



# Kommunikation und Datenhaltung

## Anfrageoptimierung



# Überblick über den Datenhaltungsteil

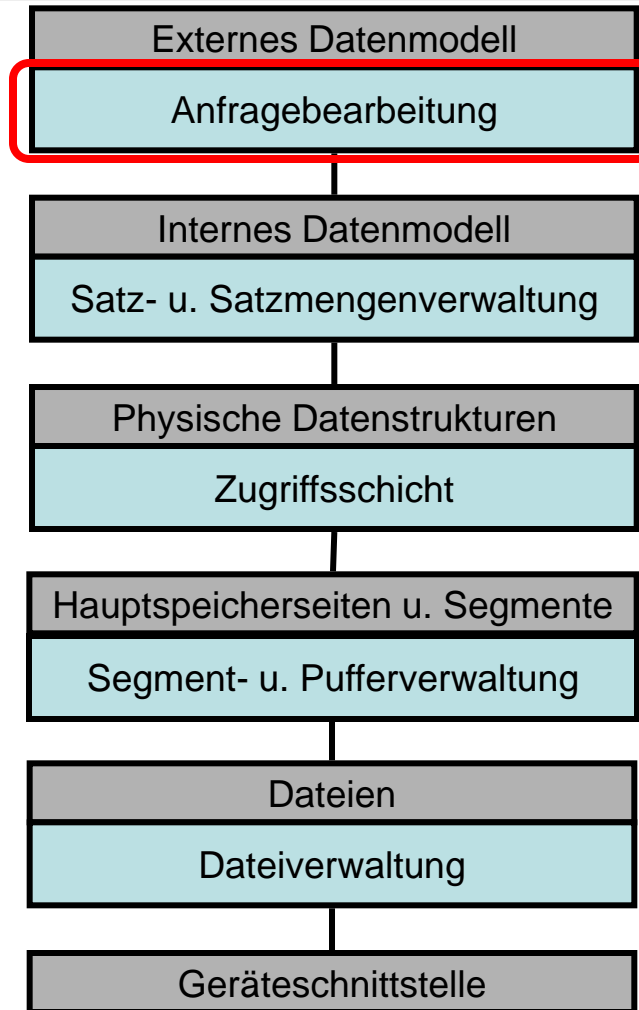
- Einleitung
  - Motivation und Grundlagen
  - Architektur von Datenbanksystemen
- Datenbankabfragen
  - Relationenmodell und Relationenalgebra
  - Relationale Datenbanksprachen (SQL)
- Datenbankentwurf
  - ER- und EER-Modell
  - Abbildung von ER-Modellen auf das Relationenmodell
  - Relationaler Entwurf
  - Sprachen zur Datenbankdefinition
- Transaktionsverwaltung
- Anfrageoptimierung
- Datenbankanwendungsentwicklung



# Agenda

- **Grundlage: Implementierung von Operatoren**
- Motivation: Kosten alternativer Ausführungen
- Anfragebearbeitung
  - Übersetzung von SQL-Anfragen
  - Optimierung

# Referenzarchitektur



# Das Interne Datenmodell (1)

- Schnittstelle, auf der die Anfragebearbeitung aufsetzt (s. Referenzarchitektur)
- Struktur:
  - Datensatz  $\approx$  Tupel,
  - Datei  $\approx$  Relation
  - Neuer atomarer Typ *tid*
    - Satzidentifikator (systemintern konstruiert)
    - Dient der Ermittlung des physischen Speicherplatzes
- Operatoren:
  - Operatoren auf einzelnen Datensätzen (**Navigationsoperatoren**).
  - Iteratoren, zur Aufzählung von Satzmenge



## Das Interne Datenmodell (2)

<b>Atomare Typen</b>	char, string(*), int, real, date, time, tid
<b>Typkonstruktoren</b>	<b>datensatz</b> ::= [sel <sub>1</sub> :atomarerTyp, ..., sel <sub>n</sub> :atomarerTyp] <b>datei</b> ::= {datensatz}
<b>Operatoren auf datei</b>	boolean isEmpty() boolean contains(in datensatz element) <b>void</b> insert(in datensatz element) <b>void</b> remove(in datensatz element) <b>datensatz</b> getKey(in sel, in atomarer Typ sk) <b>datensatz</b> get(in tid id) Iterator createIterator()
<b>Operatoren auf Iterator</b>	boolean hasNext() <b>void</b> reset() <b>datensatz</b> get() <b>void</b> next() <b>void</b> replace(in datensatz element)
<b>Konsistenzbedingung</b>	key ::= <b>datei</b> × (sel <sup>n</sup> → atomarerTyp <sup>n</sup> ) → datensatz



# Implementierung der relationalen Operatoren

Operator-Impl.  
Motivation  
Anfragebearb.

