Heterogenität	73		
3.4 Transparenz	78		
3.5 Weiterführende Literatur	80		
4 Architekturen	83		
4.1 Materialisierte und virtuelle Integration	86		
4.2 Verteilte Datenbanksysteme	91		
4.3 Multidatenbanksysteme	93		
4.4 Föderierte Datenbanksysteme	94		
4.5 Mediatorbasierte Informationssysteme	97		
4.6 Peer-Daten-Management-Systeme	101		
4.7 Einordnung und Klassifikation	104		
4.7.1 Eigenschaften integrierter Informationssysteme	104		
4.7.2 Klassifikation integrierter Informationssysteme	110		
4.8 Weiterführende Literatur	111		
II Techniken der Informationsintegration	113		
5 Schema- und Metadatenmanagement	115		
5.1 Schemaintegration	116		
5.1.1 Vorgehensweise	118		
5.1.2 Schemaintegrationsverfahren	119		
5.1.3 Diskussion	122		
5.2 Schema Mapping	123		
5.2.1 Wertkorrespondenzen	127		
5.2.2 Schema Mapping am Beispiel	129		
5.2.3 Mapping-Situationen	134		
5.2.4 Interpretation von Mappings	137		
5.3 Schema Matching	143		
5.3.1 Klassifikation von Schema-Matching-Methoden	145		
5.3.2 Schemabasiertes Schema Matching	146		
5.3.3 Instanzbasiertes Schema Matching	149		
5.3.4 Kombiniertes Schema Matching	153		
5.3.5 Erweiterungen	155		
5.4 Multidatenbanksprachen	157		
5.4.1 Sprachumfang	158		
5.4.2 Beispiele	159		
5.4.3 Implementierung von SchemaSQL	162		
5.5 Eine Algebra des Schemamanagements	165		
5.5.1 Modelle und Mappings	166		
5.5.2 Operatoren	167		
5.5.3 Schemaevolution	168		
5.6 Weiterführende Literatur	171		
6 Anfragebearbeitung in föderierten Systemen	173		
6.1 Grundaufbau der Anfragebearbeitung	174		
6.2 Anfragekorrespondenzen	184		
6.2.1 Syntaktischer Aufbau	188		
6.2.2 Komplexe Korrespondenzen	189		
6.2.3 Korrespondenzen mit nicht relationalen Elementen	194		
6.3 Schritte der Anfragebearbeitung	195		
6.3.1 Anfrageplanung	195		
6.3.2 Anfrageübersetzung	200		
6.3.3 Anfrageoptimierung	201		
6.3.4 Anfragesausführung	205		
6.3.5 Ergebnisintegration / Datenfusion	207		
6.4 Anfrageplanung im Detail	208		
6.4.1 Prinzip der Local-as-View-Anfrageplanung	209		
6.4.2 Query Containment	213		
6.4.3 »Answering queries using views«	224		
6.4.4 Global-as-View	230		
6.4.5 Vergleich und Kombination von LaV und GaV	231		
6.4.6 Anfrageplanung in PDMS	233		
6.5 Techniken der Anfrageoptimierung	234		
6.5.1 Optimierungsziele	234		
6.5.2 Ausführungsort von Anfrageprädikaten	237		
6.5.3 Optimale Ausführungsreihenfolge	241		
6.5.4 Semi-Join	244		
6.5.5 Globale Anfrageoptimierung	245		
6.5.6 Weitere Techniken	247		
6.6 Integration beschränkter Quellen	250		
6.6.1 Wrapper	252		
6.6.2 Planung mit Anfragebeschränkungen	257		
6.7 Weiterführende Literatur	262		
7 Semantische Integration	267		
7.1 Ontologien	269		
7.1.1 Eigenschaften von Ontologien	272		
7.1.2 Semantische Netze und Thesauri	277		
7.1.3 Wissensrepräsentationsprachen	282		
7.1.4 Ontologiebasierte Informationsintegration	288		
7.2 Das Semantic Web	295		
7.2.1 Komponenten des Semantic Web	298		
7.2.2 RDF und RDFS	300		
7.2.3 OWL – Ontology Web Language	311		
7.2.4 Informationsintegration im Semantic Web	312		
7.3 Weiterführende Literatur	313		
8 Datenintegration	317		
8.1 Datenreinigung	318		
8.1.1 Klassifikation von Datenfehlern	318		
8.1.2 Entstehung von Datenfehlern	322		
8.1.3 Auswirkungen von Datenfehlern	323		
8.1.4 Umgang mit Fehlern	325		
8.1.5 Data Scrubbing	326		
8.2 Duplikaterkennung	329		
8.2.1 Ziele der Duplikaterkennung	330		
8.2.2 Ähnlichkeitsmaße	334		
8.2.3 Partitionierungsstrategien	340		
8.3 Datenfusion	343		
8.3.1 Konflikte und Konfliktlösung	344		
8.3.2 Entstehung von Datenkonflikten	345		
8.3.3 Datenfusion mit Vereinigungsoperatoren	347		
8.3.4 Join-Operatoren zur Datenfusion	349		
8.3.5 Gruppierung und Aggregation zur Datenfusion	352		
8.4 Informationsqualität	353		
8.4.1 Qualitätskriterien	354		
8.4.2 Qualitätsbewertung und Qualitätsmodelle	356		
8.4.3 Qualitätsbasierte Anfrageplanung	359		
8.4.4 Vollständigkeit	362		
8.5 Weiterführende Literatur	365		
III Systeme	369		
9 Data Warehouses	371		
9.1 Komponenten eines Data Warehouse	374		
9.2 Multidimensionale Datenmodellierung	376		
9.3 Extraktion – Transformation – Laden (ETL)	382		
9.4 Weiterführende Literatur	387		
10 Infrastrukturen für die Informationsintegration	389		
10.1 Verteilte Datenbanken, Datenbank-Gateways und SQL/MED	390		
10.2 Objektorientierte Middleware	395		
10.3 Enterprise Application Integration	401		
10.4 Web-Services	404		
10.5 Weiterführende Literatur	407		

