

Optimierung

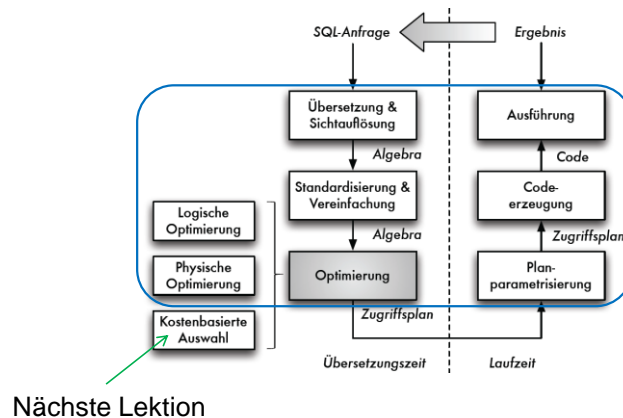
Lehrbuch Kapitel 8.4

1 L	Einführung
4 L	Datenorganisation Speicherung
4 L	Optimierung
2 L	Transaktionen, Recovery
2 L	Non-Standard Datenbanken
1 L	Repetition, Abschluss

← "You are here"

- Hash-Join-Varianten
- Grundlagen der Optimierung

- Grundüberlegungen: von der SQL-Anfrage zum Ausführungsplan

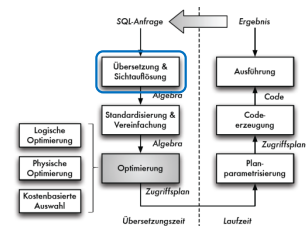
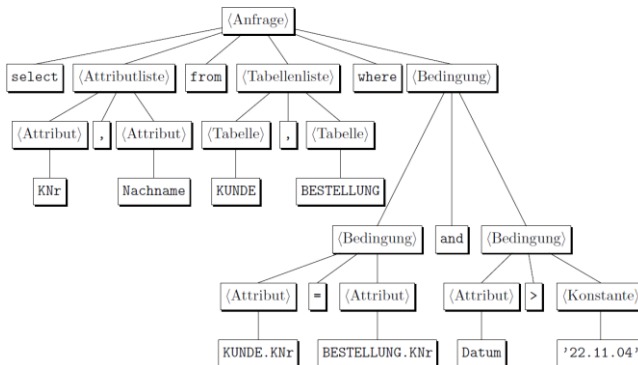


Der gesamte Ablauf vom SQL Statement bis zur Ausführung des Befehls:

1. Übersetzung und Sichtauflösung
 - SQL in Parse-Baum übersetzen (Syntax Analyse)
 - Semantische Analyse (Existieren Tabellen, Attribute, Rechte, ...)
 - Parse-Baum in relationale Bag-Algebra, respektive Operator-Baum übersetzen
 - Sichten (Views) auflösen (Sichtexpansion), Views durch SQL-Statements ersetzen
2. Standardisierung und Vereinfachung
 - Standardisierung (logische Ausdrücke und rel. Algebra)
 - Vereinfachung (Entfernung von redundanten Teilen, logische Ausdrücke, etc.)
 - Entschachtelung von Subqueries (spezieller Teil der Vereinfachung)
3. Optimierung
 - Logische Optimierung: Transformation mittels algebraischen, äquivalenzerhaltenden Regeln
 - Physische Optimierung: Entwicklung von potentiellen Ausführungsplanvarianten (Wahl der Basisalgorithmen), inkl. Berücksichtigung von parallelen Operationen
 - Kostenbasierte Bewertung der Ausführungsplanvarianten und Auswahl des Ausführungsplanes aufgrund statistischer Daten
4. Planparametrisierung: Vorcompilierte SQL-Anweisungen (Stored Procedure, Embedded SQL) müssen nicht mehr übersetzt und optimiert werden, für diese werden die gewählten Ausführungspläne gespeichert, welche als 'Methoden' mit Parametern aufgerufen werden können. Die Phase der Übersetzung kann in diesen Fällen entfallen.
Im SQL Server werden auch dynamische Abfragen in einem Pool zwischengespeichert, so dass wiederkehrende SQL-Anfragen nicht erneut übersetzt und optimiert werden müssen.
5. Codeerzeugung: Die Ausführungspläne können in ausführbaren Code umgewandelt werden. Werden die Ausführungspläne durch einen Interpreter durchgeführt, entfällt dieser Schritt.

Auf den nachfolgenden Folien dieser Lektion werden wir die Übersetzung des SQL Statement bis und mit Code-Erzeugung betrachten (blauer Rahmen in der Folie, ohne kostenbasierte Auswahl).

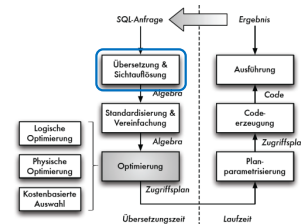
```
SELECT k.KNr, k.Nachname
FROM KUNDE k, BESTELLUNG b
WHERE k.KNr = b.KNr AND b.Datum > '22-11-04'
```



Die Zeichenfolge einer gegebenen Anfrage muss in eine Datenstruktur mit SQL-Syntax-Konstrukten, den Parse-Baum, überführen werden. Dies ist Aufgabe des Parsers.

1. Übersetzung und Sichtauflösung

1. Parsen der Anfrage (Parse-Baum)
2. Semantische Analyse des Eingabetextes
 - Lexikalische Korrektheit:
korrekte Angabe der Symbole (Schlüsselwörter etc.)
 - Syntaktische Korrektheit:
korrekte Reihenfolge, Einhaltung der Syntaxregeln
 - Semantische Korrektheit:
Existieren Relationen und Attribute, Typprüfung,
korrekte Projektionen
 - Zugriffsberechtigungen



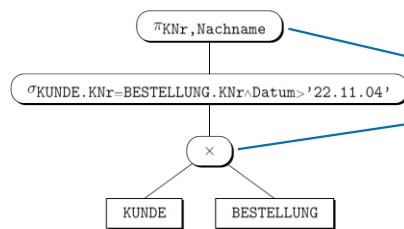
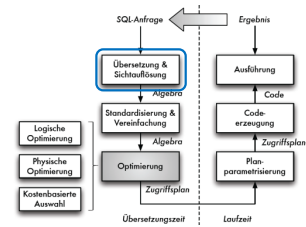
6