- Anfragebearbeitung muss relationale Operatoren (mengenbasierter Zugriff) implementieren auf navigierende Such- und Zugriffsoperationen des internen Datenmodells (Zugriff auf einzelne Datensätzen)
- Kritisch ist vor allem Implementierung der binären Operatoren (Verbindung, Vereinigung, Durchschnitt, Differenz), da potenziell quadratische Laufzeit.

# Selektion

- Sequenzielle Selektion

- Schleife über alle Tupel der Relation.
- Aufwand  $O(|R|)$ , wobei R Eingaberelation.

- Indexbasierte Selektion

- Bei einfacher Selektionsbedingung (Attribut<sub>i</sub>  $\theta$  Konstante) kann Direktzugriff erfolgen, sofern wertbasierter Zugriff über Attribut<sub>i</sub> unterstützt wird
- Aufwand:
  - $O(\log|R|)$  + Anzahl der Ergebnistupel
  - Bei Selektion auf Gleichheit und Einsatz einer Hashtabelle als Indexstruktur sogar  $O(|I|)$  + Anzahl der Ergebnistupel.
- Bei booleschen Kombinationen von Vergleichen
  - Getrennte Auswertung der einzelnen Vergleiche
  - Anschließend Bildung des Durchschnitts bzw. der Vereinigung.

Operator-Impl.  
Motivation  
Anfragebearb.



# Projektion

- Sequenzieller Durchlauf, Aufwand  $O(|R|)$ .
- Bei Duplikat-Elimination (**select distinct**) ist anschließender Sortierlauf und weiterer sequenzieller Durchlauf zur Duplikatentfernung notwendig, Aufwand in diesem Fall  $O(R \log|R|)$ .

Operator-Impl.  
Motivation  
Anfragebearb.

# Geschachtelte Verbindung (Nested Loop) (1)

- Geschachtelte Schleife, in der jeder Datensatz in Datei R (hier: BUCHUNG) mit jedem Datensatz in S (hier: FLUG) verglichen wird:

Operator-Impl.  
Motivation  
Anfragebearb.

```
Iterator bu = BUCHUNG.createliterator( );  
Tupel tb, tf;  
while bu.hasNext( ) {  
    tb = bu.get( );  
    bu.next( );  
    Iterator fl = FLUG.createliterator( );  
    while fl.hasNext( ) {  
        tf = fl.get( );  
        fl.next( );  
        if tb.flugNr = tf.flugNr then ERG.insert( tb $\circ$ tf );  
    }  
}
```

# Geschachtelte Verbindung (Nested Loop) (2)

- Aufwand:

- Zahl der Satzzugriffe:  $|R| + (|R| * |S|) = |R| * (1 + |S|)$ .
- Seien auf einer Seite  $n_R$  bzw.  $n_S$  Sätze untergebracht. Dann Zahl der Hintergrundspeicherzugriffe  $|R|/n_R * (1 + |S|/n_S)$ .
- R sollte daher die kleinere Relation sein.

Operator-Impl.  
Motivation  
Anfragebearb.

# Geschachtelte Verbindung mit Direktzugriff

- Geschachtelte Schleife, in der für jeden Datensatz in R der oder die passenden Datensätze in S anhand eines Index aufgefunden werden:

Operator-Impl.  
Motivation  
Anfragebearb.

```
Iterator bu = BUCHUNG.createIterator( );
```

```
Tupel tb, tf;
```

```
while bu.hasNext( ) {
```

```
    tb = bu.get ( );
```

```
    tf := FLUG.getKey ( flugNr, tb.flugNr );
```

```
    if erfolgreich then ERG.insert ( tb $\circ$ tf );
```

```
    bu.next ( );
```

```
}
```



# Geschachtelte Verbindung mit Direktzugriff

- Aufwand:

- Unterliegende Datenstruktur für S muss wertbasierten Zugriff für die in die Verbindung eingehenden Attribute unterstützen.
- Aufwand  $O(|R|)$  (Hash) oder  $O(|R| \log |S|)$  ( $B^*$ -Baum).
- R (äußere Schleife) sollte daher die kleinere Relation sein.
- Hash-Join: Zwischenrelation nach einem Hashverfahren erstellen, falls schneller wertbasierter Zugriff noch nicht verwirklicht

Operator-Impl.  
Motivation  
Anfragebearb.