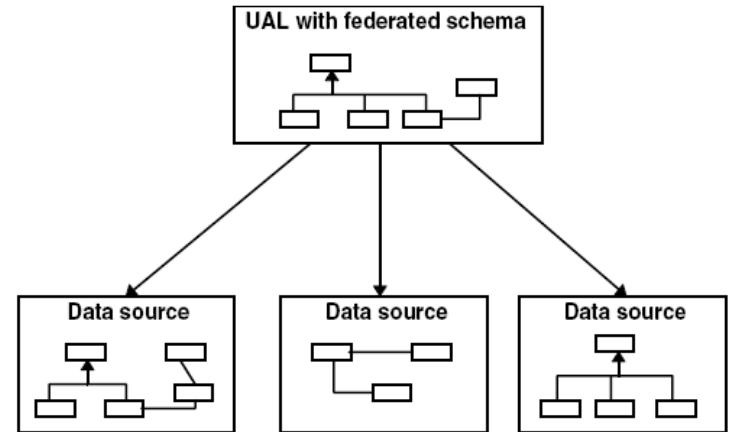
Inhalt dieser Vorlesung

- Multidatenbanksprachen
- SchemaSQL
- Ausklang

Enge versus lose Kopplung

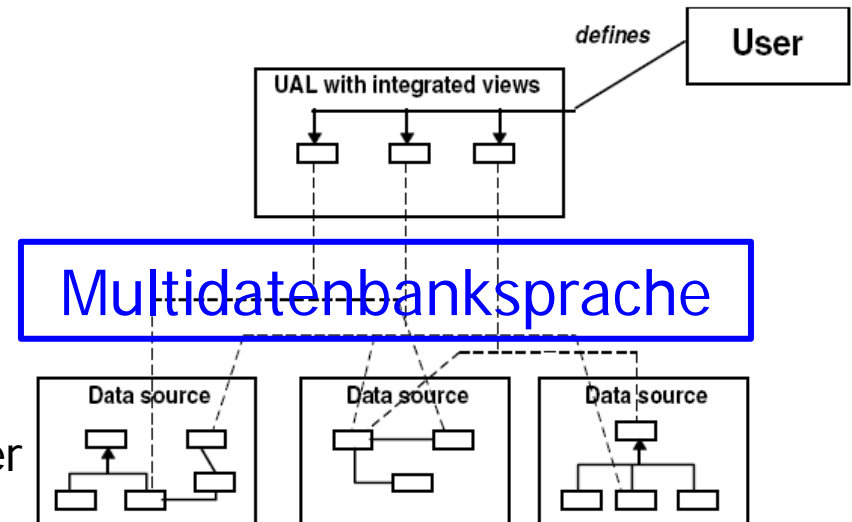
- Enge Kopplung

- Festes und **integriertes Schema**
- Für **Benutzer einheitliche Sicht**
- Automatische Anfrageübersetzung
- System muss Änderungen der Quellen kompensieren



- Lose Kopplung

- Kein integriertes Schema
- Struktur / Semantik: **Nutzer**
- Technische / Datenmodell-heterogenität ist gelöst
- Änderungen gelangen zum Benutzer



Strukturelle Heterogenität

- Schematisch: Unterschiedliche Modellelemente

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

SchemaSQL

- Unterschiedliche Verteilung von Werten auf Tabellen / Attribute

- Unterschiedlicher Normalisierungsgrad
- Fehlende/neue Attribute
- Unterschiedliche Granularität
- Unterschiedliche Fremdschlüssel

SQL

Anforderungen an Multidatenbanksprachen

- **Schemaunabhängigkeit**: Queries gegen „beliebige“ Schema
 - Zugriff auf alle (Meta-)daten
 - SQL Anfragen nur innerhalb eines Schemas gültig
 - Semantischer Check
- **Umstrukturierungsmöglichkeiten**
 - Inklusive Wechsel des Modellelements (Relation, Attribut, Wert)
- Syntaktische Abwärtskompatibilität mit SQL
- **Technische Abwärtskompatibilität** mit existierenden RDBMS
 - Anfragen sollten in SQL-Programme übersetzbar sein
- Effiziente Ausführung

Inhalt dieser Vorlesung

- Multidatenbanksprachen
- SchemaSQL
 - Grundlegende Syntax
 - Zugriff auf Metadaten
 - Horizontale Aggregation
 - Dynamische Sichten
 - Implementierung
- Ausklang

SchemaSQL [LSS96, LSS99, LSS01]

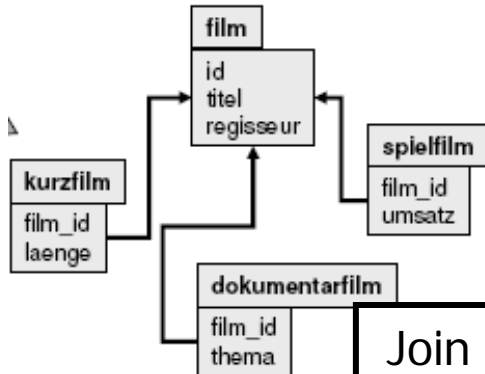
- Erweiterung von SQL
- Ziel: Überbrückung aller Arten **struktureller Heterogenität**
 - Zugriff auf Tabellen in verschiedenen Schemata
 - **Daten und Metadaten** werden gleich behandelt
 - Umstrukturierungen innerhalb der Anfrage
 - **Dynamische Sichten**
 - **Ergebnisschema** hängt vom Zustand der Datenbank ab
 - Horizontale Aggregation (über mehrere Spalten)

Beispiel

- Ziel: **Integrierte Sicht** mit Schema der ersten Quelle
 - Alle Filmtypen der ersten Quelle sollen im Ergebnis vorhanden sein
 - Diese sind durch Werte von **TYP** definiert
 - Tauchen als Tabellennamen in q2 auf

film
id
titel
regisseur
typ
laenge
thema
umsatz

```
CREATE VIEW q1_q2
SELECT id, title, regisseur, typ
FROM q1.film
UNION
SELECT F2.id, F2.title, F2.regisseur, A
FROM q1::Film.typ F1, q2->A, q2::Film F2
WHERE A = F1 AND
      F2.id = A.film_id
```



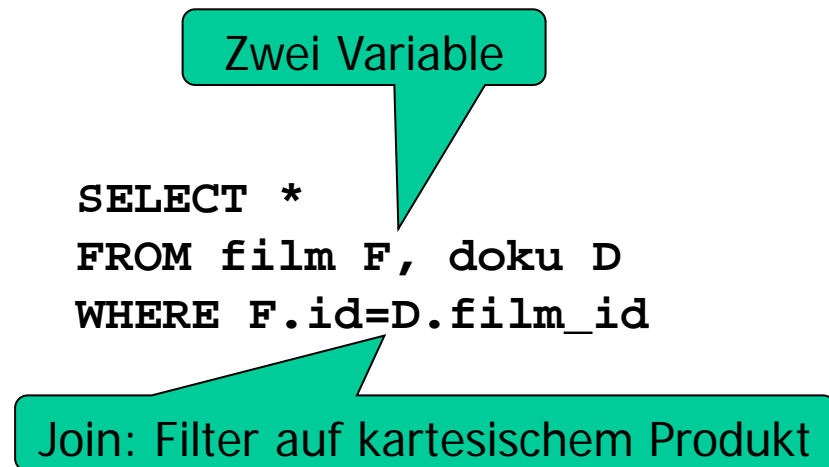
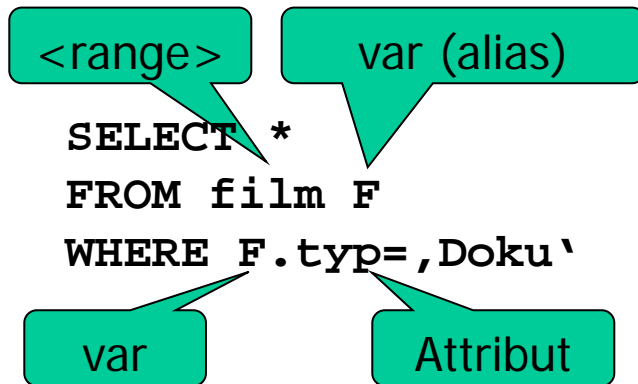
Join über **Attributwerte in q1**
und **Tabellennamen in q2**

Alle **verschiedenen Werte**
von q1.film.typ

Iteriert über alle
Relationennamen von q2

SQL: Semantik

- Variablendeklaration in FROM Klausel
- Variablenverwendung in SELECT und WHERE Klauseln
- Implizites „FORALL“ über alle Tupel einer Relation
 - Joins: Geschachtelte Schleifen
 - Implementierung kann anders vorgehen (z.B. Sort-Merge etc.)



