

Mobile und Verteilte Datenbank Systeme - Zusammenfassung

Egemen Kaba

Inhaltsverzeichnis

1 Kapitel 0 - Introduction	4
2 Kapitel 1 - Trigger	5
2.1 Zweck	5
2.2 Konzepte	5
2.3 Struktur eines Triggers	5
2.4 Beispiel	7
2.5 Databaselinks	7
3 Kapitel 2 - Distributed Design I	8
3.1 Arten der Fragmentierung	8
3.2 PHF	9
3.2.1 Predicates	9
3.2.2 PHF Beispiel	10
3.3 DHF	10
4 Kapitel 3 - Distributed Design II	10
4.1 VF	10
4.1.1 Anwendungen als Queries	10
4.1.2 [U]sage Matrix	10
4.1.3 [Acc]ess frequency Matrix	11
4.1.4 Affinitätsmatrix AA	11
4.1.5 Bond Energy Algorithmus (BEA)	11
4.1.6 Splitting der Resultatsmatrix BEA	12
4.2 Korrektheit der Fragmentierung	13
5 Kapitel 4 - Distributed Query Processing	13
5.1 Begriffe	13
5.1.1 Komplexität der Operationen	13
5.1.2 Kosten Modell	13
5.2 Methodik	14
5.3 Reduktionen	14
5.3.1 Beispielrelationen	14
5.3.2 PHF mit Selektion	15
5.3.3 PHF mit Join	15
5.3.4 VF	15
5.3.5 DHF	15
6 Kapitel 5 - Distributed Transactions I	16
7 Kapitel 6 - Distributed Transactions II	16
8 Kapitel 7 - Replication I	16
9 Kapitel 8 - Replication II	16
10 Kapitel 9 - NoSQL	16
11 Kapitel 10 - Cassandra	16

12 Kapitel 11 - MapReduce	16
13 Kapitel 12 - mongoDB	16
14 Kapitel 13 - Neo4j	16
15 Kapitel 14 - Semantic Web	16

1 Kapitel 0 - Introduction

Verteilte Datenbank (DDB)	Eine verteilte Datenbank ist eine Sammlung mehrerer, untereinander logisch zusammengehöriger Datenbanken, die über ein Computernetzwerk verteilt sind.
Verteiltes Datenbankverwaltungssystem (D-DBMS)	Ein verteiltes Datenbankverwaltungssystem ist die Software, die die verteilte Datenbank verwaltet und gegenüber den Nutzern einen transparenten Zugang erbringt.
Verteiltes Datenbank System (DDBS)	DDBS = DBS + D-DBMS
Nebenläufigkeit	<ul style="list-style-type: none"> • Synchronisation konkurrierender Transaktionen • Konsistenz und Isolation • Deadlock Erkennung
Zuverlässigkeit	<ul style="list-style-type: none"> • Robustheit gegenüber Fehler • Atomarität und Dauerhaftigkeit
Architekturen	<ul style="list-style-type: none"> • Shared Memory Architecture • Shared Disk Architecture • Shared Nothing Architecture
Mobile Datenbank Systeme	<p>Verteilte Datenbank System mit zusätzlichen Eigenschaften und Einschränkungen</p> <ul style="list-style-type: none"> • beschränkte Ressource • häufig nicht verbunden • verlangt andere Transaktions Modelle • verlangt andere Replikationsstrategien • Ortsabhängigkeit

Date's 12 Regeln

- Lokale Autonomie
- Unabhängigkeit von zentralen Systemfunktionen
- Hohe Verfügbarkeit
- Ortstransparenz
- Fragmentierungstransparenz

- Replikationstransparenz
- Verteilte Anfragebearbeitung
- Verteilte Transaktionsverarbeitung
- Hardware Unabhängigkeit
- Betriebssystem Unabhängigkeit
- Netzwerkunabhängigkeit
- Datenbanksystem Unabhängigkeit

2 Kapitel 1 - Trigger

2.1 Zweck

- Realisieren aktive Datenbanksysteme
- Berechnung abgeleiteter Attribute
- Überprüfen komplexer Integritätsbedingungen
- Implementierung von Geschäftsregeln
- Protokollierung, Statistiken
- Überprüfen von Integritätsbedingungen in verteilten Datenbanken
- Synchronisation von Replikaten