# Misch-Verbindung (1)

- Durchlaufen von R und S nach dem Reißverschlussprinzip, mit R und S aufsteigend nach Join-Attributen sortiert:

Operator-Impl.  
Motivation  
Anfragebearb.

```
Iterator fl = FLUG.createliterator(), bu = BUCHUNG.createliterator();
```

```
Tupel tb, tf;
```

```
while fl.hasNext() and bu.hasNext() {
```

```
    tf = fl.get(); tb = bu.get();
```

```
    if tf.flugNr == tb.flugNr {
```

```
        ERG.insert(tb $\circ$ tf);
```

```
        bu.next();
```

```
        tb = bu.get();
```

```
    }
```

```
    else if tf.flugNr < tb.flugNr {
```

```
        tf.next();
```

```
        tf = fl.get();
```

```
    }
```

```
    else if tf.flugNr > tb.flugNr {
```

```
        tb.next();
```

```
        tb = bu.get();
```

```
    }
```

## Misch-Verbindung (2)

- Aufwand:
  - $O(|R| + |S|)$ .
  - Seien auf einer Seite  $n_R$  bzw.  $n_S$  Sätze untergebracht. Dann Zahl der Hintergrundspeicherzugriffe  $|R|/n_R + |S|/n_S$ .
  - Sort-merge join: Vorgeschalteter Sortierlauf.

Operator-Impl.  
Motivation  
Anfragebearb.



# Agenda

- Grundlage: Implementierung von Operatoren
- **Motivation: Kosten alternativer Ausführungen**
- Anfragebearbeitung
  - Übersetzung von SQL-Anfragen
  - Optimierung

# Beispiel

## Relationen:

KUNDE { KName, Kadr, Kto }

AUFTRAG { KName, Ware, Menge }

Operator-Impl.  
Motivation  
Anfragebearb.

## Anfrage:

*SELECT kunde.KName, Kto*

*FROM kunde, auftrag*

*WHERE kunde.KName = auftrag.Kname AND Ware = 'Kaffee'*

## In relationaler Algebra:

$(\pi_{PROJ}(\sigma_{Ware='Kaffee' \wedge KName = KName} (KUNDE \times AUFTRAG)))$

- Relation KUNDE: 100 Tupel; eine Seite: 5 Tupel
- Relation AUFTRAG: 10.000 Tupel; eine Seite: 10 Tupel
- 50 der Aufträge betreffen Kaffee
- Tupel der Form (KName, Kto): 50 auf eine Seite
- 3 Zeilen von KUNDE  $\times$  AUFTRAG auf eine Seite

# Direkte Auswertung

$\pi_{\text{PROJ}}(\sigma_{\text{Ware}='Kaffee' \wedge \text{KName} = \text{KName}} (\text{KUNDE} \times \text{AUFTRAG}))$

Operator-Impl.  
Motivation  
Anfragebearb.

1.  $R_1 := \text{KUNDE} \times \text{AUFTRAG}$ 
  - Seitenzugriffe:
  - $l : (100/5 * 10.000/10) = 20.000$
  - $s : (100 * 10.000)/3 = 333.000$  (ca.)
2.  $R_2 := \sigma_{\text{Ware}='Kaffee' \wedge \text{KName} = \text{KName}} (R_1)$ 
  - $l : 333.000$  (ca.)
  - $s : 50/3 = 17$  (ca.)
3.  $\text{ERG} := \pi_{\text{PROJ}}(R_2)$ 
  - $l : 17$
  - $s : 1$

Insgesamt ca. 687.000 Seitenzugriffe und ca. 333.000 Seiten zur Zwischenspeicherung

# Optimierte Auswertung

$\pi_{\text{PROJ}}(\sigma_{\text{Ware}='Kaffee' \wedge \text{KName} = \text{KName}} (\text{KUNDE} \times \text{AUFTRAG}))$

Operator-Impl.  
Motivation  
Anfragebearb.