Der Parser überprüft auch die lexikalische und syntaktische Korrektheit des SQL-Ausdrucks:

- Lexikalische Korrektheit:
korrekte Angabe der Symbole (gültige Schlüsselwörter, Variablennamen, Zeichen etc.)
(Bsp. lexikalischer Fehler, ungültiger Variablenname: `DECLARE 2A int;`)
- Syntaktische Korrektheit (Syntaxprüfung):
korrekte Reihenfolge, Einhaltung der Syntaxregeln (korrekte 'Grammatik')
(Bsp. Syntaktischer Fehler, falsche Reihenfolge: `FROM Kunden SELECT Name;`)
- Semantische Korrektheit:
`SELECT KundenNr FROM Ortschaften` (das Attribut existiert nicht in der Tabelle).
- Zugriffsberechtigungen:
Wir werden in der Vorlesung kaum über Zugriffsberechtigungen sprechen. Diese sind in der Praxis aber natürlich trotzdem extrem wichtig...

1. Übersetzung und Sichtauflösung

1. Parsen der Anfrage (Parse-Baum)
2. Semantische Analyse des Eingabetextes
3. Übersetzung in rel. Bag-Algebra (**Operator-Baum**)
 - Operatoren als Knoten
 - Kanten repräsentieren Datenfluss (von unten nach oben)



Operatoren

```
SELECT k.KNr, k.Nachname
FROM KUNDE k, BESTELLUNG b
WHERE k.KNr = b.KNr AND b.Datum > '22-11-04'
```

$\pi_{KNr, Nachname}(\sigma_{KUNDE.KNr = BESTELLUNG.KNr \wedge Datum > '22.11.04'}(KUNDE \times BESTELLUNG))$

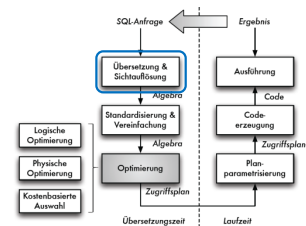
7

Wie kann ein Parse-Baum in einen Operator-Baum übersetzt werden? Anbei ein paar einfache Regeln (unvollständig), wie das aussehen könnte:

- **Transformationsregeln**
 - Relationen der Tabellenliste hinter FROM untereinander durch Kreuzprodukt verknüpfen
 - Bedingung im WHERE-Teil als Selektion übernehmen
 - Spaltenliste hinter SELECT als abschliessende Projektion
- Zusätzlich
 - Berücksichtigung von SQL-Konstrukten wie ORDER BY, GROUP BY, ...
 - Später während Vereinfachung: Auflösen von Unteranfragen (Subqueries)

1. Übersetzung und Sichtauflösung

1. Parsen der Anfrage (Parse-Baum)
2. Semantische Analyse des Eingabetextes
3. Übersetzung in rel. Bag-Algebra (Operator-Baum)
4. Auflösung von Sichten (Sichtexpansion)
 - Einsetzen der Sichtdefinition in Anfrage
 - im Parse-Baum, oder erst
 - im Operator-Baum
 - Rekursiver Prozess: Sichten über Sichten möglich

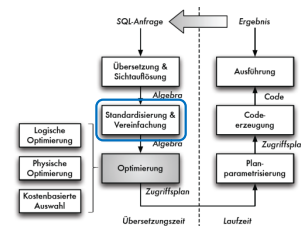


Sichten über Sichten sollten generell vorsichtig eingesetzt werden, da hier die Komplexität sehr rasch wächst und die Wartung dadurch schwer fällt.

2. Standardisierung & Vereinfachung

Ziel: Vereinfachung der folgenden Optimierungsschritte durch ein einheitliches Anfrageformat.

- **Standardisierung** von Selektions- und Verbundbedingungen.
- **Vereinfachen** und entfernen redundanter Ausdrücke.
- **Entschachtelung**: Auflösung von Subqueries (ohne geschachtelte Iteration). Die Entschachtelung ist Teil der Vereinfachung.



9

1. Standardisierung: Logische Bedingungen werden in Normalform (konjunktive oder disjunktive) gebracht.
2. Vereinfachen: Durch Ausnutzung mathematischer Gesetzmässigkeiten werden die Ausdrücke vereinfacht (z.B. ist ' $B \vee \neg B$ ' immer true).
3. Entschachtelung: Bei der Entschachtelung werden Subqueries / Unterabfragen im WHERE-Teil, z.B.:

- ... WHERE FK IN (SELECT PK FROM A)

aufgelöst, sodass diese als 'normale' Join-Operationen ausgeführt werden können.

Diese Vereinfachung durch Entschachtelung hat mehrere Gründe:

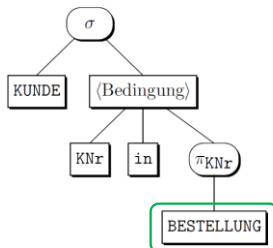
- i. Anstelle einer geschachtelten Iteration kann eine effizientere Verbundoperation (Basisalgorithmus, z.B. Hash-Join) ausgeführt werden.
- ii. Gemeinsame Teilabfragen lassen sich besser erkennen (müssen nur ein Mal durchgeführt werden).

Der SQL Server entschachtelt NICHT alle Subqueries. Zur Behandlung von Subqueries hat er mehrere verschiedene Strategien, die Entschachtelung (Flattening, Dt.=flach machen) ist eine davon (z.B. wenn die Zwischentabellen der Subqueries zu gross für den Hauptspeicher werden). Im Ausführungsplan kann betrachtet werden, für welche konkrete Strategie sich der Optimizer entschieden hat.