2.2 Konzepte

ECA Prinzip	Event: Ereignis tritt ein Condition : Bedingung ist erfüllt Action: Aktion wird ausgeführt
Ereignis	DML: UPDATE, DELETE, INSERT DDL: CREATE, ALTER, DROP, ... Datenbank: SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN (DML-Trigger können auf Tabellen oder Views definiert werden)
Timing	BEFORE, AFTER, INSTEAD OF Trigger Der INSTEAD OF Trigger ersetzt den triggernden Befehl und wird nur bei Views eingesetzt.
Granulat	STATEMENT, ROW Trigger

2.3 Struktur eines Triggers

Syntax

```
1 CREATE [OR REPLACE] TRIGGER tname
2 {BEFORE|AFTER} events
3 [WHEN(condition)]
4 pl/sql_block
```

events

```
1 {DELETE|INSERT|UPDATE
2   [OF column [ , column ]... ] }
3 [OR {DELETE|INSERT|UPDATE
4   [OF column [ , column ]... ]}]...
5 ON table [FOR EACH ROW]
```

Prinzip

```
1 DECLARE
2   Deklarationsteil
3 BEGIN
4   Programmteil
5 EXCEPTION
6   Ausnahmebehandlung
7 END;
8 /
```

BildschirmAusgabe

```
1 dbms_output.put_line (item IN VARCHAR2);
2 dbms_output.put_line (item IN NUMBER);
3 dbms_output.put_line (item IN DATE);
4 dbms_output.put(item IN VARCHAR2);
5 dbms_output.put(item IN NUMBER);
6 dbms_output.put(item IN DATE);
7 dbms_output.new_line;
8 -- Ausführung
9 EXECUTE dbms_output.put_line('Hello world');
10 -- als Block
11 BEGIN
12   dbms_output.put_line('Hello world');
13 END;
```

Datentypen

- SQL: VARCHAR2, DATE, NUMBER, ...
- PL/SQL: BOOLEAN, PLS.INTEGER, ...
- Strukturierte Datentypen: TABLE, VARRAY, RECORD
- Datentypen für Spalten und Zeilen aus Tabellen: % ROWTYPE, %TYPE

Zuweisung

```
1 -- Syntax
2 variable := expression;
3 -- Beispiel im Deklarationsteil
4 name VARCHAR2(30) := 'Kaba';
```

if then else elsif

```
1 IF condition THEN ... END IF;
2 IF condition THEN ... ELSIF condition THEN ... ELSE ... END IF;
3 IF condition THEN ... ELSE ... END IF;
```

if then else elsif

```
1 IF condition THEN ... END IF;
2 IF condition THEN ... ELSIF condition THEN ... ELSE ... END IF;
3 IF condition THEN ... ELSE ... END IF;
```

schleifen

```
1 WHILE condition LOOP ... END LOOP;
2 FOR counter IN lower_bound..higher_bound LOOP ... END LOOP;
```

2.4 Beispiel

Trigger

```
1 CREATE OR REPLACE TRIGGER regdatum_test
2   BEFORE INSERT ON registrierungen
3   FOR EACH ROW
4
5   DECLARE
6     msg VARCHAR2(30) := 'Datum falsch';
7   BEGIN
8     IF :new.datum > SYSDATE THEN
9       RAISE_APPLICATION_ERROR(-20005, msg);
10    END IF;
11 END;
```

2.5 Databaselinks

Datenbanklinks werden benötigt, um Orts- und Namenstransparenz für Tabellen zu erreichen.

Databaselinks

```
1 -- View
2 CREATE OR REPLACE VIEW filme AS
3 SELECT *
4 FROM filme@ananke.hades.fhnw.ch;
5 -- Synonyme
6 CREATE SYNONYM film FOR filme@ananke.hades.fhnw.ch;
```

3 Kapitel 2 - Distributed Design I

Ausgangslage: Anwendungen auf verschiedenen Knoten des Netzwerks greifen auf eine (relationale) Datenbank zu. Nun greifen nicht alle Knoten gleich häufig auf den selben Datensatz zu. Es gilt nun herauszufinden, welche Anwendungen (Queries) auf welchen Knoten welche Daten mit welcher Häufigkeit benötigen. Das Resultat ist eine Menge von Fragmenten, die den verschiedenen Knoten zugeteilt werden.

