Vorlesung Implementierung von Datenbanksystemen

10. Anfrageverarbeitung

Prof. Dr. Klaus Meyer-Wegener
Demian Vöhringer

Wintersemester 2019/20

Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

10.1 Einführung in die Anfrageverarbeitung

- **Realisierung eines mengenorientierten Zugriffs**
  - Nicht mehr Zugriff auf einzelne Sätze
  - Sondern inhaltliche Adressierung einer Menge von Sätzen
  - Reihenfolge der Satzzugriffe nicht vorgegeben
    - Anfrage **deskriptiv** (Was?), nicht prozedural (Wie?)
- **Abbildung von mengenorientierten Operatoren**
  - auf satzorientierte Operatoren
  - und die Benutzung von Indexstrukturen
- **SQL, Relationenalgebra**

Verarbeitung deskriptiver Anfragen

- **Verarbeitungsschritte:**
  - Überprüfung auf syntaktische Korrektheit (komplexe Syntax)
  - Überprüfung von Zugriffsberechtigung und Integritätsbedingungen
  - **Anfrageoptimierung**
    zur Erzeugung einer effizient ausführbaren Folge interner DBS-Operationen
  - Ausführung
- **Zentrale Aufgabe für RDBVS:**
  - Umsetzung deskriptiver Anfragen
    in eine "optimale" Folge
    interner DBS-Operationen (an der Satzschnittstelle)

Probleme bei deskriptiver Anfrageverarbeitung

- **Oder: Warum ist "Optimierung" so schwierig?**
- **Hohe Komplexität:**
  - Auswahlmächtigkeit an der Prädikatenlogik erster Stufe orientiert (inkl. Prädikate wie EXISTS, IS NULL, LIKE u.a.)
  - Unabhängige oder korrelierte Teilanfragen zur Bestimmung von Suchargumenten in beliebiger Schachtelungstiefe
  - Aggregations- und Sortier-Funktionen auf Partitionen der Satzmenge
- **Zusätzliche Anforderungen:**
  - Auch die Änderungsoperationen sind mengenorientiert.
  - Referenzielle Integrität ist aktiv mit Hilfe entsprechender Aktionen zu wahren.
  - Vielfältige Optionen der Datenkontrolle sind zu berücksichtigen.
- **... und was heißt überhaupt "optimal"?**
  - Maximaler Durchsatz, minimale Antwortzeit oder Einhalten von Antwortzeitschranken?

Beispiele deskriptiver SQL-Anfragen

- **Beispiel 1: Anfrage an eine einzelne Tabelle**

```
SELECT PNr, Name, Gehalt
FROM Pers
WHERE Beruf = 'Programmierer'
AND Provision > Gehalt;
```

- **Beispiel 2: Anfrage mit Korrelation**

```
SELECT P.PNr, P.Name, A.AName
FROM Pers P, Abt A
WHERE P.ANr = A.ANr
AND P.Gehalt < (SELECT MAX(Provision) FROM Pers)
AND P.Gehalt > (SELECT AVG(Provision) FROM Pers
                WHERE ANr = P.ANr);
```

  - Verbundoperation
  - Zwei Unteranfragen (**unabhängig** / **korreliert**)

Größeres Beispiel


```
Kunde { KName, KAdr, Kto }
Auftrag { KName, Ware, Menge }
```

```
SELECT Kunde.KName, Kto
FROM Kunde, Auftrag
WHERE Kunde.KName = Auftrag.KName
AND Ware = 'Kaffee';
```

Proj-Liste = `Kunde.KName, Kto`

Sel-Bed = `Kunde.KName = Auftrag.KName AND Ware = 'Kaffee'`

- Relation "Kunde": 100 Tupel; pro Seite 5 Tupel
- Relation "Auftrag": 10.000 Tupel; pro Seite 10 Tupel
- 50 Aufträge betreffen Kaffee.
- Ergebnis-Tupel der Form (KName, Kto): 50 von ihnen passen in eine Seite.
- 3 Ergebnis-Tupel von Kunde x Auftrag passen in eine Seite.
- Puffer bietet für jede Relation genau 1 Kachel.
- Keine Sätze über Seitengrenzen hinweg

Beispiel: direkte Auswertung

1. **R_1 := CROSS (Kunde, Auftrag)**

   Seitenzugriffe (*L* = lesend, *S* = schreibend):