Entschachtelung von Anfragen

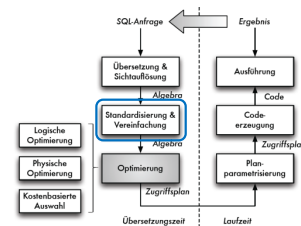
- Entschachtelung Beispiel: Wie kann das SQL-Statement ohne **Subquery** formuliert werden?

```
SELECT * FROM KUNDE
WHERE KNr IN (SELECT KNr FROM BESTELLUNG)
```



Vereinfacht:

```
SELECT * FROM KUNDE
JOIN BESTELLUNG ON BESTELLUNG.KNr = KUNDE.KNr
```



10

Sind die beiden SQL-Statements wirklich identisch? Nein, denn:

- Im zweiten Statement werden die Attribute beider Tabellen zurückgegeben. Das ist ein kleines Detail, das einfach zu lösen ist (Attributliste angeben).
- Falls für einen Kunden mehrere Bestellungen vorliegen, werden in der zweiten Abfrage auch mehrere Tupel pro Kunde im Resultat erscheinen. Es müsste also noch eine Group- oder Distinct-Operation ausgeführt werden, um die Duplikate zu eliminieren.
Im SQL Server gibt es hierfür einen speziellen Basisalgorithmus, den «Existence-Join», welcher den Join nur mit dem ersten Tupel ausführt, dadurch kann die Entschachtelung effizient ausgeführt werden.

Wir unterscheiden vier Typen von Schachtelungen (gemäss Lehrbuch):

- Typ-A-Schachtelung (A = Aggregation)
- Typ-N-Schachtelung (N = Nested)
- Typ-J-Schachtelung (J = Join)
- Typ-JA-Schachtelung (J = Join mit A = Aggregation)

Diesen 4 Typen werden auf den folgenden Folien behandelt.

In der ursprünglichen Publikation (Kim, W.: On Optimizing an SQL-like Nested Query, ACM Transactions on Database Systems, Band 7, Nr. 3, S. 443-469, September 1982) wird noch die Typ-D-Schachtelung (Division) erwähnt. Diese ist sehr selten, wir verzichten daher auf deren Betrachtung.

1. Typ-A-Schachtelung (Aggregation):

1. Innerer Block Q enthält kein Verbundprädikat, das die äussere Relation referenziert
2. Q berechnet Aggregat

• Beispiel:

```
SELECT BestNr
FROM BESTELLUNG
WHERE ProdNr = (SELECT MAX(ProdNr)
                FROM PRODUKT
                WHERE Preis < 15)
```

Q

• Ausführung:

1. Innere Anfrage + Aggregat berechnen, dann
2. Ergebnis in äussere Anfrage einsetzen und diese berechnen

Hier findet keine 'echte' Entschachtelung statt. Das Resultat der inneren Abfrage, welches völlig unabhängig vom umgebenden Select gebildet werden kann, wird einfach als Konstante in die äussere Anfrage eingesetzt.

2. Typ-N-Schachtelung (Nested):

1. Innerer Block Q enthält kein Verbundprädikat, das die äussere Relation referenziert
2. Q berechnet **kein** Aggregat

- Beispiel:

```
SELECT BestNr
FROM BESTELLUNG
WHERE ProdNr IN (SELECT ProdNr
                  FROM PRODUKT
                  WHERE Preis < 15)
```

Q

- Ausführung:

- Variante A: Innere Anfrage berechnen (Tabelle) und in Äussere einsetzen
- Variante B: Entschachtelung

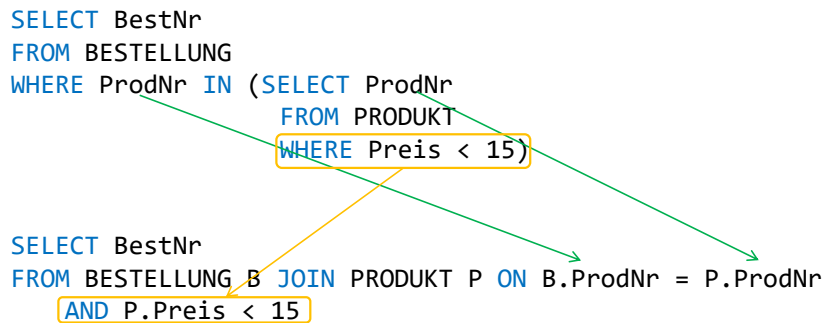
Bei der Variante A kann das Resultat der inneren Abfrage so gross werden, dass das Zwischenresultat auf Externspeichern ausgelagert werden muss. Dann ist die Variante B mit Entschachtelung eine Alternative (bei welcher einer der Verbund-Basisalgorithmen angewendet werden kann).

2. Typ-N-Schachtelung (Nested):

```

SELECT BestNr
FROM BESTELLUNG
WHERE ProdNr IN (SELECT ProdNr
                  FROM PRODUKT
                  WHERE Preis < 15)

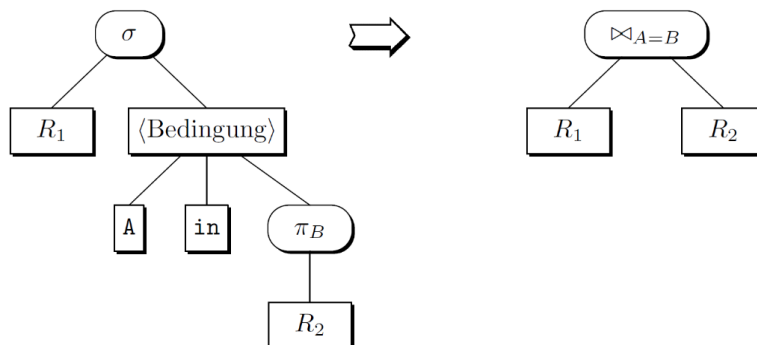
SELECT BestNr
FROM BESTELLUNG B JOIN PRODUKT P ON B.ProdNr = P.ProdNr
AND P.Preis < 15
    
```



Bem: hier wird im SQL-Server kein vollständiger Join sondern nur ein «Existence-Join» ausgeführt.

13

Typ-N-Anfragen mit IN-Prädikaten der Tiefe n-1 (Anzahl Verbundprädikate) können in semantisch äquivalente n-Relationenanfragen (Join mit n Relationen) transformiert werden:



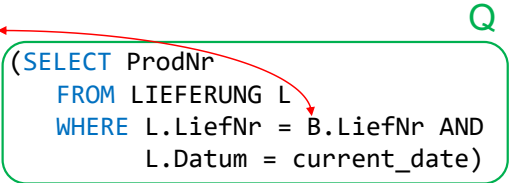
Die Grafik zeigt schematisch, wie die IN-Bedingung (ohne Abhängigkeit zum äusseren Block) in Kombination mit einer Projektion in einen Join umgewandelt werden kann.