3.1 Arten der Fragmentierung

- Horizontale Fragmentierung (HF) (Abbildung 1)
 - Primäre horizontale Fragmentierung (PHF)
 - Abgeleitete horizontale Fragmentierung (DHF)
- Vertikale Fragmentierung (VF) (Abbildung 2)
- Gemischte Fragmentierung (MF)

BIKES

BNr	BName	Preis	Typ	Bestand
B5	MCD03	4490.00	Road	2
B4	Siena	2390.00	Mountain	4
B2	City Cross	2190.00	Trekking	3
B3	Valiant	1090.00	Trekking	7
B1	Luxor	980.00	City	10
B6	Atlanta	890.00	Trekking	8
B7	Striker	890.00	Mountain	7

```
SELECT *
FROM bikes
WHERE preis < 2000
```

BIKES1

BNr	BName	Preis	Typ	Bestand
B3	Valiant	1090.00	Trekking	7
B1	Luxor	980.00	City	10
B6	Atlanta	890.00	Trekking	8
B7	Striker	890.00	Mountain	7

```
SELECT *
FROM bikes
WHERE preis >= 2000
```

BIKES2

BNr	BName	Preis	Typ	Bestand
B5	MCD03	4490.00	Road	2
B4	Siena	2390.00	Mountain	4
B2	City Cross	2190.00	Trekking	3

Abbildung 1: Horizontale Fragmentierung

BIKES

BNr	BName	Preis	Typ	Bestand
B5	MCD03	4490.00	Road	2
B4	Siena	2390.00	Mountain	4
B2	City Cross	2190.00	Trekking	3
B3	Valiant	1090.00	Trekking	7
B1	Luxor	980.00	City	10
B6	Atlanta	890.00	Trekking	8
B7	Striker	890.00	Mountain	7

**SELECT bnr, bname, preis
FROM bikes**

BIKES1

BNr	BName	Preis
B5	MCD03	4490.00
B4	Siena	2390.00
B2	City Cross	2190.00
B3	Valiant	1090.00
B1	Luxor	980.00
B6	Atlanta	890.00
B7	Striker	890.00

**SELECT bnr, typ, bestand
FROM bikes**

BIKES2

BNr	Typ	Bestand
B5	Road	2
B4	Mountain	4
B2	Trekking	3
B3	Trekking	7
B1	City	10
B6	Trekking	8
B7	Mountain	7

Abbildung 2: Vertikale Fragmentierung

3.2 PHF

3.2.1 Predicates

Simple predicates	Vergleich eines Attributs mit einem Wert (WHERE-Klausel)	$p_1: \text{Typ} = \text{'Road'}$ $p_2: \text{Typ} = \text{'Trekking'}$ $p_3: \text{Typ} = \text{City}$ $p_4: \text{Typ} = \text{'Mountain'}$ $p_5: \text{Preis} \leq 2000$ $p_6: \text{Preis} > 2000$
Minterm predicates	Verknüpfung von Simple predicates mit AND und NOT	$m_1: \text{Typ} = \text{'Road'} \text{ AND } \text{Preis} \leq 2000$ $m_2: \text{NOT}(\text{Typ} = \text{'Road'}) \text{ AND } \text{Preis} \leq 2000$ $m_3: \text{Typ} = \text{'Road'} \text{ AND NOT}(\text{Preis} \leq 2000)$ $m_4: \text{NOT}(\text{Typ} = \text{'Road'}) \text{ AND NOT}(\text{Preis} \leq 2000)$...
Vollständigkeit	Eine Menge von simple predicates ist vollständig genau dann, wenn auf beliebige 2 Tupel im gleichen Fragment von allen Anwendungen mit der gleichen Häufigkeit zugegriffen wird	
Minimalität	Wird durch ein simple predicate ein Fragment weiter aufgeteilt, dann muss es mindestens eine Anwendung geben, die auf diese Fragmente verschieden zugreift. Ein simple predicate soll also relevant sein für die Bestimmung einer Fragmentierung. Sind alle simple predicate einer Menge P relevant, dann ist P minimal	