1.  $R_1 := \sigma_{\text{Ware}='Kaffee'}(\text{AUFTRAG})$ 
  - I: 10.000/10 = 1.000
  - s: 50/10 = 5
2.  $R_2 := \text{KUNDE} \triangleright \triangleleft_{\text{KName}=\text{KName}} R_1$ 
  - I: 100/5 \* 5 = 100
  - s: 50/3 = 17
3.  $\text{ERG} := \pi_{\text{PROJ}}(R_2)$ 
  - I: 17
  - s: 1

ca. 1.140 Seitenzugriffe  
(Faktor 500 verbessert)

# Auswertung mit Indexausnutzung

$\Pi_{\text{PROJ}}(\sigma_{\text{Ware}='Kaffee'}(\text{KUNDE} \times \text{AUFTRAG}))$

Operator-Impl.  
Motivation  
Anfragebearb.

Indexe  $I(\text{AUFTRAG}(\text{Ware}))$  und  $I(\text{KUNDE}(\text{KName}))$

1.  $R_1 := \sigma_{\text{Ware}='Kaffee'}(\text{AUFTRAG})$  über  $I(\text{AUFTRAG}(\text{Ware}))$ 
  - $l$  : minimal 5, maximal 50;  $s$  :  $50/10 = 5$
2.  $R_2 := \text{sortiere } R_1 \text{ nach KName}$ 
  - $l + s$  :  $5 * \log 5 = 15$  (ca.)
3.  $R_3 := \text{KUNDE} \triangleright \triangleleft_{\text{KName}=\text{KName}} R_2$ 
  - $l$  :  $100/5 + 5 = 25$ ;  $s$  :  $50/3 = 17$
4.  $\text{ERG} := \Pi_{\text{PROJ}}(R_3)$ 
  - $l$  : 17;  $s$  : 1

maximal ca. 130 und minimal ca. 85 Seitenzugriffe

# Gegenüberstellung der Varianten

Operator-Impl.  
Motivation  
Anfragebearb.

Variante der Ausführung	Lese- und Schreibzugriffe	Seiten für Zwischenergebnisse
Direkte Auswertung	ca. 687.000	ca. 333.000
Optimierte Auswertung	ca. 1.140	17
Auswertung mit Index	ca. 85 - 130	17



# Agenda

- Grundlage: Implementierung von Operatoren
- Motivation: Kosten alternativer Ausführungen
- **Anfragebearbeitung**
  - **Übersetzung von SQL-Anfragen**
  - Optimierung

# Phasen der Anfrageverarbeitung I

## 1. *Übersetzung und Sichtexpansion*

- Übersetzung in relationalalgebraischen Ausdruck
  - Da Grammatik von SQL ziemlich komplex ist, werden Ausdrücke zunächst standardisiert (in bestimmte Standardformen überführt).
  - Unteranfragen auflösen
  - Einsetzen der Sichtdefinition
  - Für Standardformen erfolgt Übersetzung

## 2. Algebraische (*logische*) Optimierung

- Anfrageplan unabhängig von der konkreten Speicherungsform umformen; etwa Hineinziehen von Selektionen in andere Operationen

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung



# Phasen der Anfrageverarbeitung II

## 3. Nicht-algebraische *Optimierung*

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

- Konkrete Speicherungstechniken (Indexe, Cluster) berücksichtigen
- Algorithmen auswählen
- Mehrere alternative Pläne
- Kostenbasierte Auswahl
  - Statistikinformationen (Größe von Tabellen, Selektivität von Attributen) für die Auswahl eines konkreten internen Planes nutzen

## 4. *Code-Erzeugung*

- Umwandlung des Zugriffsplans in ausführbaren Code

# Grundmuster der SQL-Übersetzung

- Select-Ausdruck

```
select   $A_1, A_2, \dots, A_n$   
from     $R_1, R_2, \dots, R_m$   
where    $B$ 
```

wird überführt in:

$$\pi_{A_1, A_2, \dots, A_n} (\sigma_B (R_1 \times R_2 \times \dots \times R_m)) .$$

- Probleme:

- $B$  kann geschachtelte Anfragen enthalten.
- $R_i$  kann ein Tabellenausdruck sein, der von  $R_1, \dots, R_{i-1}$  abhängt.
- **group by** und **having**-Klauseln müssen berücksichtigt werden.
- $A_1, A_2, \dots, A_n$  können Aggregatfunktionen sein.

Operator-Impl.  
Motivation  
Anfragebearb.  
**Übersetzung**  
Optimierung

# Behandlung geschachtelter Anfragen (1)

- Grundsätzliches Vorgehen durch Standardisierung auf **select-from-where**:

- Ersetze allgemeinen Tabellenausdruck T in where-Klausel
- durch

```
select *  
from (T) as R(A1,A2,...,An) ,
```

- wobei R ein frei gewählter, sonst nirgendwo vorkommender Name ist und A<sub>1</sub>, A<sub>2</sub>,..., A<sub>n</sub> Namen für die Spalten von T sind.