3. Typ-J-Schachtelung (Join):

1. Innerer Block Q referenziert im Verbundprädikat die Relation des äusseren Blocks ('korrelierte Unterabfrage')
2. Q liefert keinen Aggregatwert

- Beispiel:

```
SELECT BestNr
FROM BESTELLUNG B
WHERE B.ProdNr IN (SELECT ProdNr
                   FROM LIEFERUNG L
                   WHERE L.LiefNr = B.LiefNr AND
                        L.Datum = current_date)
```



14

Bei einer 'naiven' Ausführung des Befehls, würde für jeden Datensatz des äusseren Blocks der innere Block ein Mal ausgeführt werden, was sehr aufwändig wäre. Durch die Entschachtelung kann ein effizienter Join-Basisalgorithmus angewendet werden.

3. Typ-J-Schachtelung (Join):

```

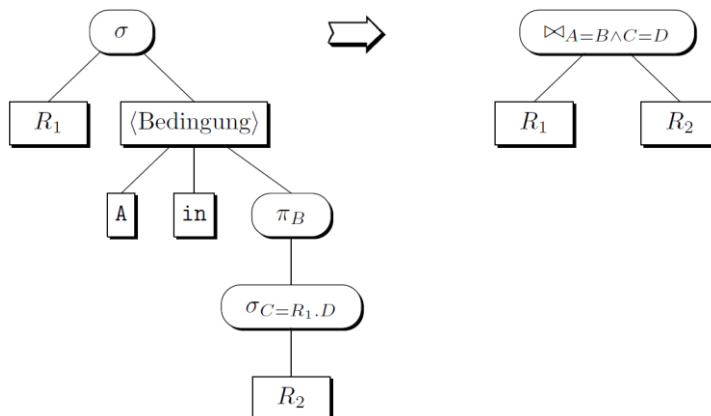
SELECT BestNr
FROM BESTELLUNG B
WHERE B.ProdNr IN (SELECT ProdNr
FROM LIEFERUNG L
WHERE L.LiefNr = B.LiefNr AND
L.Datum = current_date)

SELECT BestNr
FROM BESTELLUNG B JOIN LIEFERUNG L
ON B.ProdNr = L.ProdNr
AND L.LiefNr = B.LiefNr AND L.Datum = current_date
    
```

Diagram illustrating the transformation of a nested query (Typ-J-Schachtelung) into a join operation. The top query uses an IN condition to filter products from the BESTELLUNG table based on a subquery that selects products from the LIEFERUNG table where the delivery number and date match the order's delivery number and current date. The bottom query shows the equivalent join operation, joining BESTELLUNG (B) and LIEFERUNG (L) on the condition B.ProdNr = L.ProdNr AND L.LiefNr = B.LiefNr AND L.Datum = current_date. Arrows indicate the mapping of variables and conditions between the two queries.

15

Die Grafik zeigt schematisch, wie die IN-Bedingung in Kombination mit einer Projektion und einer Selektion mit Abhängigkeit zum äusseren Block, in einen Join umgewandelt werden kann:

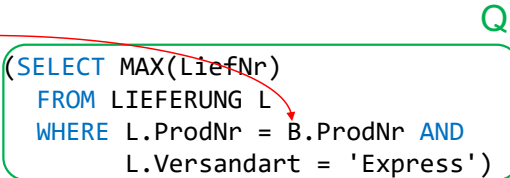


4. Typ-JA-Schachtelung (Join – Aggregation):

1. Innerer Block Q referenziert im Verbundprädikat die Relation des äusseren Blocks ('korrelierte Unterabfrage')
2. Q liefert Aggregatwert

• Beispiel:

```
SELECT BestNr
FROM BESTELLUNG B
WHERE B.BestNr IN (SELECT MAX(LiefNr)
                   FROM LIEFERUNG L
                   WHERE L.ProdNr = B.ProdNr AND
                        L.Versandart = 'Express')
```



• Ablauf:

1. In Typ-J-Schachtelung umwandeln, indem vorab die innere Relation berechnet wird, indem über L.ProdNr gruppiert wird
2. Typ-J-Schachtelung entschachteln

16

Die JA-Entschachtelung erfolgt in zwei Schritten (siehe Folie). Der Block Q kann zunächst wie folgt als Tabelle berechnet werden:

```
SELECT MAX(LiefNr) AS MAX_LiefNr, ProdNr FROM LIEFERUNG
WHERE Versandart = 'Express' GROUP BY ProdNr
```

Da der Join anschliessend zusätzlich über das Verbundattribut ProdNr erfolgt, muss über dieses gruppiert werden und dieses in der Zwischentabelle als Attribut enthalten sein.

Diese Zwischentabelle kann nun eingesetzt werden und mittels Typ J-Entschachtelung in eine Join-Operation umgewandelt werden (siehe nächste Folie).

4. Typ-JA-Schachtelung (Join – Aggregation):

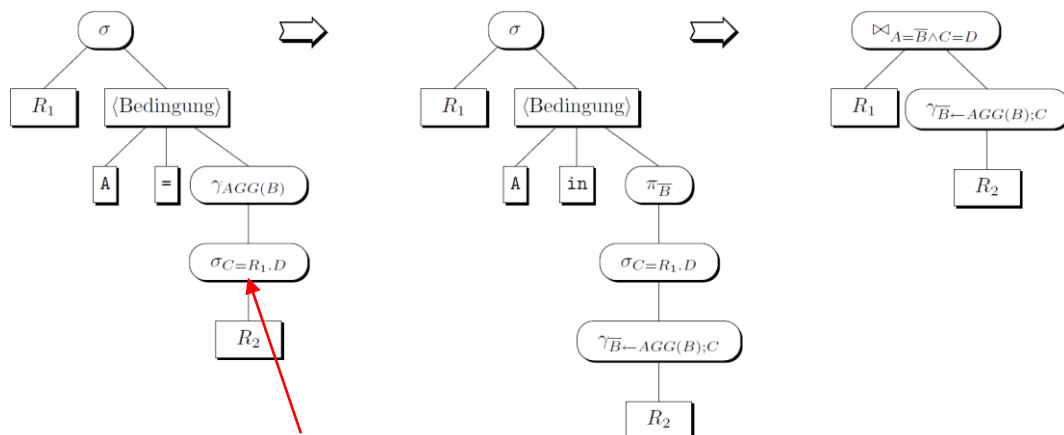
```

SELECT BestNr
FROM BESTELLUNG B
WHERE B.BestNr IN (SELECT MAX(LiefNr)
                   FROM LIEFERUNG L
                   WHERE B.ProdNr = L.ProdNr AND
                      L.Versandart = 'Express')

SELECT B.BestNr
FROM BESTELLUNG B JOIN (SELECT MAX(LiefNr) AS MAX_LiefNr, ProdNr
                       FROM LIEFERUNG
                       WHERE Versandart = 'Express'
                       GROUP BY ProdNr) L
ON B.BestNr = L.MAX_LiefNr AND B.ProdNr = L.ProdNr
    
```

