



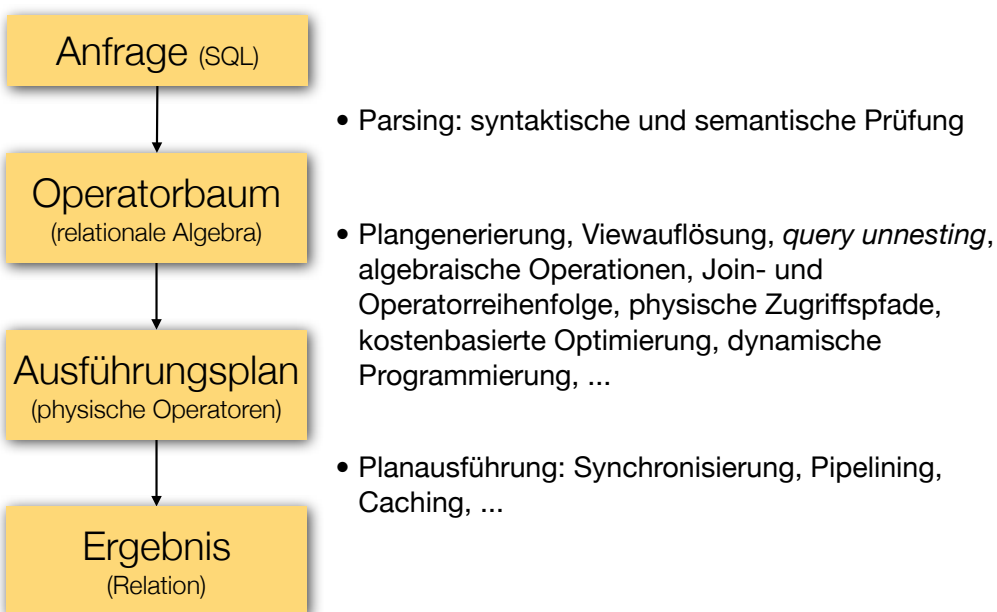
Data Warehouses

Sommersemester 2011

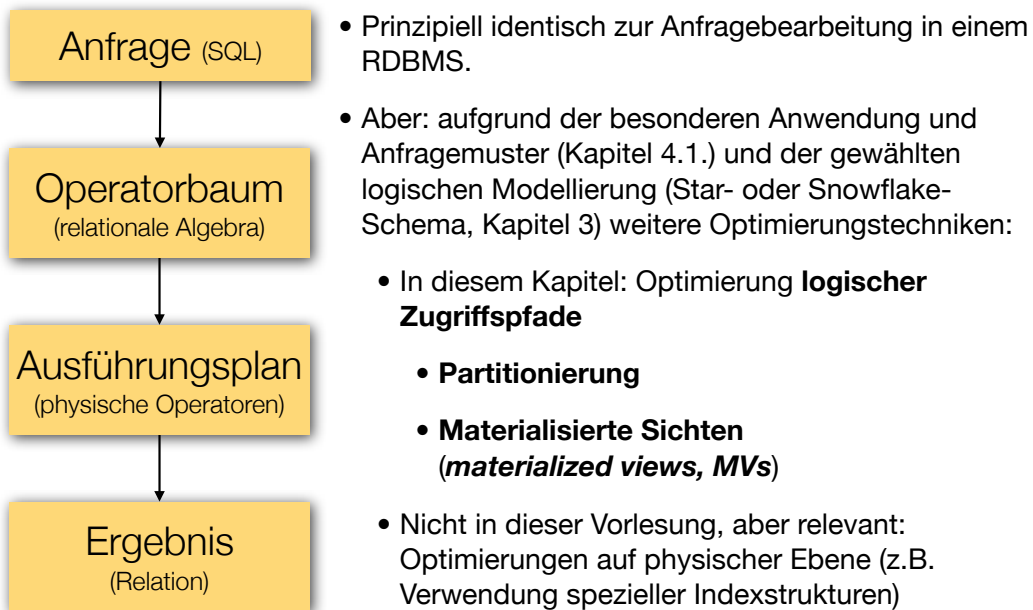
Melanie Herschel
melanie.herschel@uni-tuebingen.de

Lehrstuhl für Datenbanksysteme, Universität Tübingen

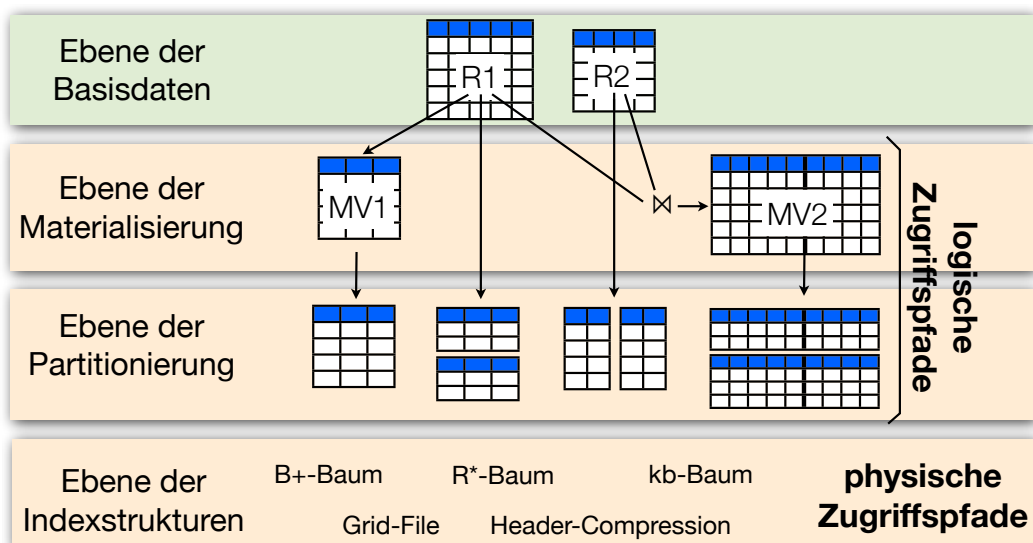
Anfragebearbeitung in einem RDBMS



Anfragebearbeitung in einem Data Warehouse



Logische vs. Physische Zugriffspfade



Bestandteil des konzeptionellen Schemas (ANSI/SPARC 3-Schicht-Architektur)