Operator-Impl.  
Motivation  
Anfragebearb.  
**Übersetzung**  
Optimierung



## Behandlung geschachtelter Anfragen (2)

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

- Rückführung von **in/not in**-Bedingungen auf **=any** bzw. **<>all**.
- Rückführung von  $\theta$ **any** und  $\theta$ **all** auf **exists** bzw. **not exists**:
  - Beispiel: Ersetze  $x \theta$  **any** (**select** A **from**  $R_1, \dots, R_m$  **where** B)
  - durch **exists** (**select** \* **from**  $R_1, \dots, R_m$  **where** B **and**  $x \theta$  A) .

## Behandlung geschachtelter Anfragen (3)

- Rückführung von **not exists** auf **exists**:

- Ersetze

```
select A1, A2, ..., An from R1, ..., Rm where B and not  
exists (T)
```

- durch

```
select A1, A2, ..., An  
from ((select * from R1, ..., Rm where B)  
except  
(select * from R1, ..., Rm where B and exists  
(T)))
```

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

# Behandlung geschachtelter Anfragen (4)

- Eliminieren von **exists**:

- Ersetze

```
select A1, A2, ..., An
from R1, ..., Rm
where B
and exists (select * from Rm+1, ..., Rk where B')
```

- durch

```
select A1, A2, ..., An
from R1, ..., Rm, Rm+1, ..., Rk
where B and B'.
```

- Wenig überraschende Erkenntnis: Geschachtelte Anfragen werden eigentlich nicht gebraucht, die Äquivalenz zu ungeschachtelten Anfragen wurde schon früher exemplarisch gezeigt.

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung



# Behandlung geschachtelter Anfragen (5)

Operator-Impl.  
Motivation  
Anfragebearb.  
**Übersetzung**  
Optimierung

- Nicht angesprochen: Tabellenausdrücke in from-Klausel können von weiter links stehenden Tabellenausdrücken abhängen. Eliminierung folgt ähnlichen Regeln (komplizierter)
- Sukzessive Anwendung der Regeln führt schließlich auf einfache Anfragen, d.h. Anfragen der Form:
  - einzelne Basistabelle,
  - einzelnen ungeschachtelten **select**-Ausdruck (evt. mit Gruppierung), dessen **from**-Klausel wieder eine einfache Anfrage ist,
  - relationale Verknüpfung (**union**, **intersect**, **except**, **join**) einfacher Anfragen.

## Einfache Anfragen (2)

- Einfache Anfragen können rekursiv in relationale Algebra übersetzt werden:
  - Basistabellen werden durch Referenzen auf die entsprechenden internen Dateien ersetzt.
  - Eine Folge einfacher Anfragen in einer **from**-Klausel wird in das Kartesische Produkt überführt.
  - **select**-Ausdruck (evt. mit Gruppierung) wird in Kombination aus Selektions- und Projektionsoperator übersetzt, die auf die übersetzte **from**-Klausel angewendet werden.
  - Gruppierung wird dabei als zusätzlicher Operator  $\gamma$  übersetzt, da in der relationalen Algebra nicht direkt abbildbar.
  - Relationale Verknüpfung (**union**, **intersect**, **except**, **join**) einfacher Anfragen wird in entsprechende relationale Operatoren übersetzt.

Operator-Impl.  
Motivation  
Anfragebearb.  
**Übersetzung**  
Optimierung

# Übersetzung einfacher Anfragen: Beispiel

- Beispiel:

```
select    flugNr, nach, abflugszeit,  
           ankunftszeit, ticketNr, datum  
from      FLUG, BUCHUNG  
where     von = "FRA"  
and FLUG.flugNr = BUCHUNG.flugNr;
```

- Die Anfrage ist bereits einfach und kann geradewegs in den algebraischen Ausdruck

$$\pi_{flugNr, nach, abflugszeit, ankunftszeit, ticketNr, datum} \left( \sigma_{von="FRA" \wedge BUCHUNG.flugNr = FLUG.flugNr} (BUCHUNG \times FLUG) \right)$$

übersetzt werden.