3.2.2 PHF Beispiel

Anwendung	Query	Parameter	Simple Predicates
Anwendung 1	SELECT bname, bestand FROM bikes WHERE typ = ?	City (4/Woche) Trekking (3/Woche) Mountain (2/Woche) Road (1/Woche)	p1: Typ = 'Road' p2: Typ = 'Mountain' p3: Typ = 'Trekking' p4: Typ = 'City'
Anwendung 2	SELECT * FROM bikes WHERE preis = ?	<2000 (3/Woche) ≥2000 (1/Woche)	p5: Preis < 2000 p6: Preis ≥ 2000

Sinnvolle minterm predicates bilden. Zum Beispiel: m1: Typ = 'Road' AND Preis < 2000.

3.3 DHF

Horizontale Fragmentierung auf einer übergeordneten horizontal fragmentierten Relation. Dadurch soll sichergestellt werden, dass auf häufig im Verbund zugegriffene Relationen auf dem selben Knoten liegen.

Falls die Tabelle KUNDEN nun in KUNDEN1 - KUNDEN4 fragmentiert werden, sieht die Fragmentierung der Tabelle AUFTRAEGE wie folgt aus: $AUFTRAEGE_i = (AUFTRAEGE) \times (KUNDEN_i)$

4 Kapitel 3 - Distributed Design II

4.1 VF

4.1.1 Anwendungen als Queries

q1	q1
SELECT bestand	SELECT bestand, preis
FROM bikes WHERE bname = ?	FROM bikes
q3	q4
SELECT preis	SELECT AVG(bestand)
FROM bikes WHERE typ = ?	FROM bikes WHERE typ = ?

4.1.2 [U]sage Matrix

	BName	Preis	Typ	Bestand
q1	1	0	0	1
q2	0	1	0	1
q3	0	1	1	0
q4	0	0	1	1

1 = Query verwendet Attribut

0 = Query verwendet Attribut nicht

4.1.3 [Acc]ess frequency Matrix

	S1	S2	S3
q1	15	20	10
q2	5	0	0
q3	25	25	25
q4	3	0	0

Frage: Wie viel mal wird ein Query auf einem Knoten ausgeführt?

Da jetzt jedes Attribut ein Fragment bilden würde, muss jetzt nach einer Attributsmenge gesucht werden, auf die ähnlich zugegriffen wird.

4.1.4 Affinitätsmatrix AA

	BName	Preis	Typ	Bestand
BName	45	0	0	45
Preis	0	80	75	5
Typ	0	75	78	3
Bestand	45	5	3	53

Vorgehen: In der Access frequency Matrix Summer über jedes Query bilden (z.B. $q1 = 45$). Funktion $aff(A_i, A_j)$ bestimmen durch Folgendes:

- Zeilen/Queries in der Usage Matrix suchen, die in diesen beiden Spalten eine 1 stehen haben.
- Summen dieser Queries aus der Access frequency Matrix zusammenzählen.
- In die Affinitätsmatrix in der Zeile/Spalte A_i und der Spalte/Zeile A_j eintragen.

4.1.5 Bond Energy Algorithmus (BEA)

Dieser Algorithmus maximiert die globale Affinität einer Affinitätsmatrix.

Die globale Affinität einer Matrix zu berechnen, muss die Funktion $bond(A_x, A_y)$ für alle benachbarten Attribute ausgeführt werden.

Vorgehen Funktion $bond(A_x, A_y)$:

- Über sämtliche Zeilen iterieren
- Für jede Zeile den Eintrag in der Spalte A_x mit dem Eintrag in der Spalte A_y multiplizieren
- Summe über diese Werte bilden

BEA

- Gegeben $n \times n$ Matrix AA der Affinitäten
- Beliebige 2 Spalte aus AA wählen und in Resultats Matrix CA stellen
- Iteration:
 - eine der übrigen $n - i$ Spalten so in Resultats Matrix positionieren ($i + 1$ mögliche Positionen), dass sich der grösste Beitrag an die globale Affinität der Nachbarschaft ergibt
 - Die Zeilen entsprechend den Spalten anordnen