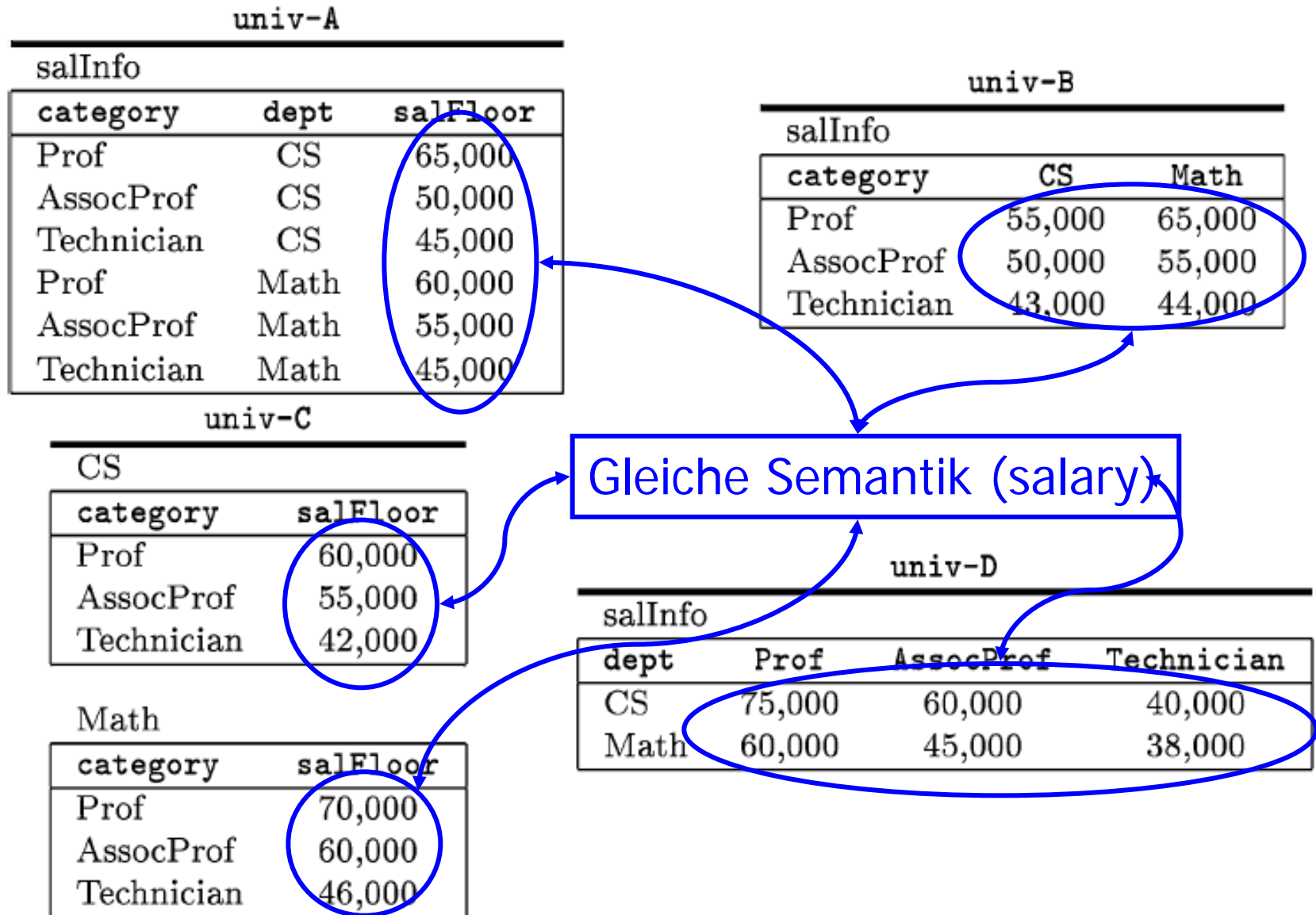
SchemaSQL: Syntax

- Grundlegende Syntax **wie SQL**
 - **SELECT ... FROM ... WHERE**
- Deklaration in FROM Klausel durch **<range> <var>**
- Variablen über **fünf Wertbereichstypen**
 - **->** Alle Datenbanknamen der Multidatenbank
 - **db->** Alle Relationennamen einer Datenbank **db**
 - **db::rel->** Alle Attributnamen einer Relation **rel** in **db**
 - **db::rel** Alle Tupel einer Relation **rel** in **db** (SQL)
 - **db::rel.attr** Alle **verschiedenen Werte** von **attr** in **rel** in **db**
- **Geschachtelte Deklarationen**
 - Spätere Deklarationen der FROM Klausel referenzieren frühere
 - Machen Queries nicht gerade sehr gut lesbar

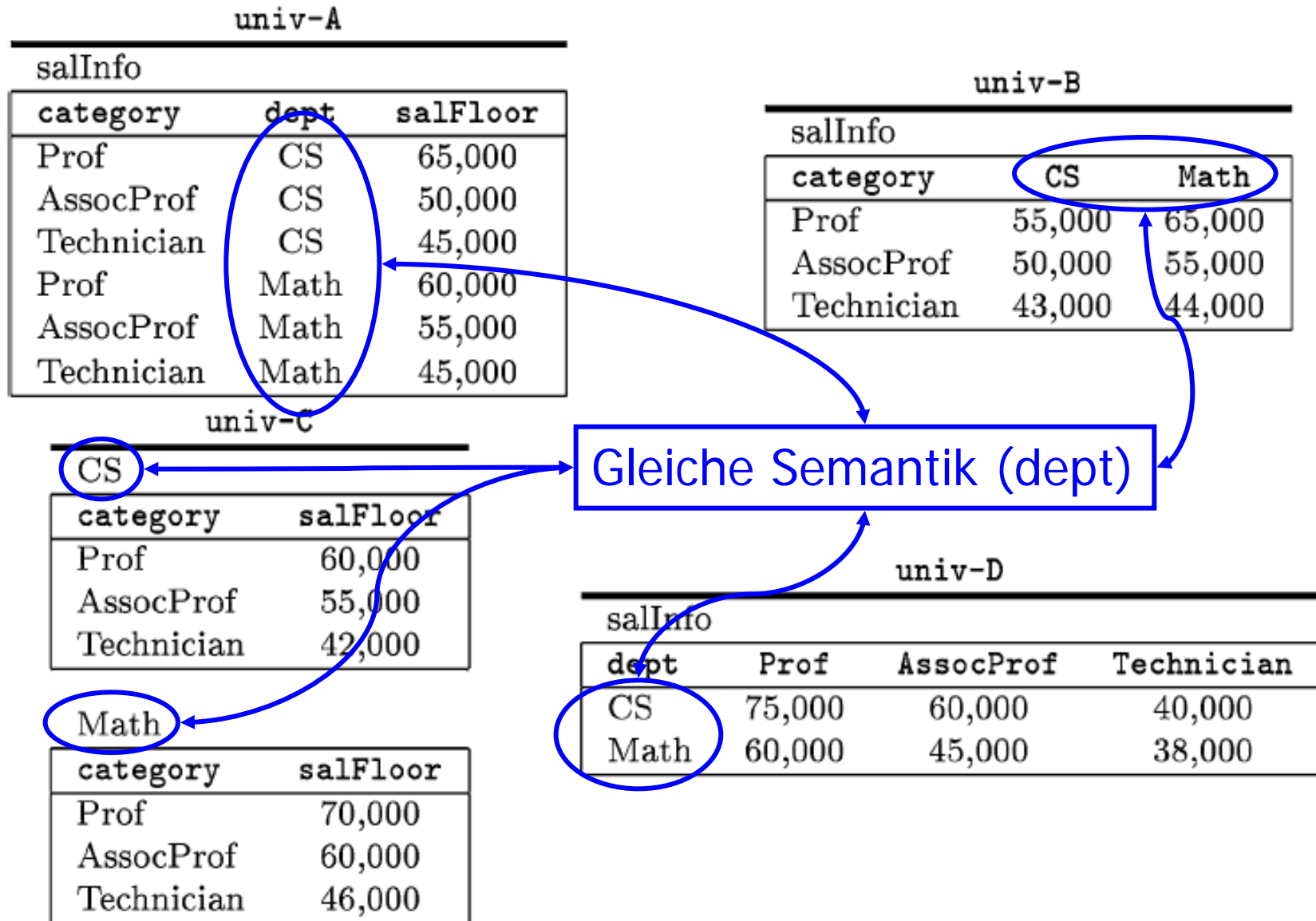
Beispiel

- Multidatenbank aus mehreren Universitätsdatenbanken
 - **univ-A, univ-B, univ-C, univ-D**
- Information über Angestellte
 - Kategorie (**category**) – Prof, Technician, ...
 - Gehalt (**salFloor**)
 - Abteilung (**dept**) – Math, CS, ...