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung



# Agenda

- Grundlage: Implementierung von Operatoren
- Motivation: Kosten alternativer Ausführungen
- **Anfragebearbeitung**
  - Übersetzung von SQL-Anfragen
  - **Optimierung**

# Anfrageoptimierung (1)

- Konkretes Vorgehen nach der Übersetzung von SQL:
  - Erhaltener Operatorbaum wird in **algebraischer Optimierungsphase** gemäß Regeln der relationalen Algebra optimiert.
  - In anschließender **nicht-algebraischer Optimierungsphase** werden spezifische Operator-Implementierungen ausgewählt und weitere Operatoren wie z.B. Sortierung, Index-Generierung etc. eingefügt.

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

# Algebraische Optimierung (1)

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

- Ziel: Transformation des Operatorbaums in äquivalenten Baum, der effizienter auszuführen ist.
- Anwendbare Gesetze der relationalen Algebra (Auszug):
  - Verbindung ist kommutativ und assoziativ.
  - Selektionen und Projektionen können zerlegt und zusammengefasst werden.
  - Projektion kommutiert mit Selektion, Verbindung und Vereinigung, solange benötigte Attribute nicht herausprojiziert werden.
  - Selektion kommutiert mit Verbindung, Vereinigung, Differenz und Gruppierung (solange die Selektionsbedingung sich nur auf die Gruppierungsattribute bezieht).
  - Kartesisches Produkt gefolgt von Selektion ergibt  $\theta$ -Verbindung.

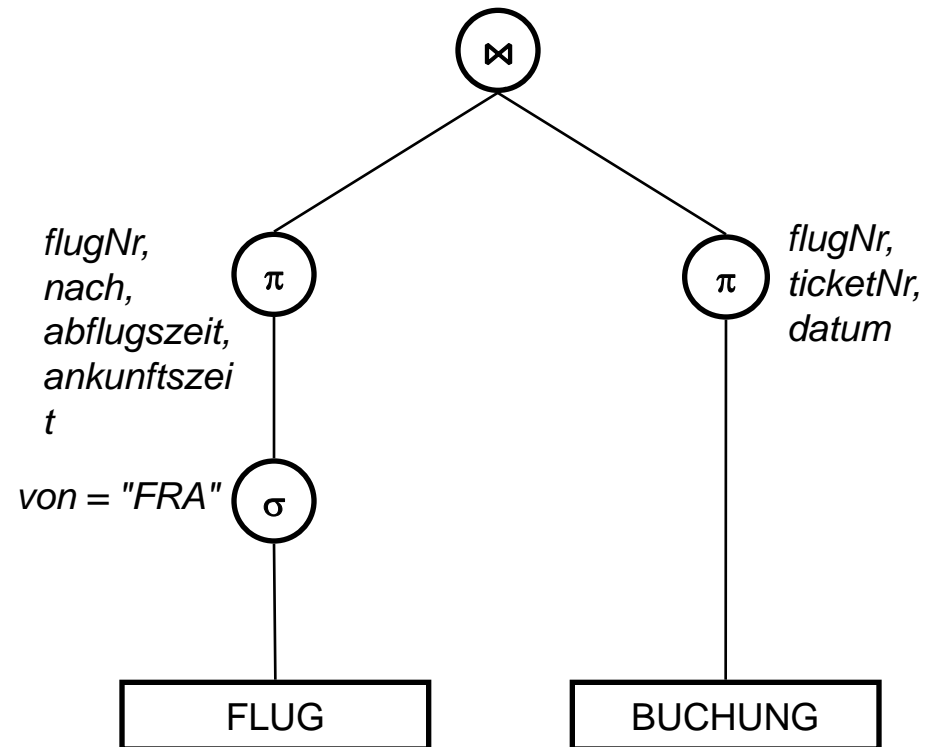
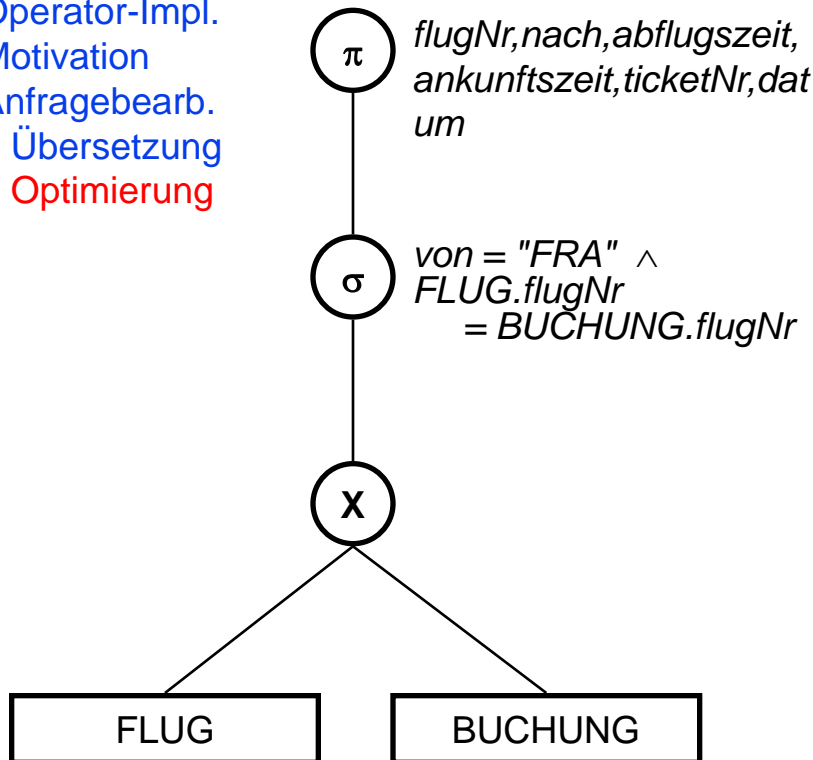
# Algebraische Optimierung (2)