   - *L* : (100 / 5 × 10.000 / 10) = 20.000
   - *S* : (100 × 10.000) / 3 ≈ 333.000

2. **R_2 := SELECT [Sel-Bed] (R_1)**

   - *L* : 333.000
   - *S* : 50 / 3 ≈ 17

3. **ERG := PROJECT [Proj-Liste] (R_2)**

   - *L* : 17
   - *S* : 1

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

Beispiel: verbesserte Auswertung

1. **R_1 := SELECT [Ware = 'Kaffee'] (Auftrag)**
   - *L* : 10.000 / 10 = 1.000
   - *S* : 50 / 10 = 5
2. **R_2 := JOIN [KName = KName] (Kunde, R_1)**
   - *L* : 100 / 5 × 5 = 100
   - *S* : 50 / 3 = 17
3. **ERG := PROJECT [Proj-Liste] (R_2)**
   - *L* : 17
   - *S* : 1
   - Ca. **1.140 Seitenzugriffe**
     - Um Faktor 500 verbessert

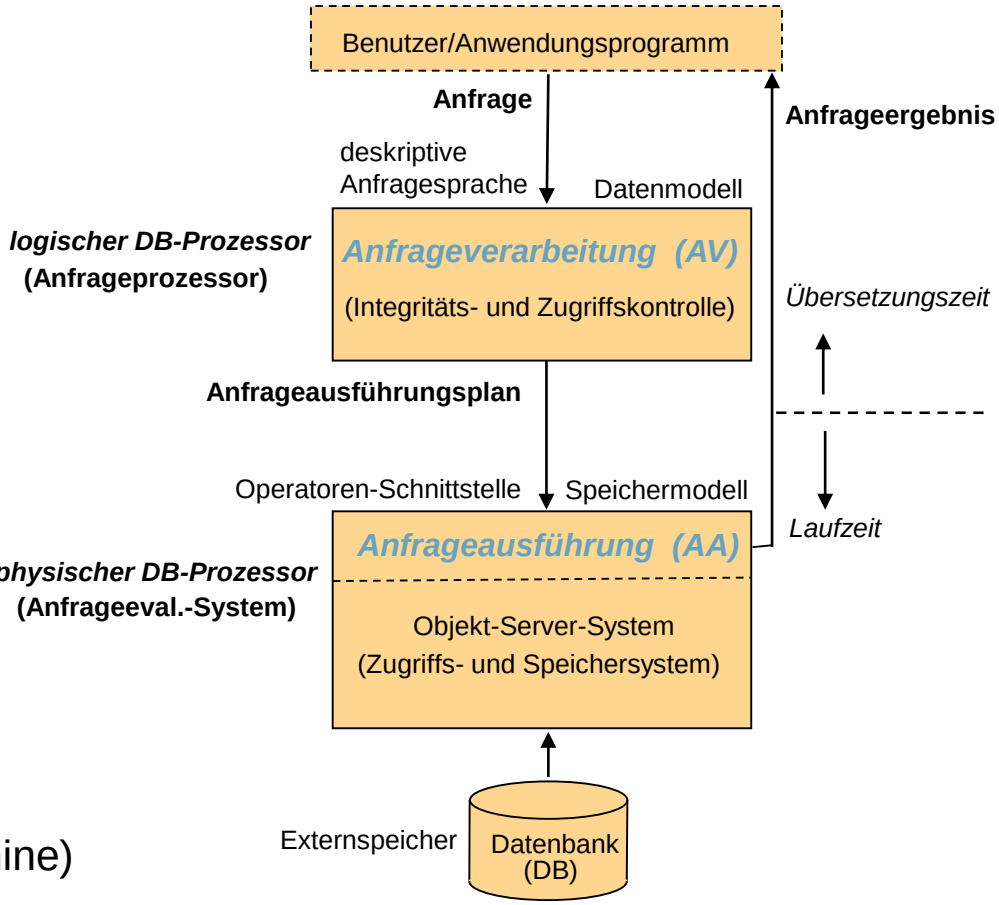
Beispiel: Indexausnutzung

Indexe `I1(Auftrag(Ware))` und `I2(Kunde(KName))`

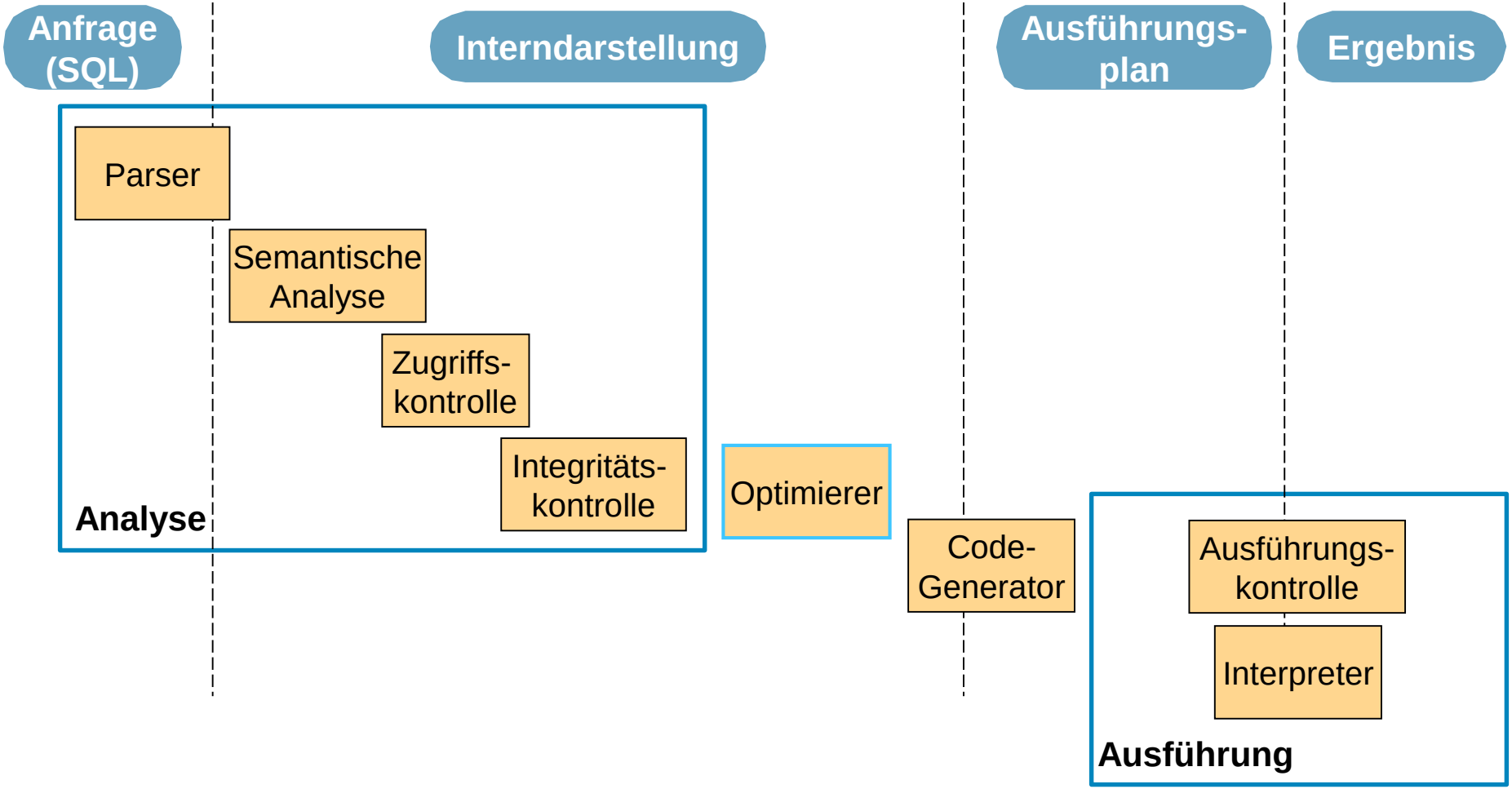
1. **R_1 := SELECT [Ware = 'Kaffee'] (Auftrag)**
   - Über `I1(Auftrag(Ware))`
     - *L* : minimal 5, maximal 50 je nach Index-Art
     - *S* : 50 / 10 = 5
2. **R_2 := sortiere R_1 nach KName**
     - $L + S : 5 \times \log 5 \approx 15$
3. **R_3 := JOIN [KName = KName] (R_2, Kunde)**
     - *L* : 5 + 100 / 5 = 25
     - *S* : 50 / 3 = 17
4. **ERG := PROJECT [Proj-Liste] (R_3)**
     - *L* : 17
     - *S* : 1
   - Maximal ca. 130 und minimal ca. **85 Seitenzugriffe**