Bestandteil des inneren Schemas (ANSI/SPARC 3-Schicht-Architektur)

Methoden zur Optimierung Logischer Zugriffspfade

- **Partitionierung**

- Logisch, zu einer Tabelle gehörende Daten werden aufgeteilt um bei Zugriff nicht die komplette Tabelle scannen zu müssen.
- Z.B. sinnvoll, um Scan der oft sehr großen Faktentabelle zu vermeiden.
- Redundanzfrei

- **Materialisierte Sichten (MVs)**

- Oft ausgeführte Anfragen oder Teilanfragen werden vorberechnet und gespeichert.
- Bei der Auswahl des besten Anfrageplans kann nun überprüft werden, ob die Nutzung einer MV einen besseren Zugriffspfad für angefragte Daten darstellt.
- Redundanzbehaftet

Kapitel 4.2

Optimierung auf logischer Ebene

Partitionierung

- Partitionierungsarten
- Management von Partitionen
- Verwendungsmöglichkeiten

Materialisierte Sichten

- Einführung
- Query Containment
- Query Rewriting

Partitionierungsarten

- Aufteilung der Daten **einer Tabelle** in Untereinheiten
 - **Physische Partitionierung** (Allokation): Für den Benutzer transparent
 - Nach Definition der Partitionen
 - **Logische Partitionierung**: Für den Benutzer nicht transparent
 - Explizites Anlegen mehrerer Tabellen
 - Anfragen müssen entsprechend formuliert werden
- Ursprünglich für verteilte Datenbanken entwickelt
 - Verteilung der Partitionen auf verschiedenen Knoten
 - Vereinfachte Synchronisation & Lastverteilung
 - Wichtig ist **Lokalität Anfrage-Partition**

Partitionierungsarten

Vertikale Partitionierung

<u>id</u>	price	amount	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	

<u>id</u>	price	amount
1	50	4
2	60	2
3	30	1
4	20	3

<u>id</u>	code	form	...
1	AX03F	CASH	
2	VF67D	VISA	
3	DS27W	VISA	
4	AP11Z	CASH	

Partitionierungsarten

Vertikale Partitionierung

- „Zerstört“ semantische Einheiten – die Tupel
 - Zusammenfassen erfordert (teure) Joins
 - Geeignete Technik zur „Auslagerung“ von:
 - Selten benutzten Attributen
 - Attributen, die häufiger als andere verändert werden (Und deshalb zu Reorganisation / RowSplits führen)
 - BLOBs, LONGS, etc.
- Herstellung von Transparenz?
 - Definition eines Views
 - Benötigt immer zusätzliche Joins
 - In der Regel Performanceverschlechterung

9

Partitionierungsarten

Horizontale Partitionierung

<u>id</u>	price	amt	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	
4	20	3	AP11Z	CASH	

<u>id</u>	price	amt	code	form	...
1	50	4	AX03F	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	AMEX	
4	20	3	AP11Z	CASH	

<u>id</u>	price	amt	code	form	...
1	50	4	AX03F	CASH	
4	20	3	AP11Z	CASH	
2	60	2	VF67D	VISA	
3	30	1	DS27W	VISA	

10

Partitionierungsarten

Horizontale Partitionierung

- Wichtige Erweiterung der meisten kommerziellen RDBMS
- Hauptvorteile
 - Verwaltung sehr großer Relationen
 - Überspringen von Partitionen bei Anfrageauswertung (*partition pruning*)
 - Ausnutzung paralleler Datenbank- und Systemarchitekturen

Partitionierungsarten

Prinzip

- Partitionen sind eigene Datenbankobjekte (Tabellen)
 - Eigene DDL Kommandos
 - Müssen explizit angelegt werden
- Einmal angelegt, ist ihre Existenz für Benutzer transparent (wie bei Indizes)
- Die letzte Partition kann als Grenze MAXVALUE haben
 - „Specifying a value other than MAXVALUE for the highest partition bound imposes an implicit integrity constraint on the able.“

Explizite Partitionierung in Oracle

```
CREATE TABLE sales_range(salesman_id NUMBER(5),salesman_name VARCHAR2(30),
sales_amount NUMBER(10),sales_date DATE)
PARTITION BY RANGE(sales_date)
(PARTITION t21VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
PARTITION t22VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION t23VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),
PARTITION t24VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY')))
```

Partitionierungsarten

Arten der Horizontalen Partitionierung

- **Bereichspartitionierung**
 - Explizite Angabe der Partitionsbereiche (Über einzelne oder zusammengesetzte Attribute)
 - Typische Attribute: Zeiträume (Gut zur Archivierung historischer Daten)
- **Hash-Partitionierung**
 - Angabe einer Hashfunktion über Attributwerten eines Tupel
 - Sinnvoll, wenn es nicht ein „Hauptanfrageattribut“ gibt
 - Herstellung hoher Parallelität (parallele Joins, parallele Scans), wenn Hashfunktion geeignet gewählt ist.
- **Achtung**
 - Größe der Partitionen wird nicht automatisch ausgeglichen
 - Schlechtes Partitionierungsattribut: Evt. sogar Performanceverschlechterung, da Parallelisierung ausgehebelt

13

Management von Partitionen

- **MERGE** – Verschmelzen zweier Partitionen
- **ADD** – Hinzufügen einer Partition (abhängig von Partitions-Art)
- **DROP/TRUNCATE** – Löschen einer Partition
 - Viel schneller als „DELETE FROM sales WHERE ...“
 - Archivierung ohne Beeinträchtigung anderer Partitionen
 - DW als „Sliding Window“
- **Verteilung** der Partitionen auf verschiedene Platten
 - Parallelität beim IO Zugriff
- **EXCHANGE** – Tabelle ↔ Partition
 - Sehr nützlich für ETL
 - Beispiel: Vorverarbeitung der Daten eines Tages in einer eigenen Tabelle
 - Dann hinzufügen zu sales mit einem Befehl
 - Keine Reorganisation, kein Indexneubau, ...