- In der einfachsten Form kommen bei der algebraischen Optimierung eine Reihe von Heuristiken für sinnvolle Transformationen zum Einsatz:
  - Führe Selektionen so früh wie möglich durch.
  - Spalte Selektionen, die auf Attribute mehrerer Relationen Bezug nehmen, in eine Folge von Selektionen auf, die jeweils nur auf Attribute einer Relation Bezug nehmen. (Dann lässt sich nämlich die erste Regel besser anwenden.)
  - Fasse kartesische Produkte und Selektionen möglichst zu  $\theta$ -Verbindungen zusammen.
  - Führe Projektionen so früh wie möglich (aber nach eventuellen Selektionen) aus.
  - Füge ggf. zusätzliche Projektionen ein, um nicht mehr benötigte Attribute so früh wie möglich zu entfernen.

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

# Algebraische Optimierung: Beispiel

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung



# Nicht-algebraische Optimierung (1)

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

- Aufgabe: Auswahl geeigneter Operator-Implementierungen sowie weitere Optimierung des Operatorbaums.
- Somit kann sich der Operatorbaum gegenüber der logischen Optimierung verändern.
- Handlungsmöglichkeiten (Auswahl):
  - Einfügen von Sortierläufen, um Mischverbindung zu ermöglichen,
  - Erzeugen eines temporären Index, um Verbindung mit Direktzugriff zu ermöglichen,
  - Änderung der Reihenfolge von komplexen Selektionsbedingungen,
  - Änderung der Reihenfolge von Verbindungen
- Wesentliches Entscheidungskriterium: Größe der erzeugten Zwischenergebnisse.