17

Anstelle zweier verschachtelter SELECT-Abfragen haben wir nun zwei 'unabhängige' SELECT-Abfragen (eine davon mit Aggregation, eine Typ A-Schachtelung), welche mittels Join-Operation miteinander verknüpft werden. Die JA-Schachtelung ist damit aufgehoben. Bildlich kann der Vorgang wie folgt beschrieben werden:



Einschränkungen der Typ-JA-Entschachtelung:

- Der Operator im Verbundprädikat θ muss '=' sein
- Die Aggregatfunktion darf nicht COUNT(B) sein
- R1.D muss eindeutig (unique) sein; ausser bei MIN() oder MAX()

Es gibt Massnahmen, die die Einschränkungen beheben, werden hier aber nicht betrachtet, sind in der Literatur aber bekannt:

- Richard A. Ganski, Harry K. T. Wong: Optimization of Nested SQL Queries Revisited, [SIGMOD Conference 1987](#): 23-33
- H. Buff, Datenbanktheorie, 2003

Nach der Entschachtelung liegt ein vereinfachter, standardisierter Operator-Baum vor. Dieser bildet die

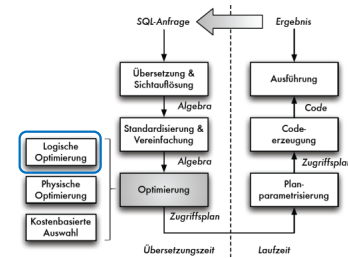
Basis für die nachfolgende Optimierung.

3. Logische Optimierung

Der Ausführungsplan wird unabhängig von konkreten Speicherungsformen mittels **algebraischen, äquivalenzerhaltenden Regeln** umgeformt.

Diese **logische, algebraische Optimierung** zielt darauf ab, kleinere Zwischenergebnisse zu erhalten. Wir zeigen dies exemplarisch:

1. Zunächst werden 12 algebraische Regeln eingeführt.
2. Anschliessend werden diese 12 algebraischen Regeln in 6 Transformationsschritten eingesetzt, um einen einfachen Optimierungsalgorithmus zu «implementieren».



18

Bemerkung am Rande: Die Begriffe Ausführungsplan, Anfrageplan, Zugriffsplan, Plan, Execution Plan werden als Synonyme verwendet. Und, je nach Phase der Übersetzung sind verschiedene Darstellungen üblich (auch der Operator-Baum ist ein Ausführungsplan).

Bei der logischen, algebraischen Optimierung wird noch keine Rücksicht auf die verfügbaren Basisalgorithmen (z.B. Hash-Join oder Sort-Merge-Join) oder Hardware genommen. Es werden z.B. Selektionsoperationen vor/zu anderen Operationen verschoben. Grundsätzlich wird dabei versucht, einen Ausführungsplan mit kleineren Zwischenergebnisse zu erhalten.

Die algebraische Optimierung ist eine heuristische Methode (Näherungsverfahren), d.h., es sind keine eindeutigen Lösungsansätze hierfür bekannt. In der logischen Optimierung wird zudem auch versucht, redundante Ausführungsteile zu erkennen und zu eliminieren.

Nebst der algebraischen Optimierung gibt es auch Verfahren (der logischen Optimierung), welche als Ziel die Minimierung der Anzahl Joins haben (z.B. Tableau-Technik, siehe Lehrbuch).

3. Logische Optimierung – algebr. Regeln

Folgende 12 **algebraische Regeln** sind äquivalenzerhaltend:

1. Aufbrechen von Konjunktionen im Selektionsprädikat:

$$\sigma_{c_1 \wedge c_2 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\dots(\sigma_{c_n}(R)) \dots))$$

2. σ ist kommutativ:

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

3. π -Kaskaden: Falls $A_1 \subseteq A_2 \subseteq \dots \subseteq A_n$, dann gilt:

$$\pi_{A_1}(\pi_{A_2}(\dots(\pi_{A_n}(R)) \dots)) \equiv \pi_{A_1}(R)$$

19

Die folgenden sechs Folien enthalten 12 algebraischen Regeln (diese Aufzählung ist nicht vollständig). Mit diesen 12 Regeln werden wir dann einen einfachen Optimierungsalgorithmus erstellen. Dieser Algorithmus selbst besteht aus sechs Transformationsschritten, welche den SQL-Ausdruck optimieren.

Diese Regeln müssen natürlich auch für Bag-Algebra gelten, ansonsten würden Fehler auftreten.

3. Logische Optimierung – algebr. Regeln

4. Vertauschen von σ und π : Falls die Selektion sich nur auf die Attribute A_1, \dots, A_n der Projektionsliste bezieht, können die beiden Operationen vertauscht werden:

$$\pi_{A_1, \dots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, \dots, A_n}(R))$$

5. \times , \bowtie_j , \cup und \cap sind kommutativ. Wenn also Φ eine dieser Operationen bezeichnet, so gilt:

$$R \Phi S \equiv S \Phi R$$

Die Operationen Vereinigung \cup und Differenz \cap (wie auch die Differenz) müssen natürlich vereinigungsverträglich sein.