14

Verwendungsmöglichkeiten

Im Folgenden besprechen wir, wie horizontale Partitionierung eingesetzt werden kann, um die **Anfragebearbeitung** potentiell zu beschleunigen.

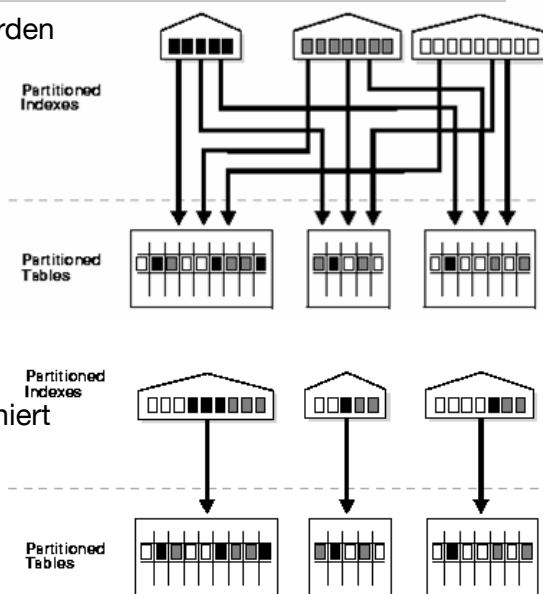
Konkrete Themen sind:

- Partitionierung in Kombination mit **Indizes**
- **Partition Pruning** zur Vermeidung gesamtter Table Scans
- **Parallelität**
- Partitionierung in Kombination mit **Joins**

Verwendungsmöglichkeiten

Partitionierung und Indizes

- Auch Indizes können partitioniert werden
- Indexierung partitionierter Tabellen
- **“Globale” Indizes**
 - Index unabhängig von Tabelle partitioniert
 - Eigene DDL Kommandos
- **“Lokale” Indizes**
 - Index wird wie die Tabelle partitioniert
 - Bildung eines lokalen Index (Indexpartition) pro Partition
 - Manipulation automatisch mit Tabellenpartition (DROP, ADD, ...)



Verwendungsmöglichkeiten

Partition Pruning

- Bei Anfragen mit **Bedingung auf Partitionierungsattribut**.
- **Punktanfragen**, z.B. ... `WHERE sales_date = '03/23/2000'`
 - Direkte Auswahl relevanter Partition
 - Scan zum Auffinden des konkreten Werts nur auf Partition, nicht auf gesamter Tabelle.
 - Schneller als ohne Partitionierung, falls kein Clustered-Index über `sales_date` existiert.
- **Bereichsanfragen**, z.B. ... `WHERE sales_date < '03/23/2000'`
 - Direkte Auswahl relevanter Partitionen
- Unterstützte Prädikate
 - Pruning mit Listen/Bereichspartitionierung nur bei `=, <, >, IN`
 - Pruning mit Hashpartitionierung nur bei `=, IN`

17

Verwendungsmöglichkeiten

Parallelität

Zwei Arten von Parallelität, die unterstützt werden:

- **Interquery**
 - Verteilen einer Menge von Anfragen $\{A_1, \dots, A_n\}$ auf k Prozessoren $\{P_1, \dots, P_k\}$, z.B. $\{A_1, A_2\}$ auf P_1 , $\{A_3\}$ auf P_2 , ...
 - Keine Beschleunigung einzelner Anfragen.
 - Eine Anfrage selbst wird nicht verteilt ausgeführt.

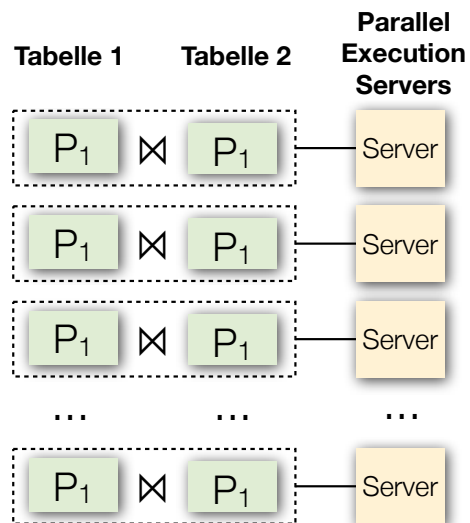
Intraquery

- Aufbrechen einer Anfrage in parallel ausgeführte Teilanfragen
- Die Anfrage wird leicht verändert auf unterschiedlichen Datenbereichen ausgeführt.
- Die Anfrage läuft parallel auf unterschiedlichen Partitionen.

18

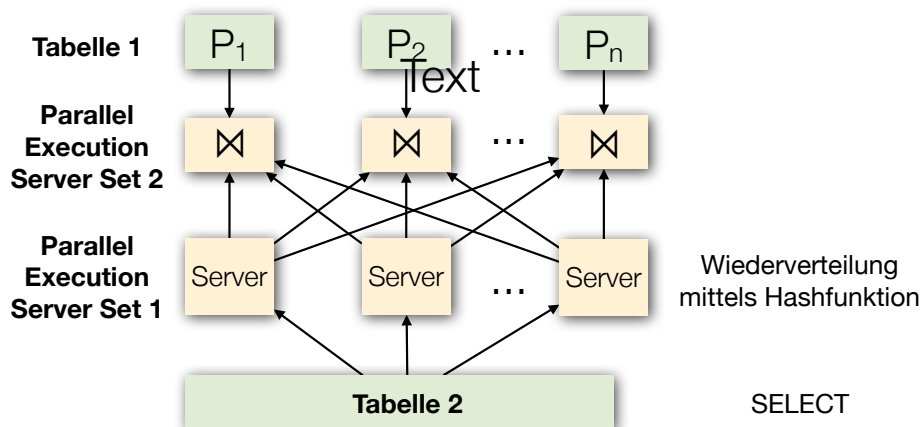
Verwendungsmöglichkeiten Partitionierung und Joins

- Join mit **zwei auf dem Join-Attribut partitionierten Tabellen**.
- Dabei sind die Tabellen nach dem **gleichen Partitionierungskriterium** partitioniert.
- Effektive Parallelisierung möglich.
- Erhebliche Beschleunigung der Anfrageausführung.