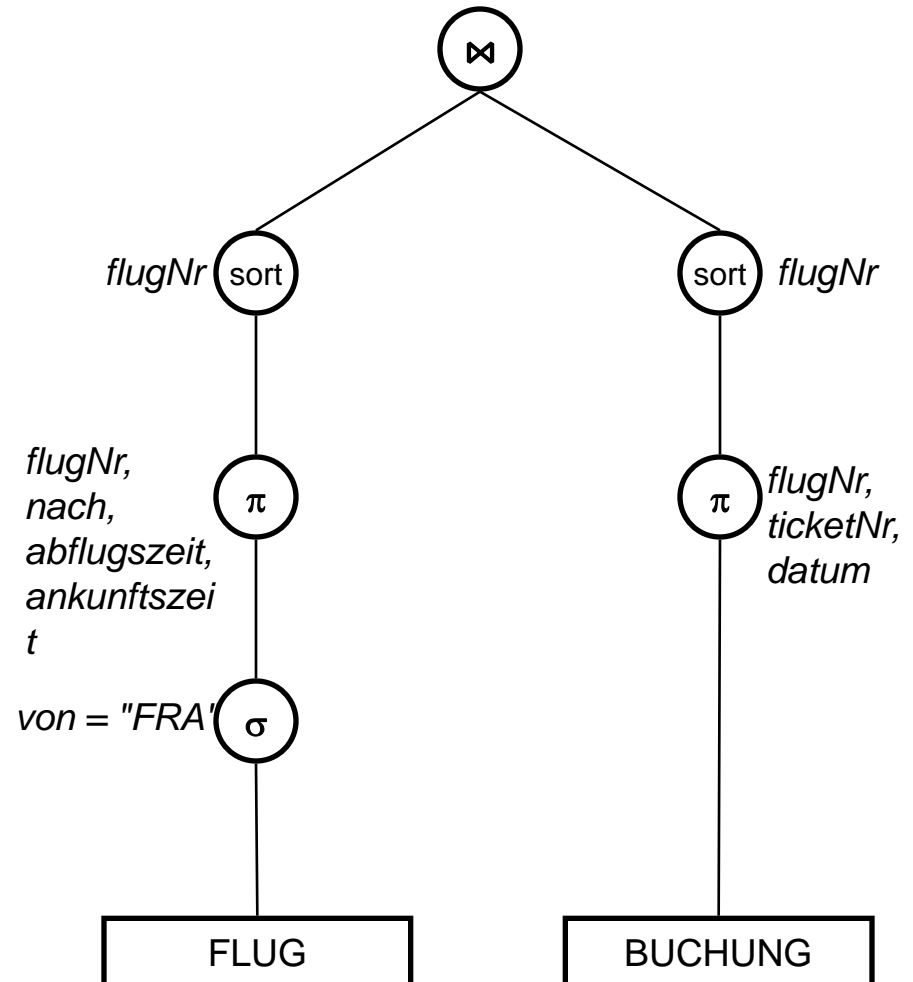
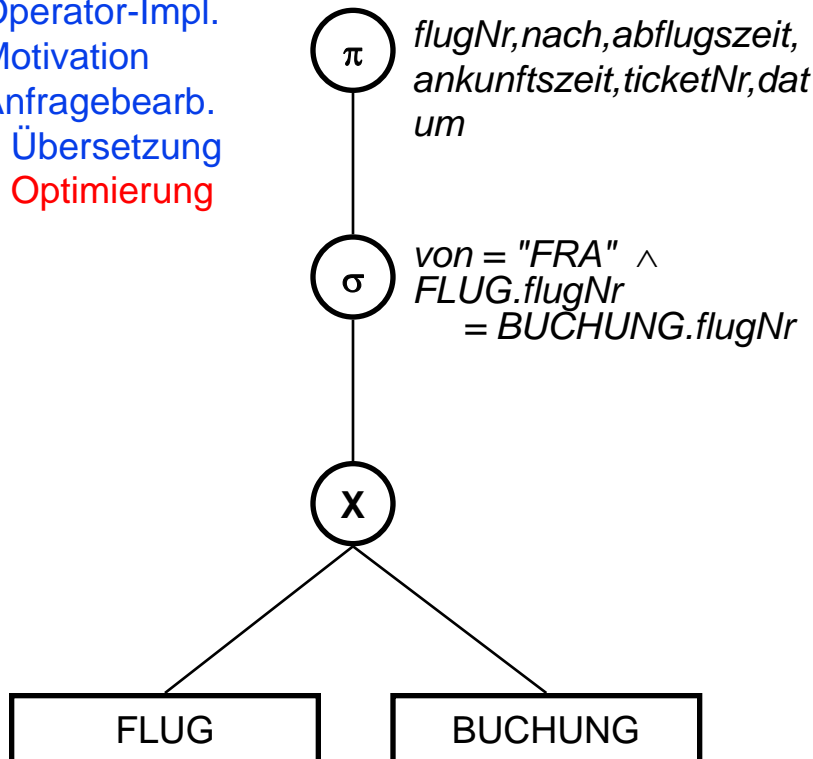
## Nicht-algebraische Optimierung (2)

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung

- Problem bei der Optimierung:  
Ausführungskosten eines bestimmten Operatorbaums müssen geschätzt werden.
- Benötigt werden u.a. Informationen über:
  - Größe der Ausgangstabellen,
  - Größe der verbleibenden Tabellen nach Selektion oder Projektion,
  - Treffer-Rate bei einer Verbindung,
  - Ausführungskosten von Operationen (hängen stark von Algorithmus ab)

# Nicht-algebraische Optimierung: Beispiel

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung





# Zusammenfassung

- Optimierung von Anfragen fundamental wichtig
  - Laufzeit kann verbessert werden
  - Größe von Zwischenergebnissen kann reduziert werden

Operator-Impl.  
Motivation  
Anfragebearb.  
Übersetzung  
Optimierung