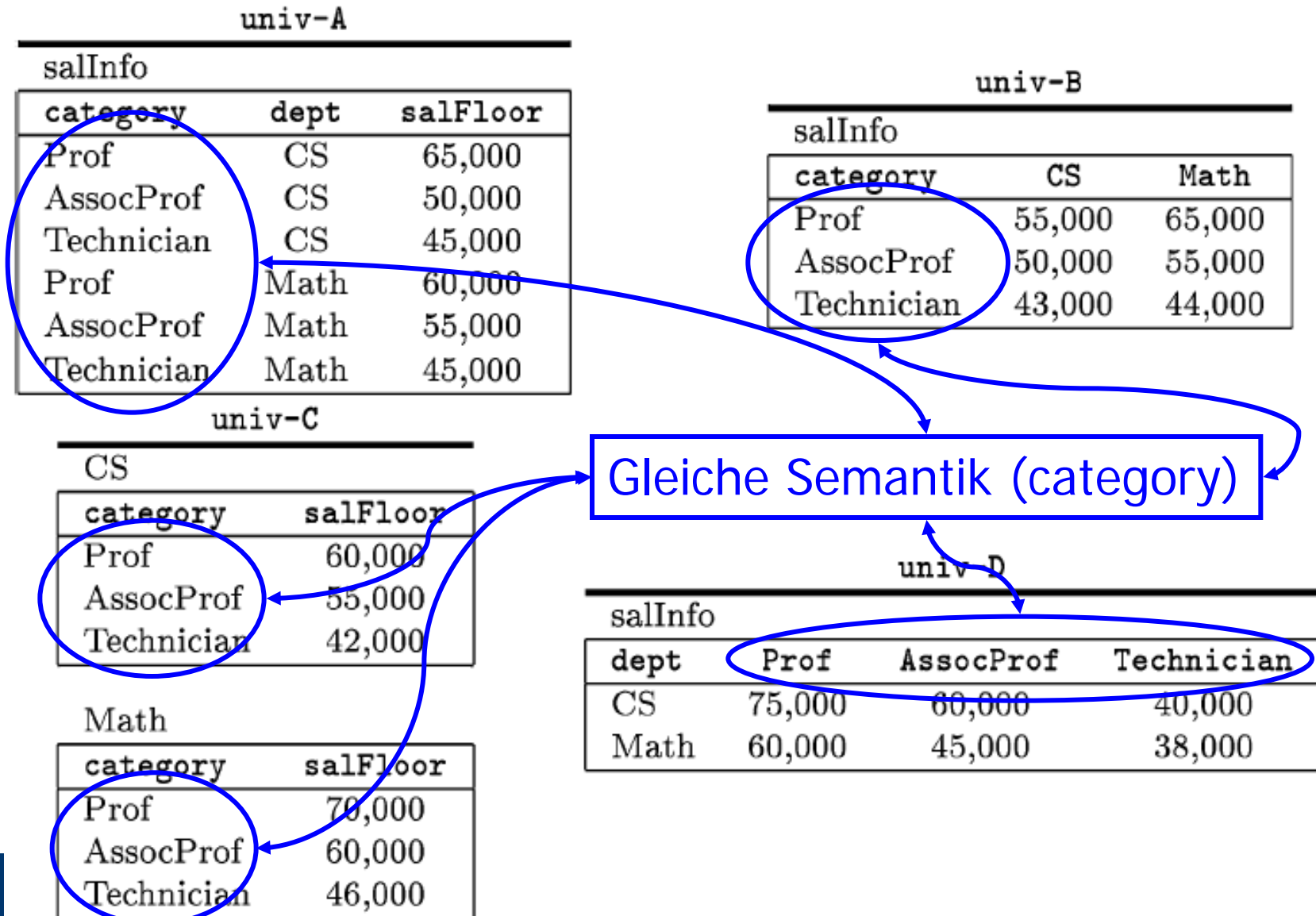
SchemaSQL – Beispiel



SchemaSQL – Beispiel



SchemaSQL – Beispiel



Beispiel 1

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 gleiche Abteilungen in **univ-B**
- Anforderungen
 - Selektionen jeweils auf `Technician`
 - Vergleich der Gehälter
 - Join zwischen beiden Tabellen
 - Verschiedene DBs
 - Über **welches Attribut?**

->	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

Lösung

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 gleiche Abteilungen in **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
```

Join zwischen
Attributnamen und
Spaltenwerten

Alle
Attributnamen

Beispiel 2

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: Alle Abteilungen in **univ-C**, die Technikern mehr zahlen als gleiche Abteilungen in **univ-D**
- Gleiche Anforderungen

->	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

Lösung

univ-C

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

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

- Gesucht: Alle Abteilungen in **univ-C**, die Technikern mehr zahlen als gleiche Abteilungen in **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
Deklaration:
Tupel von C

Iteration über Tupel
beider Tabellen in
univ-C

Join zwischen
Relationennamen
und Spaltenwerten

Auswertung 1: Wertebereiche

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

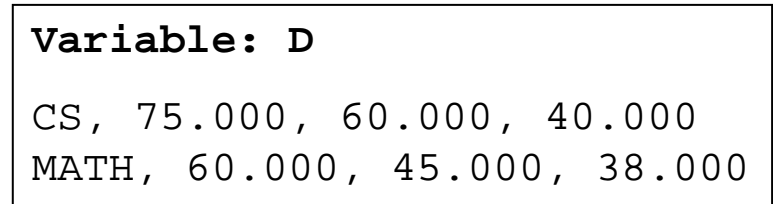
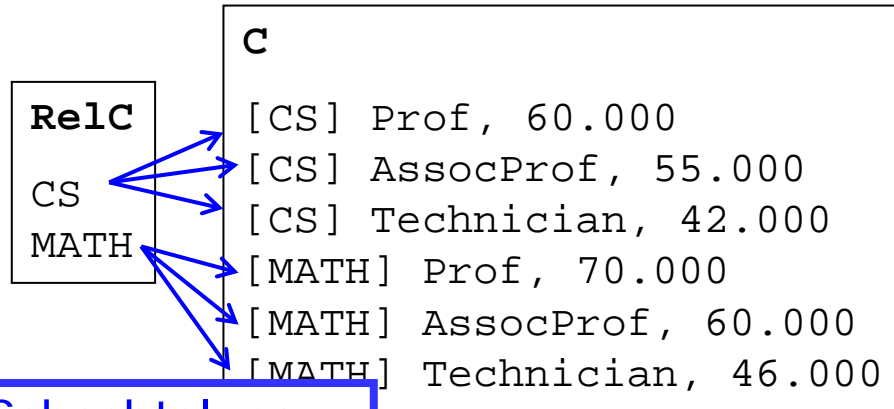
```
SELECT RelC
FROM   univ-C-> RelC,
       univ-C::RelC C,
       univ-D::salInfo D
WHERE  RelC = D.dept AND
       C.category = `Technician` AND
       C.salFloor > D.Technician
```

RelC
CS
MATH

C
[CS] Prof, 60.000
[CS] AssocProf, 55.000
[CS] Technician, 42.000
[MATH] Prof, 70.000
[MATH] AssocProf, 60.000
[MATH] Technician, 46.000

D
CS, 75.000, 60.000, 40.000
MATH, 60.000, 45.000, 38.000

Auswertung 2: Kartesisches Produkt



Schachtelung

CS	Prof, 60.000	CS, 75.000, 60.000, 40.000
CS	Prof, 60.000	MATH, 60.000, 45.000, 38.000
CS	AssocProf, 55.000	CS, 75.000, 60.000, 40.000
CS	AssocProf, 55.000	MATH, 60.000, 45.000, 38.000
CS	Technician, 42.000	CS, 75.000, 60.000, 40.000
CS	Technician, 42.000	MATH, 60.000, 45.000, 38.000
MATH	Prof, 70.000	CS, 75.000, 60.000, 40.000
MATH	Prof, 70.000	MATH, 60.000, 45.000, 38.000
MATH	AssocProf, 60.000	CS, 75.000, 60.000, 40.000
MATH	AssocProf, 60.000	MATH, 60.000, 45.000, 38.000
MATH	Technician, 46.000	CS, 75.000, 60.000, 40.000
MATH	Technician, 46.000	MATH, 60.000, 45.000, 38.000

Auswertung 3: Join