10.2 Abstrakte Sicht auf die Anfrageverarbeitung

- **Aufteilung der Anfrageverarbeitung (Query Processing)**
  - **Anfrageverarbeitung (AV)** (im engeren Sinne)
    - **Logischer DB-Prozessor**
    - Liefert einen **Anfrageausführungsplan** (query execution plan; QEP) zur **Übersetzungszeit**
  - **Anfrageausführung (AA)**
    - **Physischer DB-Prozessor**
    - Tatsächliche Ausführung des Anfrageausführungsplans zur **Laufzeit** (Interpretation, virtuelle Maschine)



Phasen der Anfrageverarbeitung – Überblick



Phasen der Anfrageverarbeitung

- **Lexikalische und syntaktische Analyse**
  - Überprüfung auf korrekte Syntax (Parsing)
  - Erstellen eines **Anfragebaums** für die nachfolgenden Übersetzungsschritte (d.h. Überführung in interne Darstellung)
- **Semantische Analyse**
  - Feststellen der Existenz und Gültigkeit der referenzierten Relationen und Attribute (Schema)
  - Ersetzen der externen durch interne Namen (Namensauflösung, Binden)
  - Konvertierung der Werte vom externen Format in interne Darstellung
- **Zugriffs- und Integritätskontrolle**
  - Durchführung einfacher Integritätskontrollen
    - Kontrolle von Formaten und ggf. Konvertierung von Datentypen
  - Generierung von Laufzeitaktionen für werteabhängige Kontrollen

Phasen der Anfrageverarbeitung (2)

- **Standardisierung und Vereinfachung**
  - Überführung des Anfragebaums in eine Normalform
  - Elimination von Redundanzen
- **Restrukturierung und Transformation**
  - **Algebraische** Verbesserung (**Restrukturierung**)
    - Anwendung von heuristischen Regeln
    - Zielt auf globale Verbesserung des Anfragebaums ab
  - **Nicht-algebraische** Verbesserung (**Transformation**)
    - Berücksichtigung ausführbarer Operationen (mit Kosten)
    - Ersetzen und ggf. Zusammenfassen der logischen Operatoren durch **Planoperatoren** (ausführbar)
  - Auswahl der günstigsten Planalternative
    - Meist sind mehrere Planoperatoren als Implementierung eines logischen Operators verfügbar.
    - Meist sind viele Ausführungsreihenfolgen und Zugriffspfade auswählbar.
    - Bewertung der Kosten und Auswahl des günstigsten Ausführungsplans

Phasen der Anfrageverarbeitung (3)

- **Code-Generierung**
  - Generierung eines zugeschnittenen Programms für die vorgegebene (SQL-) Anfrage
    - Zwischencode (früher sogar mal Assembler, heute auch: LLVM)
    - Enthält Aufrufe der Planoperatoren
  - Erzeugung eines ausführbaren **Zugriffsmoduls** (siehe Kap. 7)
  - Verwaltung der Zugriffsmodule in einer DBVS-Bibliothek

10.3 Interndarstellung einer Anfrage

- **Operationelle Betrachtung**
  - Problem: Wie wird eine SQL-Anfrage intern repräsentiert?
- **Relationale Algebra**
  - definiert relationale **logische Operatoren**,
    die für die interne Darstellung einer Anfrage
    in Form eines **Anfragebaums** (**Operatorbaums**) geeignet sind
    - Selektion: Auswahl von "Zeilen"
    - Projektion: Auswahl von "Spalten"
    - Kreuzprodukt: Konkatenation jedes Tupels der einen Relation mit jedem der anderen
    - Verbund: Konkatenation derjenigen Tupel aus zwei Relationen, die eine Bedingung erfüllen
    - Mengenoperatoren
  - erlaubt, Reihenfolge auszudrücken (prozedural)

Die Relationale Algebra

- **Mengenorientierte Operatoren auf einer Relationen**
  - **Selektion**: **SEL (R, pred(…))**
    - Auswahl einer mit **pred**(…) spezifizierten Teilmenge der Tupel von Relation R
    - Beispiel: **SEL (Personen, (Geburtsjahr > 1930))**
    - Wird in der WHERE-Klausel einer SQL-SELECT-Anweisung definiert:

```
SELECT ... FROM Personen
WHERE Geburtsjahr > 1930;
```