Beitrag einer Spalte A_k wenn zwischen A_i und A_j platziert:

$$cont(A_i, A_k, A_j) = bond(A_i, A_k) + bond(A_k, A_j) - bond(A_i, A_j)$$

4.1.6 Splitting der Resultatsmatrix BEA

	Bname	Bestand	Preis	Typ
BName	45	45	0	0
Bestand	45	53	5	3
Preis	0	5	80	75
Typ	0	3	75	78

Splitting mit Trennpunkt entlang der Diagonale führt

zu drei Varianten:

- VF1: BName; VF2: Bestand,Preis,Typ
- VF1: BName,Bestand; VF2: Preis,Type
- VF1: BName,Bestand,Preis VF2: Typ

Die Variante mit der höchsten Trennqualität muss nun bestimmt werden.

Formel Trennqualität: $sq = acc(VF1) * acc(VF2) - acc(VF1, VF2)^2$

Vorgehen, um die Trennqualität für eine Variante zu bestimmen:

In der [Acc]ess frequency Matrix Summe über jedes Query bilden (z.B. $q1 = 45$).

Vorgehen Funktion acc:

- Zeilen/Queries in der Usage Matrix suchen, die **nur** in Spalten der Fragmentierung eine 1 und in den restlichen eine 0 stehen haben.
- Summen dieser Queries aus der Access frequency Matrix zusammenzählen.

Zum Schluss Fragmente in relationaler Algebra zum Beispiel wie folgt definieren:

BIKES1: $\pi_{BName, Bestand}(BIKES)$

BIKES2: $\pi_{Preis, Typ}(BIKES)$

4.2 Korrektheit der Fragmentierung

vollständig	Wenn R zerlegt wird in R1, R2, ..., Rn, dann muss jedes Datenelement aus R in einem Ri enthalten sein.
rekonstruierbar	Wenn R zerlegt wird in R1, R2, ..., Rn, dann muss es relationale Operatoren geben, so dass R wiederhergestellt werden kann.
disjunkt	Wenn R horizontal zerlegt wird in R1, R2, ..., Rn, dann müssen die Fragmente paarweise disjunkt sein. Wenn R vertikal zerlegt wird in R1, R2, ..., Rn, dann müssen die Fragmente bezogen auf die nichtprimen Attribute paarweise disjunkt sein.

5 Kapitel 4 - Distributed Query Processing

5.1 Begriffe

5.1.1 Komplexität der Operationen

σ, π (mit Duplikate)	$O(n)$
π (ohne Duplikate), GROUP	$O(n \log n)$
$\bowtie, \div, \cup, \cap$	$O(n \log n)$
\times	$O(n^2)$

5.1.2 Kosten Modell

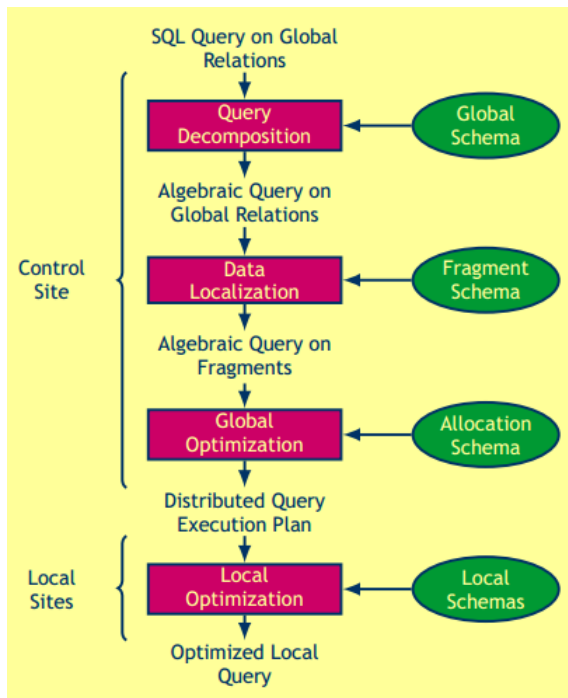
Gesamtzeit für Verbesserung des Durchsatzes.