```
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
```

CS	Prof, 60.000	CS, 75.000, 60.000, 40.000
CS	Prof, 60.000	MATH, 60.000, 45.000, 38.000
CS	AssocProf, 55.000	CS, 75.000, 60.000, 40.000
CS	AssocProf, 55.000	MATH, 60.000, 45.000, 38.000
CS	Technician, 42.000	CS, 75.000, 60.000, 40.000
CS	Technician, 42.000	MATH, 60.000, 45.000, 38.000
MATH	Prof, 70.000	CS, 75.000, 60.000, 40.000
MATH	Prof, 70.000	MATH, 60.000, 45.000, 38.000
MATH	AssocProf, 60.000	CS, 75.000, 60.000, 40.000
MATH	AssocProf, 60.000	MATH, 60.000, 45.000, 38.000
MATH	Technician, 46.000	CS, 75.000, 60.000, 40.000
MATH	Technician, 46.000	MATH, 60.000, 45.000, 38.000

Auswertung 4: Bedingung 1

```
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
```

CS	Prof, 60.000	CS, 75.000, 60.000, 40.000
CS	AssocProf, 55.000	CS, 75.000, 60.000, 40.000
CS	Technician, 42.000	CS, 75.000, 60.000, 40.000
MATH	Prof, 70.000	MATH, 60.000, 45.000, 38.000
MATH	AssocProf, 60.000	MATH, 60.000, 45.000, 38.000
MATH	Technician, 46.000	MATH, 60.000, 45.000, 38.000

Auswertung 5: Bedingung 2 und Projektion

```
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
```

CS	Technician, 42.000	CS, 75.000, 60.000, 40.000
MATH	Technician, 46.000	MATH, 60.000, 45.000, 38.000

Einschub: Zugriff auf Metadaten?

- Liegen immer im **Data dictionary** (Katalog)
 - [Oracle]: Alle Attributnamen und –typen im eigenen Schema

```
SELECT attribute_name, attribute_type, table_name
FROM user_attributes;
```
 - Tabellennamen, Integritätsconstraints, Berechtigungen, ...
- Können in Anfragen verwendet werden
 - [Oracle] Alle Tabellen, die ein Attribut enthalten, dass denselben Namen hat wie ein Wert in `person2.geschlecht`

```
SELECT t.table_name
FROM user_tables t, person p2
WHERE t.attribute_name = p2.geschlecht;
```
- Kein dyn. Zugriff auf Tabellen „hinter“ den Metadaten
- Kein Standard

<code>person1(<u>Id</u>, vorname, nachname, maennlich, weiblich)</code>
--

<code>person2(<u>Id</u>, vorname, nachname, geschlecht)</code>

Einschub: Was sind Daten, was sind Metadaten?

- Universal database schema
 - Annahme: Datenbankweit eindeutige Tupel-IDs

```
tables( table_name );
attribute( table_name, attribute_name, attribute_type );
tuple( table_name, tuple_id );
value( tuple_id, attribute_name, value );
```
- Kann prinzipiell alles ausdrücken, was ein beliebiges relationales Schema ausdrücken kann
 - Erweiterungen für IC's, Datentypkonformität, ...
- Schemaevolution wird zur Datenmanipulation
 - Neue Spalte: `INSERT INTO attribute VALUES(...)`

Warum brauchen wir mehr?

- Vorteile
 - Sehr flexibel
 - Sehr gut für theoretische Analysen (... über alle Schemata ...)
 - Einfach implementierbar
- Aber ...
 - Konzeptionelle Verwirrung – Daten / Metadaten?
 - Unverständlich – wo ist mein Schema? (meine Daten?)
 - Konsistenzhaltung sehr aufwändig
 - Viele Anfragen werden extrem kompliziert (`SELECT * FROM person`)
 - Schlechte Performanz
 - Alle Daten stehen in einer Spalte einer Tabelle
 - Keine separate Indexierung möglich
 - Keine differenzierte Selektivitätsabschätzungen möglich

Inhalt dieser Vorlesung

- Multidatenbanksprachen
- SchemaSQL
 - Grundlegende Syntax
 - Zugriff auf Metadaten
 - **Horizontale Aggregation**
 - Dynamische Sichten
 - Implementierung
- Ausklang
 - Andere Multidatenbanksprachen
 - Pivot-Operator

Aggregation in SQL

- Gruppierung und Aggregation in SQL ist immer vertikal
- Aggregation: Mehrere Werte einer Spalte zusammenfassen

```
SELECT COUNT(*)  
FROM   mitarbeiter
```

- Gruppierung: Werte einer Spalte zu Gruppen formen

```
SELECT      m.id, SUM(p.Budget), MAX(p.Budget)  
FROM        mitarbeiter m, projekt p  
WHERE       m.p_id = p.p_id  
GROUP BY   m.id
```

Horizontale Aggregation 1

- Ges: Durchschnittliches Gehalt in **univ-B** pro Kategorie über alle Abt.
 - Durchschnitt über Werte in **zwei Spalten**

univ-B

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

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

-> 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

Semantik: Geschachtelte Projektion:
Aggregation über Werte der Attribute
T.D im aktuellen Tupel

Erklärung

univ-B

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

D

Category

CS

MATH

T

Prof, 55, 65

AssocProf, 50, 55

Technician, 43, 44

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

D x T

CS, P, 55, 65

Math, P, 55, 65

CS, AP, 50, 55

Math, AP, 50, 55

CS, T, 43, 44

Math, T, 43, 44

Group-by

P, CS, 55, 65

Math, 55, 65

AP, CS, 50, 55

Math, 50, 55

T, CS, 43, 44

Math, 43, 44

Horiz-AVG

P, CS, 55

Math, 65

AP, CS, 50

Math, 55

T, CS, 43

Math, 44

Result

P, 60

AP, 52.5

T, 43.5

Horizontale Aggregation 2

- Ges: Durchschnittliches Gehalt in **univ-C** pro Kategorie über alle Abt
 - Durchschnitt über Werte in zwei Spalten in **verschiedenen Tabellen**

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

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

-> 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

Normale Aggregation über Liste von Attributwerten

Tafel

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

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

D

CS
MATH

T

[CS], P, 65
[CS], AP, 55
[CS], T, 42
[math], P, 70
[math], AP, 60
[math], T, 46

D x T + schachtelung

CS, P, 65
CS, AP, 55
CS, T, 42
math, P, 70
math, AP, 60
math, T, 46

Group-by

P, CS, 65
 math, 70
AP, CS, 55
 math, 60
T, CS, 42
 math, 46

Gewöhnliche agg

P, 67.5
AP, 57.5
T, 44

Blockweise: Mehrere Spalten, mehrere Tupel

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	MatNat II
Physics	MatNat I
CS	MatNat II

- Durchschnittliches Gehalt in **univ-D** aller Angestellten **pro Fakultät**
 - Aggregation **über einen Block**
 - Mehrere Zeilen, mehrere Spalten