6. Vertauschen von σ mit \bowtie_j (statt Join natürlich auch mit Cross-Join):

Falls das Selektionsprädikat c nur auf Attribute der Relation R zugreift, kann man die beiden Operationen vertauschen:

$$\sigma_c(R \bowtie_j S) \equiv \sigma_c(R) \bowtie_j S$$

Falls c_1 sich nur auf Attribute aus R und c_2 sich nur auf Attribute aus S bezieht, gilt folgende Äquivalenz:

$$\sigma_{c_1 \wedge c_2}(R \bowtie_j S) \equiv \sigma_{c_1}(R) \bowtie_j (\sigma_{c_2}(S))$$

7. Vertauschung von π mit \bowtie_j

Die Projektionsliste L sei: $L = \{A_1, \dots, A_n, B_1, \dots, B_m\}$, wobei A_i Attribute aus R und B_i Attribute aus S seien. Falls sich das Joinprädikat j nur auf Attribute aus L bezieht, gilt folgende Umformung:

$$\pi_L (R \bowtie_j S) \equiv (\pi_{A_1, \dots, A_n} (R)) \bowtie_j (\pi_{B_1, \dots, B_m} (S))$$

Falls das Joinprädikat sich auf weitere Attribute, sagen wir A_1', \dots, A_p' aus R und B_1', \dots, B_q' aus S bezieht, müssen diese für die Join-Operation erhalten bleiben und können erst danach herausprojiziert werden:

$$\pi_L (R \bowtie_j S) \equiv \pi_L (\pi_{A_1, \dots, A_n, A_1', \dots, A_p'} (R) \bowtie_j \pi_{B_1, \dots, B_m, B_1', \dots, B_q'} (S))$$

8. Die Operationen \times , \bowtie , \cup , \cap sind jeweils (einzeln betrachtet) assoziativ. Wenn also Φ eine dieser Operationen bezeichnet, so gilt:

$$(R \Phi S) \Phi T \equiv R \Phi (S \Phi T)$$

9. Die Operation σ ist distributiv mit \cup , \cap , $-$. Falls Φ eine dieser Operationen bezeichnet, gilt:

$$\sigma_c(R \Phi S) \equiv (\sigma_c(R)) \Phi (\sigma_c(S))$$

10. Die Operation π ist distributiv mit \cup :

$$\pi_A(R \cup S) \equiv (\pi_A(R)) \cup (\pi_A(S))$$

11. Die Join- und/oder Selektionsprädikate können mittels de Morgan's Regeln umgeformt werden:

$$\neg (c_1 \wedge c_2) \equiv (\neg c_1) \vee (\neg c_2) \text{ bzw. } \neg (c_1 \vee c_2) \equiv (\neg c_1) \wedge (\neg c_2)$$

12. Ein kartesisches Produkt, das von einer Selektions-Operation gefolgt wird, deren Selektionsprädikat Attribute aus beiden Operanden des kartesischen Produktes enthält, kann in eine Join-Operation umgeformt werden.

Sei c eine Bedingung der Form $A \theta B$, mit A ein Attribut von R und B ein Attribut aus S : $\sigma_c(R \times S) \equiv R \bowtie_c S$

3. Logische Optimierung – Transformation

Anwendung von algebraischen Regeln in **Transformationsschritten** (durch das RDBMS):

- a. Mittels Regel 1 werden konjunktive Selektionsprädikate in Kaskaden von σ -Operationen zerlegt.
- b. Mittels Regeln 2, 4, 6, und 9 werden Selektionsoperationen soweit „nach unten“ propagiert wie möglich.
- c. Forme eine \bowtie -Operation, die von einer σ -Operation gefolgt wird, wenn möglich in eine \bowtie -Operation um (Regel 12).
- d. Mittels Regel 5 und 8 werden die Blattknoten so vertauscht, dass derjenige, der das kleinste Zwischenergebnis liefert, zuerst ausgewertet wird.
- e. Mittels Regeln 3, 4, 7, und 10 werden Projektionen soweit wie möglich nach unten propagiert.
- f. Versuche Operationsfolgen zusammenzufassen, wenn sie in einem „Durchlauf“ ausführbar sind (z.B. Anwendung von Regel 1, Regel 3, aber auch Zusammenfassung aufeinanderfolgender Selektionen und Projektionen zu einer „Filter“-Operation).

25

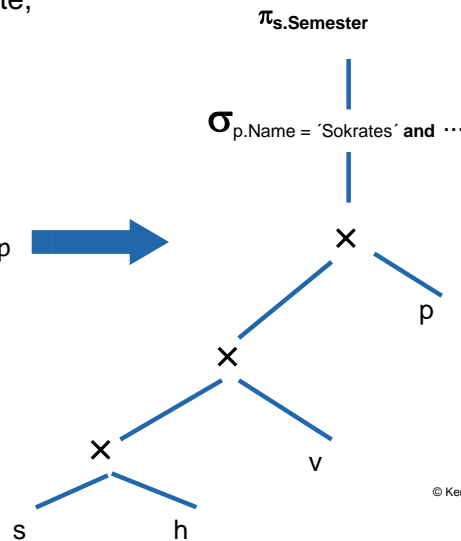
Aufgrund der eingeführten 12 algebraischen Regeln kann nun ein einfacher Optimierungsalgorithmus erstellt werden. Die oben aufgeführten Transformationsschritte zeigen einen solchen möglichen Algorithmus. Ein echtes DBMS wird das ähnlich, aber natürlich ganz anders, lösen.

3. Logische Optimierung – Transformation

- Anwendung Transformationsschritte,
Ausgangspunkt: kanonische Form

Beispiel:

```
SELECT DISTINCT s.Semester  
FROM Studenten s, Hören h,  
Vorlesungen v, Professoren p  
WHERE  
  p.Name = 'Sokrates' AND  
  v.gelesenVon = p.PersNr AND  
  v.VorINr = h.VorINr AND  
  h.MatrNr = s.MatrNr
```