Verwendungsmöglichkeiten Partitionierung und Joins

- Join bei nur einer partitionierten Tabelle
 ➔ **Dynamische Partitionierung zur Laufzeit**



Kapitel 4.2

Optimierung auf logischer Ebene

Partitionierung

- Partitionierungsarten
- Management von Partitionen
- Verwendungsmöglichkeiten

➔ Materialisierte Sichten

- Einführung
- Query Containment
- Query Rewriting
- Auswahl
- Implementierung in DBMS

21

Materialisierte Sichten

Grundidee

- Vielzahl gleicher oder ähnlicher Anfragen auf immer denselben Relationen
→ Einführung von Sichten zur Anfragevereinfachung
- Überwiegend lesender Zugriff auf weitgehend stabiler Datenbasis
→ Materialisierung der Sichten ggf. sinnvoll
 - Seltene Änderungen in der Datenbasis bedeuten geringen Aufwand bei der Aktualisierung der Sichten.
 - Materialisierung **reduziert Berechnungsaufwand** bei wiederkehrenden Anfrageteilen.
 - System erkennt automatisch Anfrageteile, deren (Teil-) Ergebnisse durch materialisierte Sichten bereits zur Verfügung stehen.
- Auch: **materialized views (MV), summary tables**

22

Materialisierte Sichten

Probleme

- **Problem 1: Auswahl** der zu materialisierenden Sichten
 - Nötiger Speicherbedarf?
 - Erwartete Laufzeitverbesserung?
- **Problem 2:** Ermittlung einer logischen **Anfrageumschreibung**, die materialisierte Sichten nutzt.
 - Im allgemeinen Fall NP-hartes Problem.
- **Problem 3: Wartung** materialisierter Sichten bei Änderungen in den Quelldaten
 - Neuberechnung?
 - Inkrementelle Wartung?

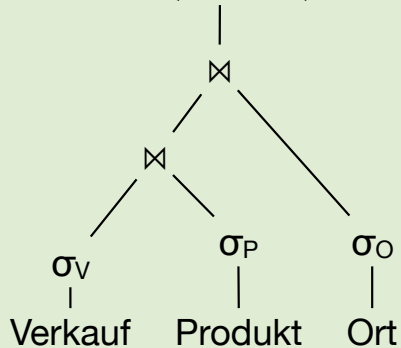
In dieser Vorlesung besprechen wir Lösungen zu Problem 2.

Materialisierte Sichten

Beispiel

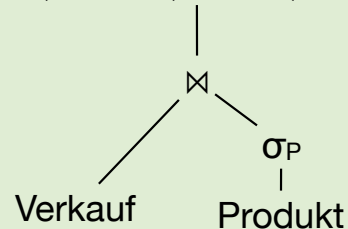
Anfrage q bei vorhandener Sicht v

```
GROUP BY Produkttyp, Stadt;  
SUM(Verkäufe)
```



Anfrage q

```
GROUP BY Produkttyp, Stadt;  
SUM(Verkäufe), COUNT(Anzahl)
```



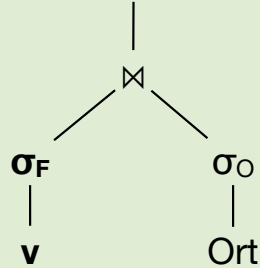
Sicht v

Materialisierte Sichten

Beispiel

Einsetzen der vorhandenen Sicht v in Anfrage q durch Umschreiben (Rewriting)
→ Anfrage q'

```
GROUP BY Produkttyp, Stadt;  
SUM(Verkäufe)
```



Anfrage q'

25

Materialisierte Sichten

Besprechung Beispiel

- Voraussetzung für derartige Anfrageumformulierungen unter Verwendung materialisierter Sichten:
 - Umformulierte Anfrage ist **äquivalent** zur ursprünglichen, d.h. sie liefert dasselbe Anfrageergebnis, unabhängig von der Instanz.
- 2 Problemstellungen
 - Existenz einer Anfrageersetzung (**Query Containment**)
 - Eigentliche Anfrageumformung (**Query Rewriting**)

26

Kapitel 4.2

Optimierung auf logischer Ebene

Partitionierung

- Partitionierungsarten
- Management von Partitionen
- Verwendungsmöglichkeiten

Materialisierte Sichten

- Einführung
- Query Containment
- Query Rewriting



27

Query Containment

- Gegeben sind eine Anfrage q (query) und eine Sicht v (view).
- Fragen
 - ▶ Ist Ergebnis von v identisch dem Ergebnis von q ?
 - ▶ Kurz: Ist v **äquivalent** zu q , $v \equiv q$?
- Rückführung auf „Enthalten sein“ (**containment**)
 - ▶ Ist das Ergebnis von v in q enthalten?
 - ▶ Kurz: Ist v in q enthalten, $v \subseteq q$?
- Denn $v \subseteq q, q \subseteq v \Rightarrow v \equiv q$

Query containment

Sei S ein Datenbankschema, I eine Instanz von S und q_1, q_2 Anfragen gegen I . Sei $q(I)$ das Ergebnis (genau: die Extension) einer Anfrage q angewandt auf I .

Dann ist q_1 **enthalten** in q_2 , geschrieben $q_1 \subseteq q_2$ genau dann wenn $q_1(I) \subseteq q_2(I)$ **für alle möglichen I** .