  - **Projektion**: **PROJ (R, L)** mit L = (A_1, ..., A_k)
    - Auswahl aller Tupel bezüglich einer Teilmenge L von Attributen der Relation R
    - Eliminierung von Duplikaten
    - Beispiel: **PROJ (Personen, (Vorname, Nachname))**
    - Wird in der SELECT-Klausel einer SQL-SELECT-Anweisung definiert:

```
SELECT Vorname, Nachname
FROM Personen ... ;
```

Die Relationale Algebra (2)

- **Mengenorientierte Operatoren auf zwei Relationen**
  - **Kreuzprodukt** zweier Relationen $R(A_1,...,A_n)$ und $S(B_1,...,B_m)$: **CROSS (R, S)**
    - Konkatenation jedes Tupels der Relation R mit jedem Tupel der Relation S
    - Beispiel: **CROSS (Personen, Filme)**
    - Wird in der FROM-Klausel definiert:

```
SELECT ... FROM Personen, Filme;
```

  - **Verbund** zweier Relationen $R(A_1,...,A_n)$ und $S(B_1,...,B_m)$:
    JOIN (R, S, pred) mit pred = $P(A_i, B_j)$
    - Verbinden zweier Relationen R und S gemäß eines Prädikats $P(A_i, B_j)$ über Attributen aus beiden Relationen
      - Eigentlich schon eine Optimierung, da auch durch Kreuzprodukt und Selektion darstellbar
    - Beispiel: **JOIN (Personen, Filme, (Nachname = Hauptdarstellername))**
    - Wird in der FROM-Klausel definiert:

```
SELECT ...
FROM Personen JOIN Filme
        ON Nachname = Hauptdarstellername;
```

Die Relationale Algebra (3)

- **Mengenorientierte Operatoren auf zwei Relationen (Forts.)**
  - **Mengenoperatoren:**

R ∪ S	**bzw. UNION (R, S)**
R ∩ S	**bzw. INTERSECT (R, S)**
R \ S	**bzw. EXCEPT (R, S)**
...	

- **Merke: auf logischer Ebene auch n-stellige Operatoren !**
  - Zurückführen auf eine Sequenz von binären

Die Relationale Algebra (4)

- **Multimengen-orientierte Operatoren**
  - Multimengen (engl. bags) haben Performance-Vorteile:
    - Bei Vereinigung einfach beide Operanden ausgeben
    - Bei Projektion nach Bearbeitung aller einzelnen Tupel fertig
  - Außerdem bei manchen Aggregationen (AVG, COUNT) explizit gewünscht
  - Also dafür **eigene (logische) Operatoren** – mit anderer Wirkung
    - Elemente (Tupel) kommen in den Multimengen mit einer best. Häufigkeit *n* vor; dabei ist $n > 1$ erlaubt.
    - UNION: Häufigkeiten addieren
    - INTERSECT: Minimum der Häufigkeiten
    - EXCEPT: max(0, n – m)
    - PROJECT: wie oben, nur ohne Duplikat-Eliminierung
    - SELECT, CROSS, JOIN: unverändert

Die Relationale Algebra (5)

- **Weitere Operatoren:**
  - Durch Anfragesprachen wie SQL hinzugekommen
  - **Umbenennung:**
    - **RENAME (R, [Rnew,] ((A1, A1new), (A2, A2new), …))**
  - **Duplikat-Eliminierung**:
    - **DUP-ELIM (R)**
  - **Aggregation**:
    - **SUM (R, attr), AVG (R, attr), MIN (R, attr), MAX (R, attr)**
      - Für numerische Attribute, bei MIN und MAX auch Zeichenketten
    - **COUNT (R)**
      - Anzahl der Tupel
  - **Gruppierung**:
    - **GROUP (R, L, agg)** mit Gruppierungsattributen L = (A_1, ..., A_k) und Aggregationen agg = ((AGG_1 (attr), $name_1$), (AGG_2 (attr), $name_2$), …)