```
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
```

Semantik: Aggregation über
Liste von Listen

Lösung

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	MatNat II
Physics	MatNat I
CS	MatNat II

F

M, MNII
P, MNI
C, MNII

T

CS, 75, 60, 40
Math, 60, 45, 38

C

~~Dept~~
Prof
AssocProf
Tech

```
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
```

T ⋈ F

CS, 75, 60, 40, MNII
Math, 60, 45, 38, MNII

(T join F) x C

CS, 75, 60, 40, MNII, P
CS, 75, 60, 40, MNII, AP
CS, 75, 60, 40, MNII, T
Math, 60, 45, 38, MNII, P
Math, 60, 45, 38, MNII, AP
Math, 60, 45, 38, MNII, T

Group-by

MNII, CS, 75, 60, 40, P
CS, 75, 60, 40, AP
CS, 75, 60, 40, T
Math, 60, 45, 38, P
Math, 60, 45, 38, AP
Math, 60, 45, 38, T

Horizon Agg

MNII, CS, 75
CS, 60
CS, 40
Math, 60
Math, 45
Math, 38

Result

MNII, ...

Inhalt dieser Vorlesung

- Multidatenbanksprachen
- SchemaSQL
 - Grundlegende Syntax
 - Zugriff auf Metadaten
 - Horizontale Aggregation
 - **Dynamische Sichten**
 - Implementierung
- Ausklang
 - Andere Multidatenbanksprachen
 - Pivot-Operator

Sichten zur 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 aller Daten aus **univ-B** in das Schema von **univ-A**
- Zwei Schritte
 - Definition des Outputschemas
 - Umstrukturierung der Daten
 - Schema steht hier fest

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

Dynamische Sichten (siehe auch PIVOT)

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-A** in das Schema von **univ-B**
 - Dynamische Schemaerzeugung

Domain: Alle distinct Werte von univ-A::A.dept

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

Semantik: Domainvariable in Output-Schema-Definition eines Views wird dynamisch expandiert zu D_1, D_2, \dots, D_n

Erklärung

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

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

D
CS
Math

Prof, CS, 65.000	CS
AssocProf, CS, 50.000	CS
Technician, CS, 45.000	CS
Prof, Math, 60.000	Math
AssocProf, Math, 55.000	Math
Technician, Math, 45.000	Math

A
Prof, CS, 65.000
AssocProf, CS, 50.000
Technician, CS, 45.000
Prof, Math, 60.000
AssocProf, Math, 55.000
Technician, Math, 45.000

Magie 1

- Geschachtelte Deklaration
 - Kein Kartesisches Produkt

Erklärung 2

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

Prof, CS, 65.000	CS
AssocProf, CS, 50.000	CS
Technician, CS, 45.000	CS
Prof, Math, 60.000	Math
AssocProf, Math, 55.000	Math
Technician, Math, 45.000	Math

Prof, CS, 65.000	CS
Math, 60.000	Math
AssocProf, CS, 50.000	CS
Math, 55.000	Math
Technician, CS, 45.000	CS
Math, 45.000	Math

Magie 2

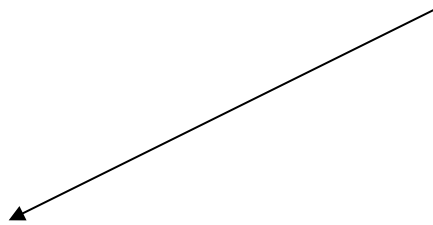
- Gruppieren nach salInfo.category

partment. Intuitively, all tuples in `univ-A::salInfo` corresponding to the same category are grouped together and “merged” to produce one output tuple.

Another aspect of this restructuring view is the use

Erklärung 3

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



Prof, CS, 65.000	CS
Math, 60.000	Math
AssocProf, CS, 50.000	CS
Math, 55.000	Math
Technician, CS, 45.000	CS
Math, 45.000	Math

Category	CS	Math
Prof	65.000	60.000
AssocProf	50.000	55.000
Technician	45.000	45.000

Magie 3

- Jeder Wert von D wird eine Spalte des Ausgabeschemas
- Übernimmt implizit Werte von salFloor entsprechend Werten von A.dept = D
- Das ist nicht gut lesbar ...

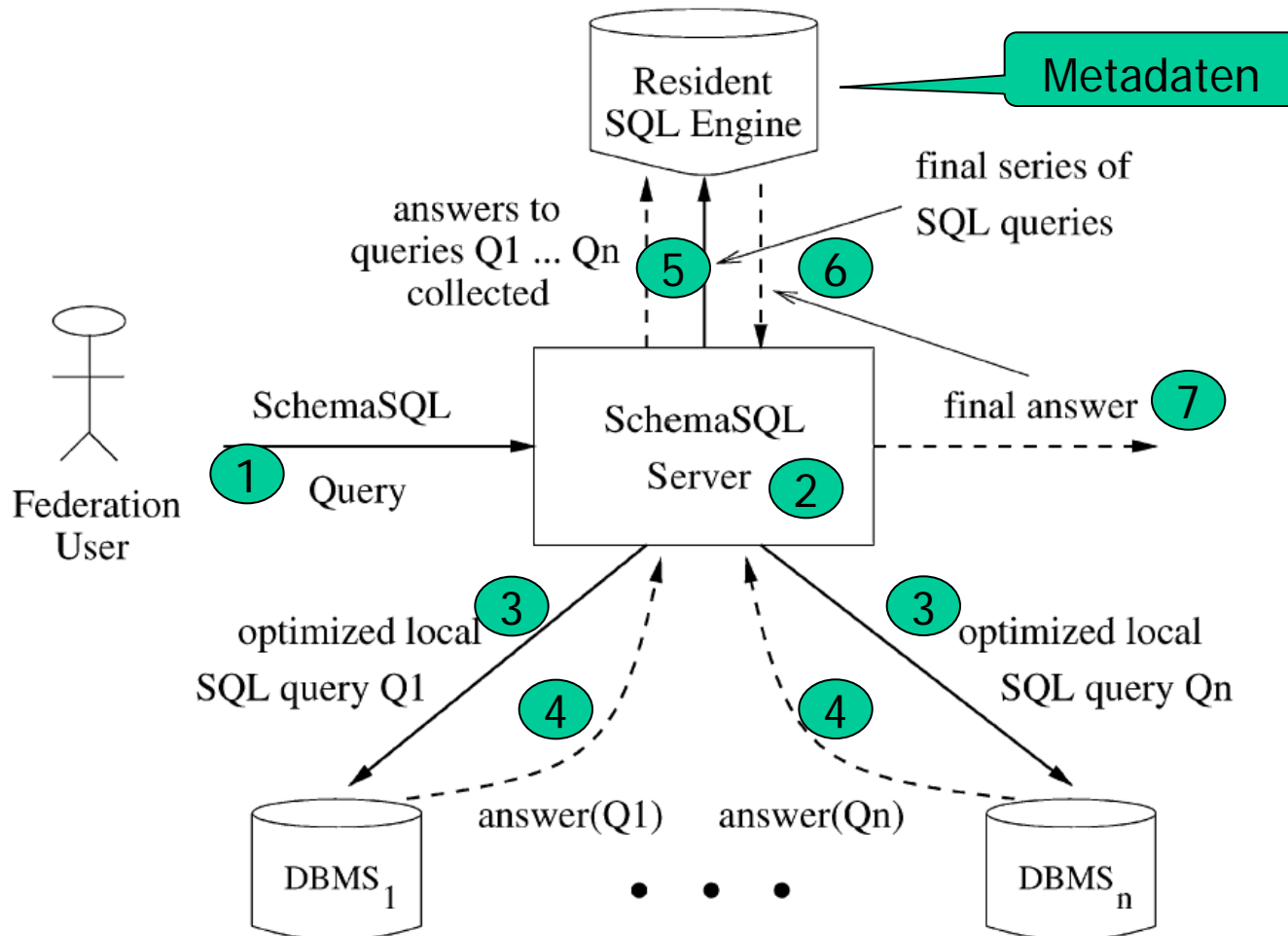
Inhalt dieser Vorlesung

- Multidatenbanksprachen
- SchemaSQL
 - Grundlegende Syntax
 - Zugriff auf Metadaten
 - Horizontale Aggregation
 - Dynamische Sichten
 - [Implementierung](#)
- Ausklang
 - Andere Multidatenbanksprachen
 - Pivot-Operator

Ziele

- „Non-intrusive“
- Ausnutzung vorhandener RDBMS
- Übersetzung von Schema-SQL in **Sequenz von (verteilten) SQL Befehlen**
- Optimierung
- Metadatenverwaltung

Ablauf



Anfragebearbeitung

- Phase 1
 - Variablen der FROM Klausel instantiieren
 - VITs ([Variable instantiation table](#))
 - Unter Verwendung der Metadaten
 - FST ([Federation System Table](#))
 - Schema
- Phase 2
 - SchemaSQL Anfrage in lokale Query gegen VITs umschreiben
 - Umgeschriebene Anfrage in der Resident SQL Engine ausführen

Beispielanfrage

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

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