28

Query Containment

```
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 = „Tübingen“
AND    L.univ = „Tübingen“
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 = „Tübingen“
AND    L.univ = „Tübingen“
```

29

Query Containment

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

```
SELECT K.titel, K.kurs_id
FROM   Kurs K
AND    K.univ = „Tübingen“
AND    K.year > 2007
```

\neq

\neq

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

```
SELECT K.titel
FROM   Kurs K
AND    K.univ = „Tübingen“
AND    K.year < 2010
```

30

Query Containment

Einschub: Datalog Notation

- Im Folgenden: Nur **konjunktive Anfragen**
 - ▶ Nur Equijoins und Bedingungen mit =, <, > zw. Attribut und Konstanten
 - ▶ 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

SQL vs. Datalog Notation

SELECT S.price P, L.region name RN	q(P, RN) :-
FROM sales S, time T, ...	sales(SID, PID, TID, RID, P, ...),
WHERE S.day_id = T.day_id AND	time(TID, D, M, Y),
S.product_id = P.product_id AND	localization(SID, LID, SN, RN),
S.shop_id = L.shop_id AND	product(PID, PN, PGN),
L.shop_id = 123 AND	Y > 1999, SID = 123
T.year > 1999	

Data Warehouses | SS 2011 | Melanie Herschel | Universität Tübingen

Query Containment

Weitere Beispiele in Datalog Notation

Geben Sie für folgende Anfragen in Datalog Notation an, ob q_1 jeweils in q_2 enthalten ist.

$q_1(B, D) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D), \text{path}(D, E)$
 $q_2(A, C) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D)$
 $q_1 \subseteq q_2 ?$

$q_1(C, B) :- \text{path}(A, B), \text{path}(C, A), \text{path}(B, C), \text{path}(A, D)$
 $q_2(X, Z) :- \text{path}(X, Y), \text{path}(Y, Z)$
 $q_1 \subseteq q_2 ?$

$q_1(A, C) :- \text{path}(A, B), \text{path}(B, C)$
 $q_2(A, C) :- \text{threeNodePath}(A, B, C)$
 $q_1 \subseteq q_2 ?$

32

Data Warehouses | SS 2011 | Melanie Herschel | Universität Tübingen

Query Containment

Containment Mapping

Symbol Mapping

Ein **Symbol Mapping** h von einer Anfrage q_2 in eine Anfrage q_1 ist eine Funktion $h: \text{sym}(q_2) \mapsto \text{sym}(q_1)$.

Für ein Literal $l \in q_2$, $l = \text{rel}(A_1, \dots, A_m)$ ist $h(l)$ definiert als $h(l) := \text{rel}(h(A_1), \dots, h(A_m))$.

Bemerkung: jedes Symbol Mapping ist eine Funktion, bildet also jedes Symbol aus q_2 auf exakt ein Symbol aus q_1 ab.

Containment Mapping

Ein **Containment Mapping** (CM) h von Anfrage q_2 nach Anfrage q_1 ist ein Symbol Mapping von q_2 nach q_1 für das gilt:

- $\forall c \in \text{const}(q_2)$ gilt: $h(c) = c$
(Jede Konstante in q_2 wird auf dieselbe Konstante in q_1 abgebildet)
- $\forall l \in q_2$ gilt: $h(l) \in q_1$
(Jedes Literal in q_2 wird auf (mindestens) ein Literal in q_1 abgebildet)
- $\forall e \in \text{exp}(q_2)$ gilt: $h(e) \in \text{exp}(q_1)$, wobei exp exportierte Variablen liefert
(Der Kopf von q_2 wird auf den Kopf von q_1 abgebildet)
- $\text{cond}(q_1) \rightarrow \text{cond}(h(q_2))$
(Die Bedingungen von q_1 sind logisch restriktiver als die von q_2)

33

Query Containment

Containment Mapping Intuition

Containment Mapping von q_2 nach q_1 :

1. $\forall c \in \text{const}(q_2)$ gilt: $h(c) = c$
→ Konstanten dürfen sich nicht ändern.
2. $\forall l \in q_2$ gilt: $h(l) \in q_1$
→ Zusätzliche Literale in q_1 sind nur Filter auf dem Ergebnis;
Containment wird dadurch nicht beeinflusst
3. $\forall e \in \text{exp}(q_2)$ gilt: $h(e) \in \text{exp}(q_1)$
→ Es müssen auch die richtigen Variablen ausgegeben werden;
 q_1 darf weitere Variablen exportieren, aber nicht weniger.
4. $\text{cond}(q_1) \rightarrow \text{cond}(h(q_2))$
→ Bedingungen in q_1 müssen äquivalent oder strikter sein (also höchstens Tupel wegfiltern) als die von q_2 .

34

Query Containment

Containment Mapping Beispiele

Containment Mapping Beispiele

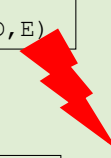
$q_1 \subseteq q_2 ?$

$q_2(A, C) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D)$
 $q_1(B, D) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D), \text{path}(D, E)$

Mapping: $A \rightarrow A, B \rightarrow B, C \rightarrow C, D \rightarrow D$

$q_2(A, C) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D)$
 $q_1(B, D) :- \text{path}(A, B), \text{path}(B, C), \text{path}(C, D), \text{path}(D, E)$

Mapping: $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$



35

Query Containment

Containment Mapping Beispiele

Containment Mapping Beispiele

$q_2(TID, P) :- \text{sales}(SID, TID, P, \dots), \text{time}(TID, D, \dots), D > 28, D < 31$
 $q_1(Y, Z) :- \text{sales}(X, Y, Z, \dots), \text{time}(Y, U, \dots), U > 1, U < 30$

CM: $SID \rightarrow X, TID \rightarrow Y, P \rightarrow Z, D \rightarrow U$
 $h(D) = U$

Aber: $U > 1 \wedge U < 30 \not\rightarrow h(D) > 28 \wedge h(D) < 31$

$q_1 \not\subseteq q_2$

36

Query Containment

Vom Containment Mapping zum Query Containment

Vom Containment Mapping zum Query containment

Theorem:

$q_1 \subseteq q_2$ genau dann wenn es ein containment mapping von q_2 nach q_1 gibt.