$C_{CPU} \cdot \text{Anzahl Instruktionen} + C_{I/O} \cdot \text{Anzahl Disk I/O} + C_{MSG} \cdot \text{Anzahl Meldungen} + C_{TR} \cdot \text{übertragene Bytes}$

Antwortzeit, um die Antwortzeit der Anfrage zu reduzieren

$\max(TC_1, TC_2, \dots, TC_n)$; ein TCi: Gesamtkosten eines Thread der parallel ausgeführten Anfrage

5.2 Methodik



5.3 Reduktionen

5.3.1 Beispielrelationen

AUF(ANr, Datum, KNr) ist fragmentiert:

$$\text{AUF1} = \sigma_{\text{ANr} \leq \text{A3}}(\text{AUF})$$

$$\text{AUF2} = \sigma_{\text{A3} < \text{ANr} \leq \text{A6}}(\text{AUF})$$

$$\text{AUF3} = \sigma_{\text{ANr} > \text{A6}}(\text{AUF})$$

$$\text{AUF} = \text{AUF1} \cup \text{AUF2} \cup \text{AUF3}$$

APOSTEN(ANr, BNr, Menge) ist fragmentiert:

$$\text{APO1} = \sigma_{\text{ANr} \leq \text{A3}}(\text{APO})$$

$$\text{APO2} = \sigma_{\text{ANr} > \text{A3}}(\text{APO})$$

$$\text{APO} = \text{APO1} \cup \text{APO2}$$

KUNDEN(KNr, KName, Ort) ist fragmentiert:

$$\text{KUN1} = \pi_{\text{KNr}, \text{KName}}(\text{KUN})$$

$$\text{KUN2} = \pi_{\text{KNr}, \text{Ort}}(\text{KUN})$$

$$\text{KUN} = \text{KUN1} \bowtie \text{KUN2}$$

Für Reduktion in DHF:

KUNDEN(KNr, KName, Ort) ist fragmentiert:

$$\text{KUN1} = \sigma_{\text{Ort} = \text{Basel}}(\text{KUN})$$

$$\text{KUN2} = \sigma_{\text{Ort} \neq \text{Basel}}(\text{KUN})$$

$$\text{KUN} = \text{KUN1} \cup \text{KUN2}$$

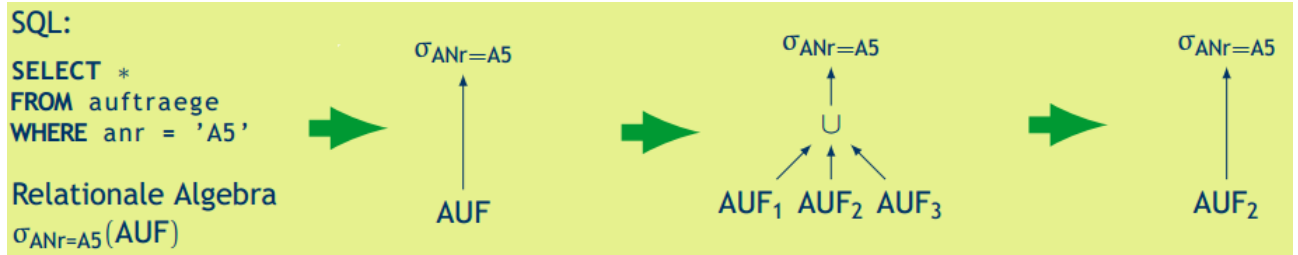
AUFRAEGE(ANr, Datum, KNr) ist abgeleitet fragmentiert:

$$\text{AUF1} = \text{AUF} \bowtie \text{KUN1}$$

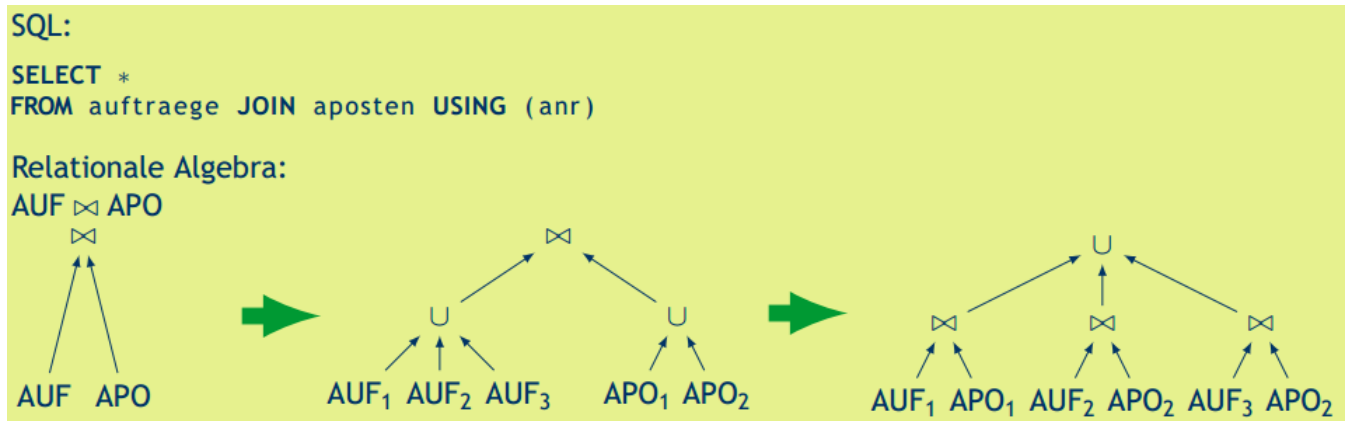
$$\text{AUF2} = \text{AUF} \bowtie \text{KUN2}$$

$$\text{AUF} = \text{AUF1} \cup \text{AUF2}$$

5.3.2 PHF mit Selektion



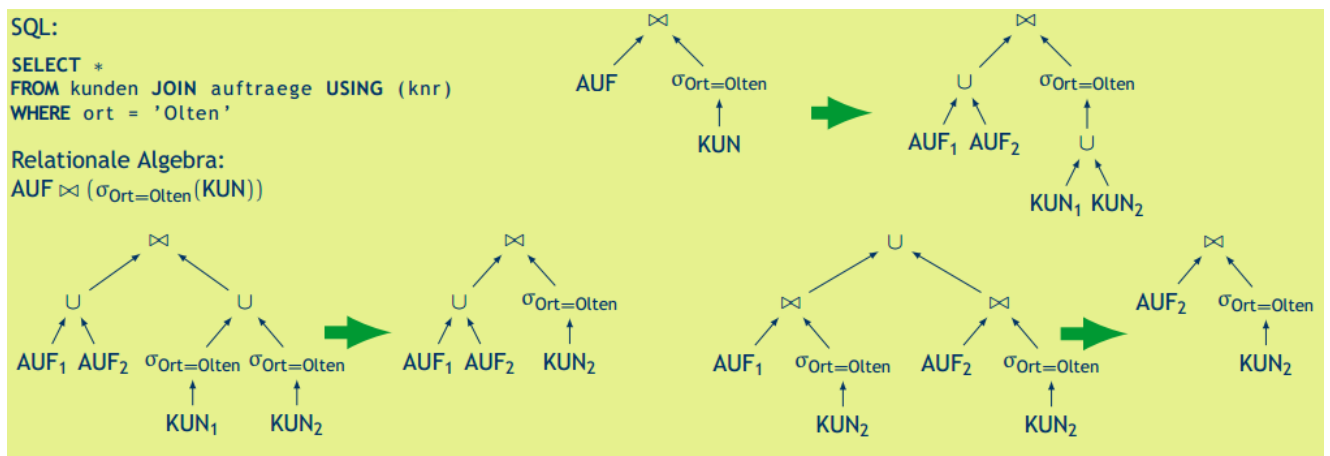
5.3.3 PHF mit Join



5.3.4 VF



5.3.5 DHF



- 6 Kapitel 5 - Distributed Transactions I**
- 7 Kapitel 6 - Distributed Transactions II**
- 8 Kapitel 7 - Replication I**
- 9 Kapitel 8 - Replication II**
- 10 Kapitel 9 - NoSQL**
- 11 Kapitel 10 - Cassandra**
- 12 Kapitel 11 - MapReduce**
- 13 Kapitel 12 - mongoDB**
- 14 Kapitel 13 - Neo4j**
- 15 Kapitel 14 - Semantic Web**