```
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
```

Phase 1

univ-C	
CS	
category	salFloor
Prof	60,000
AssocProf	55,000
Technician	46,000
Math	
category	sa
Prof	75,000
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	

FST(dbname, relname, attname)

```
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
```

- $VIT_{RelC}(RelC)$
 - Anfrage an Metadaten
`SELECT DISTINCT relname
FROM FST
WHERE dbname = ,univ-C`;`
- $VIT_C(RelC, C.category, C.salFloor)$:
 - Alle Werte von VIT_{RelC}
`SELECT RelC FROM VIT_{RelC};`
 - Zu einer Query an univ-C kompilieren
`SELECT ,CS` AS RelC,
category as Ccategory
salFloor AS CsalFloor
FROM univ-C.CS
UNION ... UNION
SELECT ,MATH` AS RelC ,
category as Ccategory,
salFloor AS CsalFloor
FROM univ-C.MATH;`
- $VIT_D(D.dept, D.technician)$
 - Anfrage direkt an univ-D
`SELECT dept AS Ddept,
technician AS D
FROM univ-D.salInfo;`

Optimiert: Selektionen pushen

univ-C	
CS	
category	salFloor
Prof	60,000
AssocProf	55,000
Technician	46,000
Math	
category	salFloor
Prof	75,000
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	

`FST(dbname, relname, attrname)`

```
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
```

- $VIT_{RelC}(RelC)$
 - Anfrage an Metadaten


```
SELECT DISTINCT relname
FROM FST
WHERE dbname = ,univ-C`;
```
- $VIT_C(RelC, C.salFloor)$:
 - Alle Werte von VIT_{RelC}

```
SELECT RelC FROM VIT_{RelC};
```
 - Zu einer Query an univ-C kompilieren


```
SELECT ,CS` AS RelC,
        salFloor AS CsalFloor
FROM univ-C.CS
WHERE category=,Technician`
UNION ... UNION
SELECT ,Math` AS RelC,
        salFloor AS CsalFloor
FROM univ-C.Math
WHERE category=,Technician`;
```
- $VIT_D(D.dept, D.technician)$
 - Anfrage direkt an univ-D


```
SELECT dept AS Ddept,
        technician AS D
FROM salInfo;
```

Werte in den VIT

- $VIT_{RelC}(RelC)$
 - $\{(Math), (CS)\}$
- $VIT_C(RelC, C.salFloor)$
 - $\{([CS], 42.000), ([Math], 46.000)\}$
- $VIT_D(D.dept, D.technician)$
 - $\{(CS, 40.000), (Math, 38.000)\}$

```
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
```

univ-C

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

Math

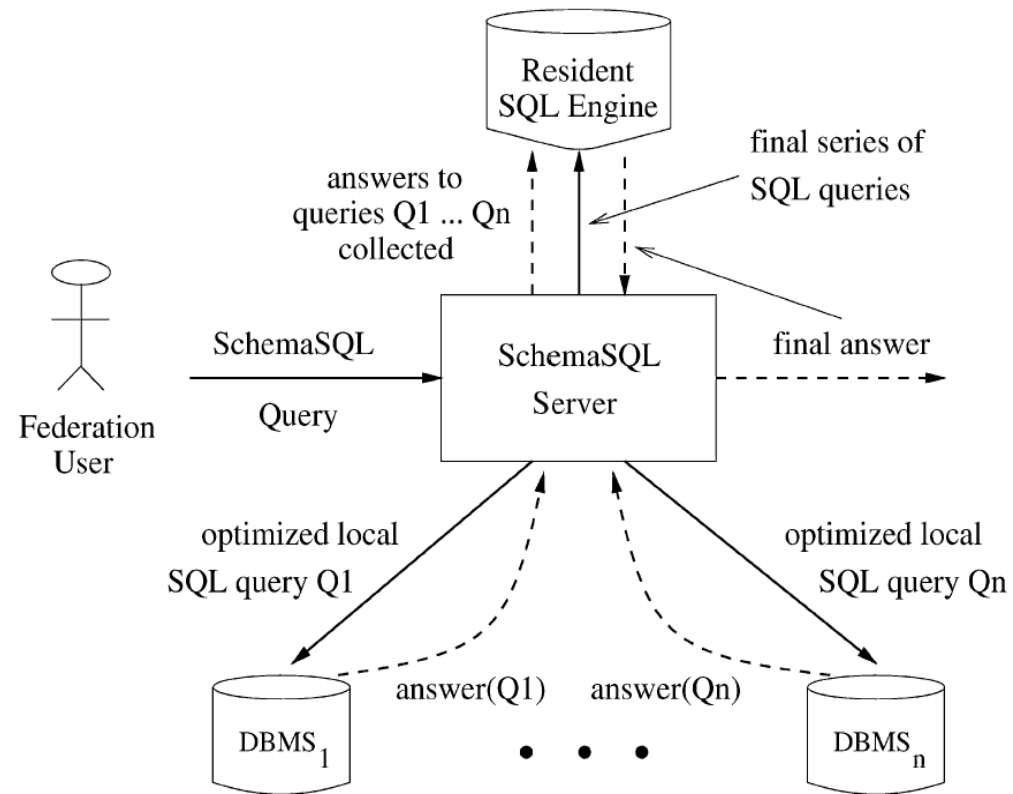
category	salFloor
Prof	70,000
AssocProf	60,000
Technician	46,000

univ-D

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

Phase 2

- VITs sind in lokaler Datenbank **materialisiert**
- SchemaSQL Anfrage so umschreiben, dass das **Endergebnis auf den materialisierten VITs** berechnet wird



Umgeschriebene Anfrage

VIT_{RelC}	VIT_C	VIT_D
RelC	RelC	Ddept
CS	CS	CS
Math	Math	Math
	CsalFloor	Dtechnician
	42,000	40,000
	46,000	38,000

- Zusätzlicher **Natural Join** über alle VITs bei geschachtelten Deklarationen

```
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
```

```
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

Endergebnis

- Noch mal die umgeschriebene Anfrage

```
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
```

- **Endgültige** Anfrage
 - Projektionen, Sortierungen, etc.

```
SELECT  RelC, CsalFloor
FROM    JVIT
```