Lemma:

$q_1 \equiv q_2$ genau dann wenn es ein containment mapping von q_2 nach q_1 und von q_1 nach q_2 gibt.

- Beweise
 - über die Semantik von Anfragen
 - Literatur [CM77]
 - Ursprünglich zur Anfrageminimierung entwickelt
- Richtungen beachten
 - Containment Mapping von q_2 nach q_1
 - Bedingungen von q_1 implizieren Bedingungen von q_2

37

Query Containment

Mengensemantik

- Alles gesagte gilt nur unter **Mengensemantik**
 - Das ist Standard in SQL, aber nicht in RDBMS
- Beispiel
 - $q_1(X,Y) :- r(X,Y)$
 - $q_2(X,Y) :- r(X,Y), s(Y,Z)$
 - Mengensemantik: $q_2 \subseteq q_1$
 - Aber: Im Ergebnis ist jedes Tupel (X,Y) aus r so oft enthalten, wie $(Y,_)$ in s enthalten ist.
 - Unter Multimengensemantik gilt das Containment daher nicht.
- **Multimengensemantik**
 - Anfragen sind nur dann äquivalent, wenn sie **isomorph** sind; Homomorphismen reichen nicht
 - Containment bei Anfragen mit Ungleichheit ist **unentscheidbar** (PODS2006)

38


Kapitel 4.2

Optimierung auf logischer Ebene

Partitionierung

- Partitionierungsarten
- Management von Partitionen
- Verwendungsmöglichkeiten

Materialisierte Sichten

- Einführung
- Query Containment
-  • Query Rewriting

39

Query Rewriting

Verwendung von Query Containment

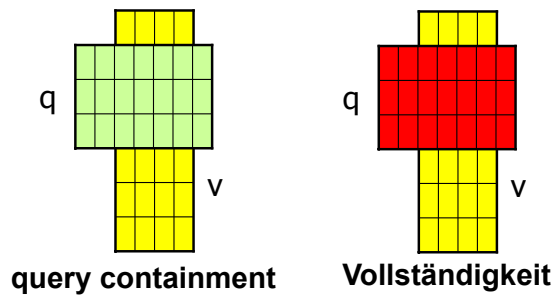
- Gegeben eine Anfrage q und eine materialisierte Sicht v , so können wir mittels Query Containment nun zeigen, ob $q \subseteq v$.
- Wie wenden wir das an?
- **Möglichkeit 1: $v \equiv q$**
 - Fertig: v als Ergebnis von q ausgeben
 - Test auf Äquivalenz erfordert 2 x Containment
- **Möglichkeit 2: $v \subseteq q$ (aber nicht umgekehrt)**
 - v berechnet nur korrekte, aber nicht alle Antworten von q
 - Um q zu beantworten, müsste man v' berechnen so dass $q \equiv v \cup v'$
 - Im Allgemeinen ist diese Berechnung schwierig
 - Keine Verwendung außerhalb der Forschung
- **Möglichkeit 3: $q \subseteq v$ (aber nicht umgekehrt),**
 - Nicht alle Tupel von v sind korrekte Ergebnisse für q , aber v berechnet alle Ergebnisse von q
 - Manche Tupel müssen aus dem Ergebnis von v entfernt werden

40

Query Rewriting

Probleme

- Wir konzentrieren uns auf **Möglichkeit 3**.
- Probleme:
 - **Vollständigkeit:** v enthält alle Tupel - auch die richtigen Attribute?
 - **Ableitbarkeit:** Wie findet man einen Filter F auf v , so dass nur die richtigen Tupel selektiert werden, also $F(v) \equiv q$?
- Um diese Probleme zu lösen, müssen die Bedingungen von Containment Mappings angepasst werden.



Data Warehouses | SS 2011 | Melanie Herschel | Universität Tübingen

41

Query Rewriting

Erweitertes Containment Mapping

Erweitertes Containment Mapping

Ein erweitertes Containment Mapping (CM) h von Anfrage v nach Anfrage q ist ein Symbol Mapping von v nach q für das gilt:

- $\forall c \in \text{const}(v)$ gilt: $h(c) = c$
- $\forall l \in v$ gilt: $h(l) \in q$
- $\forall e \in \text{exp}(q)$ gilt: $\exists e' \in \text{exp}(v)$ mit $h(e') = e$
Der Kopf von q ist im Bild des Kopfes von v enthalten
- $\text{cond}(q) \rightarrow \text{cond}(h(v))$

- In Zukunft verwenden wir CM, für ein erweitertes Containment Mapping.
- Mit dieser Definition hat man nun alle Attribute, aber immer noch zu viele Tupel.
- Wie findet man die richtigen?

Data Warehouses | SS 2011 | Melanie Herschel | Universität Tübingen

42

Query Rewriting

Ableitbarkeit

- Wenn $q \subseteq v$, dann gilt: $q \rightarrow h(v)$
 - h : Containment Mapping von v nach q
 - \rightarrow bezeichnet hier die logische Implikation zwischen Formeln in Prädikatenlogik
- Gesucht: Ausdruck F für den gilt: $q \equiv h(v) \wedge F$
 - Im Allgemeinen unentscheidbar wegen Unentscheidbarkeit der Prädikatenlogik
 - Aber wir betrachten nur konjunktive Anfragen
- Im Folgenden
 - Ableitbarkeit von Bedingungen
 - Ableitbarkeit von Joins
 - Ableitbarkeit von Aggregaten

Query Rewriting

Ableitbarkeit von Bedingungen

- Annahmen
 - $q \subseteq v$ (mit erweitertem CM)
 - q und v beinhalten **dieselben Literale, Joins und Gruppierungen**
- Damit können wir $\text{cond}(q)$ als Filter verwenden
 - Per Definition CM gilt: $\text{cond}(q) \rightarrow \text{cond}(h(v))$, **$\text{cond}(q)$ sind also schärfere Bedingungen**; mit denen müssen wir die Tupel aus v filtern.