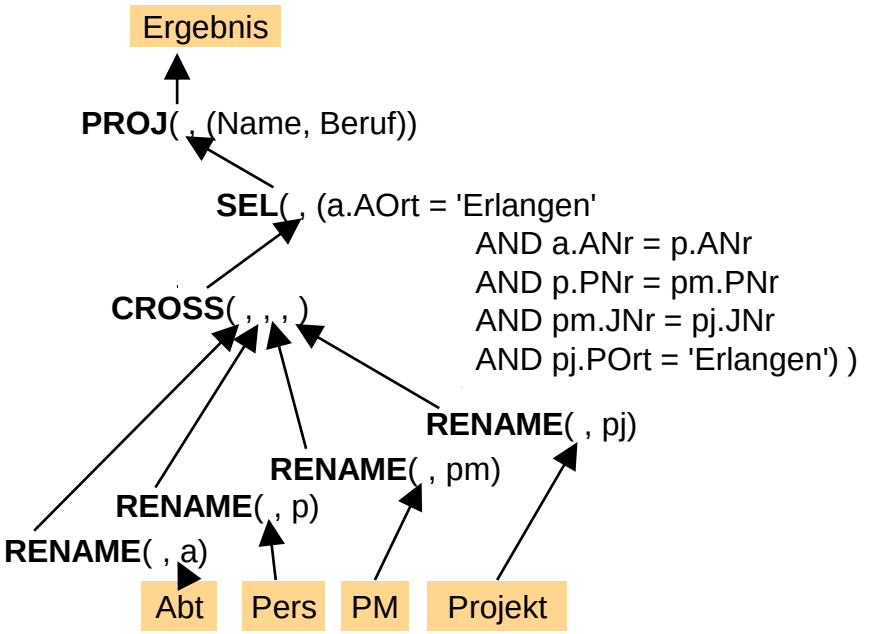
Die Relationale Algebra (6)

- **Weitere Operatoren (Forts.):**
  - **Erweiterte Projektion:**
    - **G-PROJ (R, L)** mit L = (name$_1$ = expr$_1$, name$_2$ = expr$_2$, …) Liste von Ausdrücken zur Berechnung von neuen Attributwerten
  - **Sortierung**:
    - **SORT (R, L)** mit L = (A_1, ..., A_k) Liste der Attribute, nach denen sortiert wird
    - Ergebnis ist Liste! Also nur ganz an Ende sinnvoll.
  - **Äußerer Verbund**:
    - **OUTER-JOIN (R, S, pred, case)** mit pred = P(A_i, B_j) wie beim normalen Join und case $\in$ {left, right, full}

Operatorbaum

- **Effiziente Datenstruktur mit geeigneten Zugriffsfunktionen**
  - Prozedurale Darstellung einer deskriptiven, mengenorientierten Anfrage
  - **Knoten** sind Operatoren der Relationalen Algebra.
  - **Blattknoten** sind (üblicherweise) Relationen.
  - Gerichtete **Kanten** repräsentieren den Datenfluss.
- **Beispiel**

```
SELECT Name, Beruf
FROM Abt a, Pers p,
     PM pm, Projekt pj
WHERE a.ANr = p.ANr
AND a.AOrt = 'Erlangen'
AND p.PNr = pm.PNr
AND pm.JNr = pj.JNr
AND pj.POrt = 'Erlangen';
```



10.4 Standardisierung und Vereinfachung

- **Standardisierung (der Qualifikationsbedingungen)**
  - Wahl einer Normalform
    - Konjunktive Normalform **$(P_{11}$ OR ... OR $P_{1n})$ AND ... AND $(P_{m1}$ OR ... OR $P_{mp})$**
    - Disjunktive Normalform **$(P_{11}$ AND ... AND $P_{1q})$ OR ... OR $(P_{r1}$ AND ... AND $P_{rs})$**
    - Pränex-Normalform (Verschiebung der Quantoren)
      z.B. zur Auflösung geschachtelter SELECT-Anweisungen
- **Vereinfachung**
  - Äquivalente Ausdrücke können einen unterschiedlichen Grad an Redundanz besitzen.
    - Idempotenzregeln
    - Ausdrücke mit "leeren Relationen"
  - Behandlung / Eliminierung gemeinsamer Teilausdrücke

$$(A_1 = a_{11} \text{ OR } A_1 = a_{12}) \text{ AND } (A_1 = a_{12} \text{ OR } A_1 = a_{11})$$

Standardisierung und Vereinfachung (2)