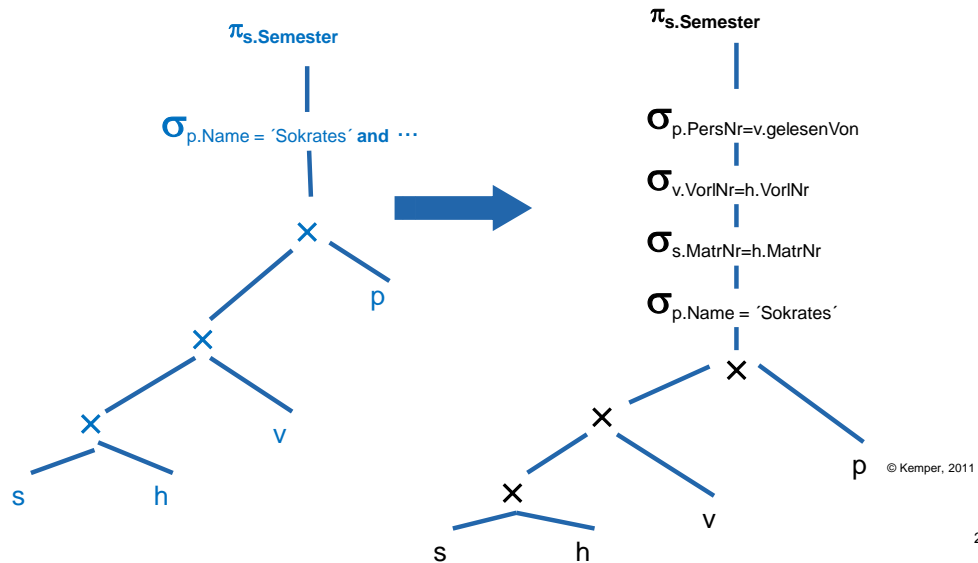
© Kemper, 2011

26

Auf den folgenden Folien wird gezeigt, wie durch Anwendung der Transformationsschritte (und der passenden algebraischen Regeln) der obige SQL-Ausdruck optimiert wird.

3. Logische Optimierung – Transformation

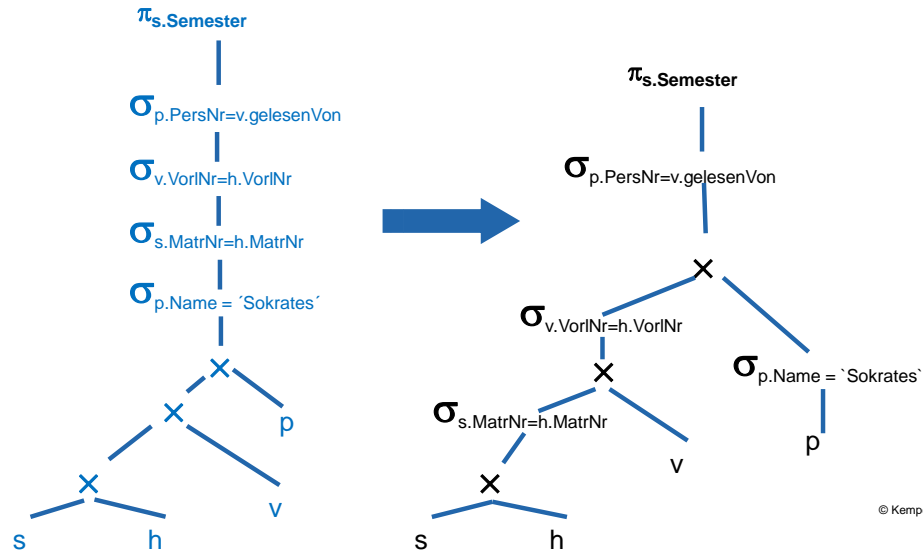
- Transformationsschritt a: Selektionsprädikate aufspalten



Mittels Regel 1 werden konjunktive Selektionsprädikate in Kaskaden von σ -Operationen zerlegt.

3. Logische Optimierung – Transformation

- Transformationsschritt b: Selektionen nach «unten» verschieben

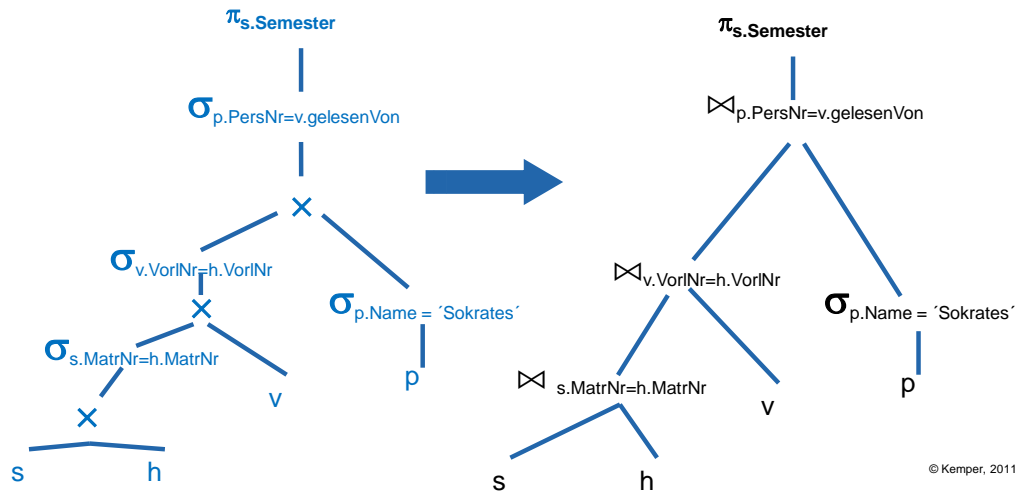


28

Mittels Regeln 2, 4, 6, und 9 werden Selektionsoperationen soweit „nach unten“ propagiert wie möglich. Hier wird Regel 6 angewandt.

3. Logische Optimierung – Transformation

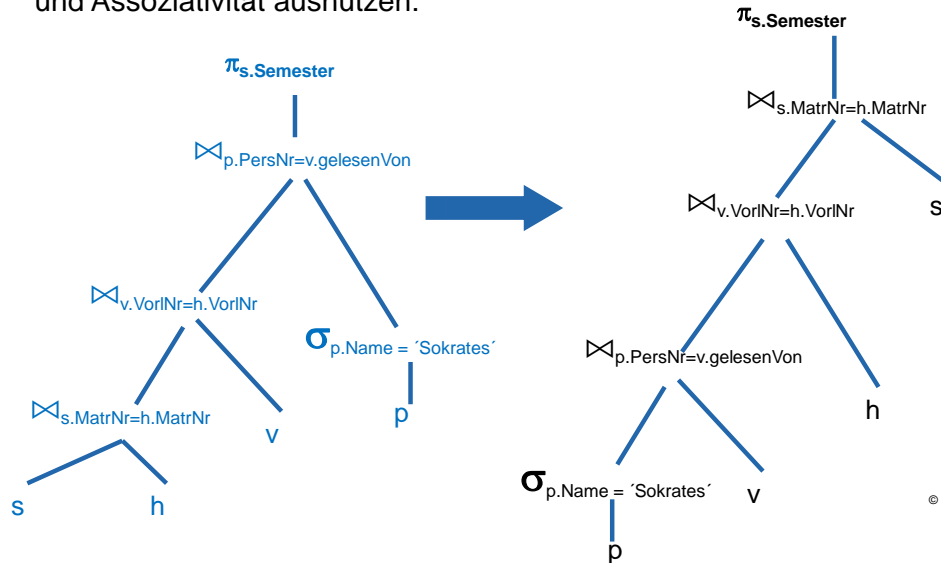
- Transformationsschritt c: Zusammenlegen von Selektionen & Kreuzprodukten zu Joins



Regel 12: Forme eine \bowtie -Operation, die von einer σ -Operation gefolgt wird, wenn möglich in eine σ -Operation um.