Ableitbarkeit von Bedingungen: Beispiel 1

$v(A, B) :- r(A, B, C), B < 40$
 $q(A, B) :- r(A, B, C), B < 30$

- Prüfung der Annahmen:
 - Es gilt $q \subseteq v$
 - Sicht v und Anfrage q sind bis auf die Bedingungen cond identisch.
- Anwendung der schärferen Bedingung von q auf v führt zu neuer Anfrage q' :
 $q'(A, B) :- v, B < 30$

Ableitbarkeit von Bedingungen: Beispiel 2

$v(A, B) :- r(A, B, C), B < 40, C > 60$
 $q(A, B) :- r(A, B, C), B < 30, C > 70$

- Prüfung der Annahmen: OK
- Anwendung der schärferen Bedingung von v auf q möglich?
 $q'(A, B) :- v, B < 30, C > 70$ **NEIN!**
- Wir müssen also auf exportierte Variablen aufpassen.

Query Rewriting

Ableitbarkeit von Bedingungen

Algorithmus zur Ableitbarkeit von Bedingungen

Gegeben

Anfrage q ;
Sicht v ;
 $q \subseteq v$, q und v beinhalten dieselben Literale, Joins und Gruppierungen;
Sei $\text{cond}(q) := \{C_1, C_2, \dots, C_n\}$ ohne Joins;

begin

```
q' := q;  
for all  $C_i \in \text{cond}(q) \wedge h(v) \not\rightarrow C_i$  do  
  if  $C_i$  enthält Variablen, deren Urbild bzgl.  $v$  nicht  
  exportiert ist  
  then return error;  
  else  $q' := q', C_i$   
return  $q'$ ;  
end
```

45

Query Rewriting

Ableitbarkeit von Joins

- Annahmen
 - $q \subseteq v$ mit CM $h: v \rightarrow q$
 - q und v beinhalten **dieselben Bedingungen und Gruppierungen**
 - Aber **unterschiedliche Literale bzw. Joins**
- Potentielle Probleme
 - q enthält Literale, die v nicht enthält
z.B. $q(A, B) :- r(A, B), s(B, C)$ vs. $v(A, B) :- r(A, B)$
 - q enthält Joins, die v nicht enthält
z.B. $q(A, C) :- r(A, B), s(B, C)$ vs. $v(A, C) :- r(A, B_1), s(B_2, C)$
- Einfachste Idee: Wende gesamte Anfrage q auf v an: $q = q(v(D))$, aber schwierig in DBMS zu implementieren → Wir wollen lieber die Anfrage umschreiben

Beispiel zur Ableitbarkeit von Joins

```
v(P, PN, TID) :- s(SID, PID, LID, TID, P), p(PID, PN), l(LID, SN)  
q(P, TID) :- s(SID, PID, LID, TID, P), p(PID, P), l(LID, SN), t(TID, D, M, Y);
```

Es gelten alle Annahmen ($q \subseteq v$, jeweils keine Bedingungen und Gruppierungen, aber unterschiedliche Literale und Joins).

Umgeschriebene Anfrage:

```
q'(P, TID) :- v(P, PN, TID), s(TID, D, M, Y), PN = P
```

46

Query Rewriting

Ableitbarkeit von Joins

Algorithmus zur Ableitbarkeit von Join

Gegeben

Anfrage q , Sicht v ;

$q \subseteq v$ mit containment mapping $h: v \rightarrow q$;

// h bildet per Definition jedes Literal aus v auf **mindestens ein** Literal aus q ab.

begin

$L :=$ Menge der Literale aus q , die nicht im Bild von h sind;

for all $l_i \in L$ **do**

for all Variable V_j in l_i , die als Bild in h enthalten **do**

begin

$X :=$ Variable in v , für die $X \rightarrow V_j$ in h gilt;

if $X \notin \text{exp}(v)$ **then return error;** // nicht kompensierbarer Join

end

$J := \{ (X = Y) \mid (X \rightarrow V) \in h \text{ AND } (Y \rightarrow V) \in h \}$ // Joins in q aber nicht in v

for all $(X = Y) \in J$ **do**

if $X \notin \text{exp}(v) \vee Y \notin \text{exp}(v)$ **then return error;**

return $q' := h(v), L, J$;

end

47

Query Rewriting

Ableitbarkeit von Joins

Beispiel Algorithmusverlauf

$v(P, PN, TID) := s(SID, PID, LID, TID, P), p(PID, PN), l(LID, SN)$

$q(P, TID) := s(SID, PID, LID, TID, P), p(PID, P), l(LID, SN), t(TID, D, M, Y)$;

• $L = \{t\}$

• $(TID \rightarrow TID)$ in h , während D, M und Y nicht im containment mapping auftauchen.
Wir prüfen nur TID , konkret, ob $TID \in \text{exp}(v)$
→ Ja, also kein nicht-kompensierbarer Join

• $J = \{(PN = P)\}$, denn $P \rightarrow P$ und $PN \rightarrow P$ tauchen in containment mapping auf

• Sowohl PN als auch P werden von v exportiert → kein Fehler

• $q' := v, L, J$ also

$q'(P, TID) := v(P, PN, TID), s(TID, D, M, Y), PN = P$

48

Query Rewriting

Ableitbarkeit von Aggregation

- Annahmen
 - q und v haben die Form

```
SELECT G1, G2, ..., Gn, sum(A1)
FROM ...
WHERE ...
GROUP BY G1, G2, ..., Gn
```
 - Ohne Gruppierung / Aggregation soll gelten: $q \equiv v$
Bzw. $q \equiv h(v) \wedge B \wedge L \wedge J$ falls vorangehende Ableitung von Bedingungen und Joins
- Wir betrachten nur die Aggregatsfunktion SUM
Bei anderen Funktionen Anwendbarkeit in Hierarchien beachten (distributiv, algebraisch, holistisch ...)
- G_1, G_2, \dots, G_n heißen **Gruppierungsattribute**. **Nur hier sind Unterschiede zwischen q und v.**
- Welche q kann man unter Verwendung eines gegebenen v (schneller) beantworten?
 - In v sind alle Kombinationen von $G = \{G_1, \dots, G_n\}$ mit Summe vorhanden.
 - Aufsummierung für **jede Untermenge** von G möglich.