- **Vereinfachung (Forts.)**
  - Konstanten-Propagierung (allg. Hüllenbildung der Qualifikationsprädikate)

```
A <op> B AND B = const.
⇒    A <op> const.
```

  - Nicht erfüllbare Ausdrücke

```
A ≥ B AND B > C AND C ≥ A
⇒    A > A    ⇒ false
```

  - Nutzung von Information über semantische Integritätsbedingungen
    - A ist Primärschlüssel: **project**[A] → keine Duplikateliminierung erforderlich
    - Auswertung hinterlegter Regeln:

```
IF Fam-Stand = "verh." THEN Steuerklasse ≥ 3;
Fam-Stand = "verh." AND Steuerklasse = 1    ⇒ false
```

  - Umformungs- und Idempotenzregeln für Boole'sche Ausdrücke
  - Umformungsregeln für quantifizierte Ausdrücke

10.5 Restrukturierung

- **Äquivalente Umformung des Operatorbaums**
  - Man kann sich die effizientere Variante aussuchen!
- **Regeln:**
  - **(1) Ein n-facher Verbund kann durch eine Folge von binären Verbunden ersetzt werden und umgekehrt:**

```
JOIN (R1, R2, ..., Rn,)) =
JOIN (pred(R1, R2, ..., Rn
  ...
    JOIN (
      JOIN (
        JOIN (R1, R2, pred(R1, R2)),
        R3, pred(R1, R2, R3)
      ),
      R4, pred(R1, R2, R3, R4)
    ),
  ...,
  Rn, pred(R1, R2, ..., Rn)
)
```

Restrukturierung (2)

- **Regeln (Forts.):**
  - (2) **Verbund ist kommutativ.**
  - (3) **Verbund ist assoziativ.**
  - (4) **Selektionen können zusammengefasst werden:**
    `SEL(SEL(R,pred1),pred2) = SEL(R,(pred1 AND pred2))`
  - (5) **Projektionen können zusammengefasst werden:**
    `PROJ(PROJ(R,L1),L2) = PROJ(R,L2)`
  - (6) **Projektion dürfen (in erweiterter Form) vorgezogen werden:**
    `PROJ(SEL(R,pred(M)),L) = PROJ(SEL(PROJ(R,(L∪M)),pred(M)),L)`
  - (7) **Selektion und Verbund dürfen vertauscht werden:**
    `SEL(JOIN(R,S,pred1),pred2(R)) = JOIN(SEL(R,pred2),S,pred1)`
  - (8) **Selektion darf mit Vereinigung und Differenz vertauscht werden:**
    `SEL(UNION(R,S),pred) = UNION(SEL(R,pred),SEL(S,pred))`
  - (9) **Selektion und Kreuzprodukt können zu Verbund zusammengefasst werden:**
    `SEL(CROSS(R,S),pred) = JOIN(R,S,pred)`
  - (10) **...**

Nur Beispiele!
Es gibt noch viel mehr!

Algorithmus zur Restrukturierung

- **Ziel: Zwischenergebnisse möglichst klein halten**
  - Vor allem Kreuzprodukt vermeiden

- **Vereinfachte Vorgehensweise (Heuristik)**
  - Komplexe Verbundoperationen zerlegen in binäre Verbunde (Bilden von binären Verbunden, Regel 1)
  - Selektionen mit mehreren Prädikat-Termen separieren in Selektionen mit jeweils einem Prädikat-Term (Regel 4)
  - **Selektionen so früh wie möglich ausführen**,
    - d.h. Selektionen hinunterschieben zu den Blättern des Anfragebaums (engl. selection push-down, Regeln 7 und 8)
  - **Selektionen und Kreuzprodukt zu Verbund zusammenfassen**,
    - wenn das Selektionsprädikat Attribute aus den beiden Relationen verwendet (Regel 9)

Algorithmus zur Restrukturierung (2)

- **Vereinfachte Vorgehensweise (Forts.)**
  - Einfache Selektionen wieder zusammenfassen,
    d.h. aufeinanderfolgende Selektionen (derselben Relation) gruppieren
    (Regel 4)
  - **Projektionen so früh wie möglich ausführen**,
    - d.h. Projektionen hinunterschieben zu den Blättern des Anfragebaums
      (engl. projection push-down, Regel 6)
    - Dabei aber die teure Duplikat-Eliminierung vermeiden!