3. Logische Optimierung – Transformation

- Transformationsschritt d: Optimierung der Joinreihenfolge; Kommutativität und Assoziativität ausnutzen:



© Kemper, 2011

30

Mittels Regel 8 werden die Blattknoten so vertauscht, dass derjenige, der das kleinste Zwischenergebnis liefert, zuerst ausgewertet wird.

Dem Beispiel liegen natürlich Annahmen über die Anzahl der Records und die Anzahl verschiedener Werte über die Verbundattribute zugrunde. Das DBMS führt hierzu intern, wie bereits erwähnt, eigenen Statistiken.

Die Transformationsschritte e und f können im Beispiel nicht angewendet werden, diese entfallen.

Die logische Optimierung des SQL-Ausdruckes anhand unsere einfachen Algorithmus ist damit abgeschlossen.

3. Logische Optimierung – Transformation

- Logische Optimierung in SQL-Syntax dargestellt:

```
SELECT DISTINCT s.Semester
FROM Studenten s, Hören h,
      Vorlesungen v, Professoren p
WHERE
  p.Name = 'Sokrates' AND
  v.gelesenVon = p.PersNr AND
  v.VorlNr = h.VorlNr AND
  h.MatrNr = s.MatrNr
```

```
SELECT DISTINCT s.Semester FROM Professoren p
JOIN Vorlesung s ON p.PersNr = v.gelesenVon
JOIN Hören h ON v.VorlNr = h.VorlNr
JOIN Studenten s ON h.MatrNr = s.MatrNr
WHERE p.Name = 'Sokrates'
```

© Kemper, 2011

31

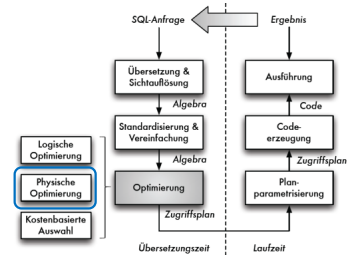
Hier wird die Umwandlung unseres einfachen Beispiels anhand der korrespondierenden SQL-Ausdrücke dargestellt.

4. Physische Optimierung

Physische oder Interne Optimierung:

Optimalen, physischen Plan mit Basisalgorithmen erstellen.

Umformung ist nicht eindeutig ->
verschiedenen Pläne generieren und
'anschliessend' Kostenabschätzung



32

Das Ergebnis der logischen Optimierung muss in einen ausführbaren und optimalen, physischen Plan umgeformt werden. Die algebraischen Operatoren müssen dabei durch Basisalgorithmen des DBMS (z.B. Hash-Join) ersetzt werden.

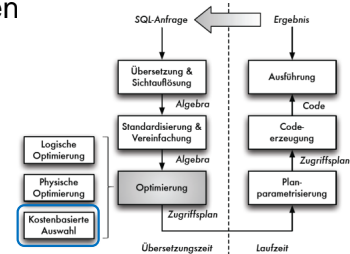
Diese Umformung ist aber nicht eindeutig, es müssen daher verschiedenen Pläne generiert werden, so dass die 'anschliessende' Kostenabschätzung den 'besten' Plan auswählen kann. Berücksichtigt werden müssen z.B.:

- Konkrete Speicherungstechniken (Indexe, Cluster)
- Algorithmen (Hash-Join, Sort-Merge-Join, ...)
- RAM, Speicher, SSD, ...

5. Kostenbasierte Auswahl

Statistikinformationen (Grösse von Tabellen, Selektivität von Attributen, etc.) für die Auswahl eines konkreten internen Planes nutzen.

→ Wird in der nächsten Vorlesung genauer betrachtet.



Achtung: Logische, physische und kostenbasierte Auswahl werden **nicht sequentiell** ausgeführt, sondern bieten verschiedene Möglichkeiten, aus denen der Optimierer eines RDBMS auswählen kann (wie diese Auswahl erfolgt ist ein gut gehütetes Geheimnis der RDBMS-Hersteller)

Phasen der Anfrageverarbeitung:

6. Planparametrisierung

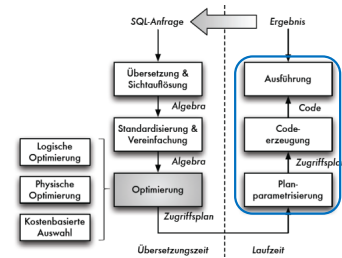
Bei vorkompilierten Anfragen (etwa Embedded-SQL, Stored Procedure): Ersetzen der Platzhalter durch Werte

7. Code-Erzeugung

Umwandlung des Zugriffsplans in ausführbaren Code,
oder Ausführung mittels Interpreter

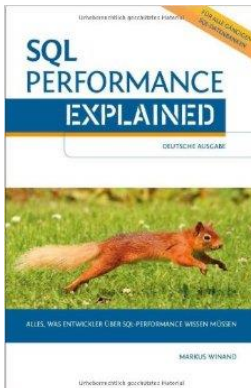
8. Ausführung

Ausführung des Codes auf dem DB-Server



Wir werden auf die Anfrageverarbeitung in DAB2 nicht weiter eingehen.

- Das nächste Mal: Optimierung in der Praxis
- Lesen (freiwillig, für Praktiker aber sehr empfehlenswert):



M. Winand: SQL Performance Explained

ISBN 978-3950307818

Z.B. zu beziehen unter:

https://sql-performance-explained.de/?utm_source=winand.at&utm_campaign=for-developers&utm_medium=web&utm_content=sql-performance-explained

Auch als E-Book erhältlich