Optimierungspotential

- Selektionen und Projektionen zu **den Quellen pushen**
- Kombinierte VIT
 - Wenn mehrere Variablen zu einer Datenbank gehören und die Verknüpfung in SQL ausdrückbar ist („normaler“ Join)
 - **Verknüpfte Anfrage an Quelle** schicken
 - Das kombinierte Ergebnis in einer kombinierten VIT speichern
- Ergebnisse einer VIT in die Berechnung der späteren pushen
 - Also gezieltere Anfragen erzeugen – weniger Datentransfer, komplexere Anfragelogik
 - Reihenfolgeproblem (Heuristik: Metadatenzugriffe zuerst)

Inhalt dieser Vorlesung

- Multidatenbanksprachen
- SchemaSQL
- **Ausklang**
 - Andere Multidatenbanksprachen
 - Pivot-Operator

Andere Multidatenbanksprachen

- Vorläufer
 - F-Logic, SchemaLog, Pegasus, SQL/M, ...
 - Beispiel F-Logic: Logische Sprache (wie Tupelkalkül) mit Vererbung, strukturierten und mengenwertigen Attributen, Tupel-ID, ...
 - Zugriff auf Strukturdaten (Metadaten) in einer Anfrage
 - „Semantically First-Order, but syntactically second-order“
- Nachfolger: FISQL/FIRA (Wyss & Wyss 2007, Wyss & Robertson 2005)
 - Basiert auf [erweiterter relationaler Algebra](#) – sauberer Entwurf
 - Generischer: Dynamische Schema, Subqueries, etc.
 - Keine horizontale Aggregationen
 - Regelbasierter Optimierer
 - Ziel: Größtmögliche SQL-Teilqueries direkt an Quellen schicken
 - Vollständig implementiert (angeblich)

First-Order Queries

```
SELECT C.CS
FROM   salInfo C
WHERE  C.category='Technician';
```


$$\{Y \mid \text{salInfo}(X, Y, Z) \wedge X = \text{'Technician'}\}$$

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

Syntactically second-order

```

SELECT B.C
FROM   univ-C-> C,
       univ-B::salInfo B;
    
```



$\{X_i \mid \exists C : C(?) \wedge B.salInfo(X_1, \dots, X_n) \wedge X_i = C\}$

?

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

... semantically first-order ...

- Schemata sind endlich
- Zum Zeitpunkt der Anfrage stehen alle **möglichen Instanzen der Variablen** fest
 - Bzw. während der Ausführung bei geschachtelten Deklarationen
- Umschreiben in **UNIONS von First-Order Anfragen** möglich
 - Genau diese Strategie implementiert SchemaSQL

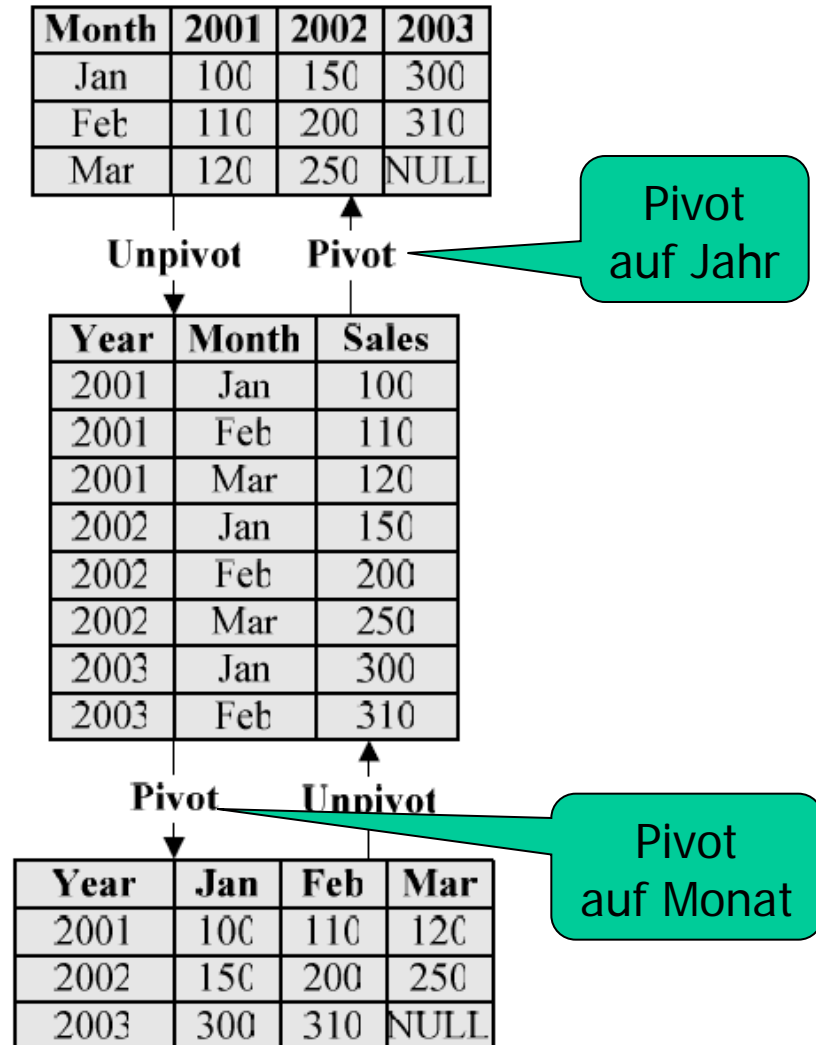
```
SELECT B.C
FROM   univ-C-> C,
       univ-B::salInfo B;
```

```
get_values( univ-C->);
construct_query(

SELECT B.CS
FROM   univ-B::salInfo B
UNION
SELECT B.Math
FROM   univ-B::salInfo B
);
...
```

Beispiel weitere Operatoren: Pivot

- Existiert in diversen Systemen
 - Excel, Access, ...
- Vertauscht Spalten und Zeilen einer Tabelle
 - **Pivot**: „Ausklappen“ aller **DISTINCT Werte einer Spalte** als eigene Spalte
 - Ergebnis hat mehr Spalten, aber weniger Zeilen
 - **Unpivot**: „Einklappen“ aller Spalten als zusätzliches Zeilenpräfix
 - Ergebnis hat weniger Spalten, aber mehr Zeilen
- Voraussetzung: Eindeutigkeit
 - Nur ein Wert pro Monat/Jahr
 - Sonst „Data Collision“



Pivot in SQL

- Ist in SQL ausdrückbar
 - Komplizierte Anfragen
 - Hängen von Attributwerten ab
 - Syntactically second-order
 - Schwierig zu optimieren
- Raum für **eigene Operatoren**
 - Pivot-Operator seit ~2009 in kommerziellen RDBMS

Year	Month	Sales
2001	Jan	100
2001	Feb	110
2001	Mar	120
2002	Jan	150
2002	Feb	200
2002	Mar	250
2003	Jan	300
2003	Feb	310

Pivot Unpivot

Year	Jan	Feb	Mar
2001	100	110	120
2002	150	200	250
2003	300	310	NULL

SELECT
Year,

(SELECT Sales FROM SalesTable AS T2 WHERE Month = 'Jan' AND T2.Year = T1.Year) AS 'Jan',
(SELECT Sales FROM SalesTable AS T2 WHERE Month = 'Feb' AND T2.Year = T1.Year) AS 'Feb',
(SELECT Sales FROM SalesTable AS T2 WHERE Month = 'Mar' AND T2.Year = T1.Year) AS 'Mar'
FROM SalesTable AS T1
GROUP BY Year

Literatur

- SchemaSQL
 - [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)
 - [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
- Pivot
 - [CGGL04] C Cunningham, G Graefe, CA Galindo-Legaria: PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS, in VLDB 2004