49

Query Rewriting

Ableitbarkeit von Aggregation

Beispiel zur Ableitbarkeit von Aggregation

```
CREATE VIEW v AS
SELECT S.PGID, S.SID, T.YID, SUM(amount * price)
FROM Sales S, Time T, ...
WHERE ...
GROUP BY S.PGID, S.SID, T.YID
```

→ Aus dieser Sicht lässt sich z.B. folgende Summe ableiten

```
PGID, SID, SUM(...) mit SELECT S.PGID, S.SID, SUM(...)
FROM v
GROUP BY S.PGID, S.SID
```

→ Aber noch weitere (Ableitung analog)

```
SID, YID, SUM(...)
PGID, SUM(...)
SID, SUM(...)
YID, SUM(...)
SUM(...)
```

50

Query Rewriting

Ableitbarkeit von Aggregation

- Wir haben beobachtet, dass eine Gruppierung G aus einer Gruppierung H ableitbar ist, wenn $G \subseteq H$.
- Ableitbarkeit einer Gruppierung relativ zu einer anderen Gruppierung, unabhängig von funktionalen Abhängigkeiten, lässt sich in einem **Aggregationsgitter** darstellen (siehe nächste Folie).
- Es folgt eine formale Definition, die insb. auch funktionale Abhängigkeiten betrachtet.

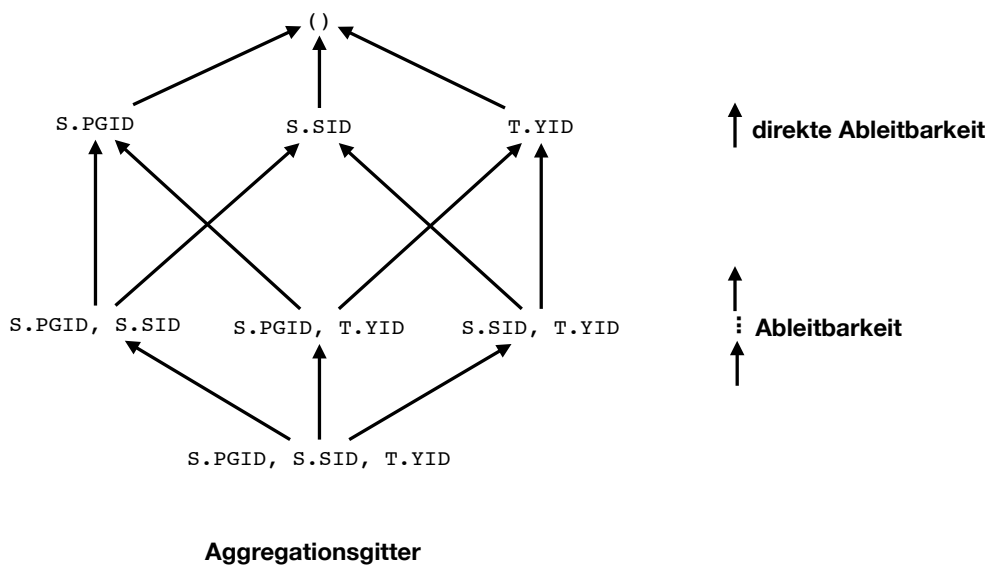
Definition der Ableitbarkeit von Aggregation

- Sei v ein Sicht mit Gruppierungsattributen G_v und q eine Anfrage mit Gruppierungsattributen G_q
- Wir nehmen zudem an, dass q und v äquivalente SELECT-PROJECT-JOIN Klauseln haben.
- q ist **direkt ableitbar** aus v , $v \rightarrow q$, genau dann wenn
 - $G_q \subset G_v$ und $|G_q| = |G_v| - 1$ oder
 - $G_q = G_v \setminus a_x \cup a_y$ mit $a_x \in G_v$, $a_y \notin G_v$ und a_y ist funktional abhängig von a_x
- q ist **ableitbar** aus v , $v \rightarrow^* q$, genau dann wenn
 - $v \rightarrow q$ oder
 - Es existieren Sichten v_1, \dots, v_n , so dass folgende funktionale Abhängigkeiten gelten: $v \rightarrow v_1 \rightarrow \dots \rightarrow v_n \rightarrow q$

51

Query Rewriting

Ableitbarkeit von Aggregation



52

Query Rewriting

Ableitbarkeit von Aggregation

Beispiel zur Ableitbarkeit von Aggregation bei zusätzlichen funktionalen Abhängigkeiten

- Gegeben folgende Sichtdefinition

```
CREATE VIEW v AS
SELECT S.PGID, S.SID, T.DID, SUM(amount * price)
FROM Sales S, Time T, ...
WHERE ...
GROUP BY S.PGID, S.SID, T.YID
```

- Es gelten folgende funktionalen Abhängigkeiten entlang der Time-Dimension
DID → MID → YID
- Zusätzlich zur Potenzmenge der Gruppierungsattribute können wir nun z.B. auch folgende Anfrage ableiten (und natürlich weitere):

```
SELECT S.PGID, S.SID, T.MID, SUM(amount * price)
FROM Sales S, Time T, ...
WHERE ...
GROUP BY S.PGID, S.SID, T.MID
```

Zusammenfassung

- Partitionierung, insb. horizontale Partitionierung zur effizienteren Anfragebearbeitung (bei Joins, durch Parallelisierung, *Partition Pruning*)
- Materialisierte Sichten
 - Existenz einer Anfrageersetzung
 - Query Containment
 - Eigentliche Anfrageumformung
 - Query Rewriting
 - Ableitbarkeit

