8. Relationenoperationen – Implementierung*

- **Operationen der Relationenalgebra**
  - unäre Operationen: π, σ
  - binäre Operationen: $\bowtie$, $\times$, $\div$, $\cap$, $\cup$, -

  ➥ SQL-Anfragen enthalten logische Ausdrücke, die auf die Operationen der Relationenalgebra zurückgeführt werden können. Sie werden in Zugriffspläne umgesetzt. Sog. Planoperatoren implementieren diese logischen Operationen

- **Planoperatoren auf einer Relation**

- **Selektion**

- **Operatoren auf mehreren Relationen**

- **Verbundalgorithmen**
  - Nested-Loop-Verbund
  - Sort-Merge-Verbund
  - Hash-Verbund
    (Classic Hashing, Simple-Hash-Join, Hybrid-Hash-Join)
  - Nutzung von typübergreifenden Zugriffspfaden
  - verteilte Verbundalgorithmen

- **weitere binäre Operationen (Mengenoperationen)**

---

* Graefe, G.: Query Evaluation Techniques for Large Databases, ACM Computing Surveys 25:2, June 1993

Planoperatoren auf einer Relation

- **Planoperatoren zur Selektion**
  Allgemeine Auswertungsmöglichkeiten:
  - direkter Zugriff über ein gegebenes TID, über ein Hash-Verfahren oder eine ein- bzw. mehrdimensionale Indexstruktur
  - sequentielle Suche in einer Relation
  - Suche über eine Indexstruktur (Indextabelle, Bitliste)
  - Auswahl mit Hilfe mehrerer Verweislisten, wobei mehr als eine Indexstruktur ausgenutzt werden kann
  - Suche über eine mehrdimensionale Indexstruktur

- **Projektion**
  wird typischerweise in Kombination mit Sortierung, Selektion
  oder Verbund durchgeführt

- **Planoperatoren zur Modifikation**
  - Änderungen sind in SQL mengenorientiert, aber auf eine Relation beschränkt
  - INSERT, DELETE und UPDATE werden direkt auf die entsprechenden Operationen der Speicherungsstrukturen abgebildet
  - „automatische“ Abwicklung von Wartungsoperationen
    zur Aktualisierung von Zugriffspfaden,
    zur Gewährleistung von Cluster-Bildung und Reorganisation usw.
  - Durchführung von Logging- und Recovery-Maßnahmen usw.

Planoperatoren für die Selektion

- **Nutzung des Scan-Operators**
  - Definition von Start- und Stopp-Bedingung
  - Definition von einfachen Suchargumenten

- **Planoperatoren**
  1. *Relationen-Scan*
     - immer möglich
     - SCAN-Operator implementiert die Selektionsoperation
  2. *Index-Scan*
     - Auswahl des kostengünstigsten Index
     - Spezifikation des Suchbereichs (Start-, Stop-Bedingung)
  3. *k-d-Scan*
     - Auswertung mehrdimensionaler Suchkriterien
     - Nutzung verschiedener Auswertungsrichtungen durch Navigation
  4. *TID-Algorithmus*
     - Auswertung aller „brauchbaren" Indexstrukturen
     - Auffinden von variabel langen TID-Listen
     - Boolesche Verknüpfung der einzelnen Listen
     - Zugriff zu den Tupeln entsprechend der Trefferliste

- **Weitere Planoperatoren in Kombination mit der Selektion**
  - Sortierung
  - Gruppenbildung
    (siehe Sortieroperator)
  - spezielle Operatoren z. B. in Data-Warehouse-Anwendungen zur Gruppen- und Aggregatbildung (CUBE-Operator)

Operatoren über mehrere Relationen

- **SQL erlaubt komplexe Anfragen über k Relationen**
  - **Ein-Variablen-Ausdrücke:**
    beschreiben Bedingungen für die Auswahl von Elementen aus einer Relation.
  - **Zwei-Variablen-Ausdrücke:**
    beschreiben Bedingungen für die Kombination von Elementen aus zwei Relationen.
  - k-Variablen-Ausdrücke werden typischerweise in Ein- und Zwei-Variablen-Ausdrücke zerlegt und durch entsprechende Planoperatoren ausgewertet

- **Planoperatoren über mehrere Relationen**

  Allgemeine Auswertungsmöglichkeiten:

  - **Schleifeniteration** (*nested iteration*)

    für jedes Element der äußeren Relation R_a Durchlauf der inneren Relation R_i

    - $O(N_a \cdot N_i + N_a)$
    - wichtigste Anwendung: *nested loops join*

  - **Mischmethode** (merge method)

    sequentieller, schritthaltender Durchlauf beider Relationen R1, R2

    - $O(N_1 + N_2)$
    - ggf. zusätzliche Sortierkosten
    - wichtigste Anwendung: *merging join*

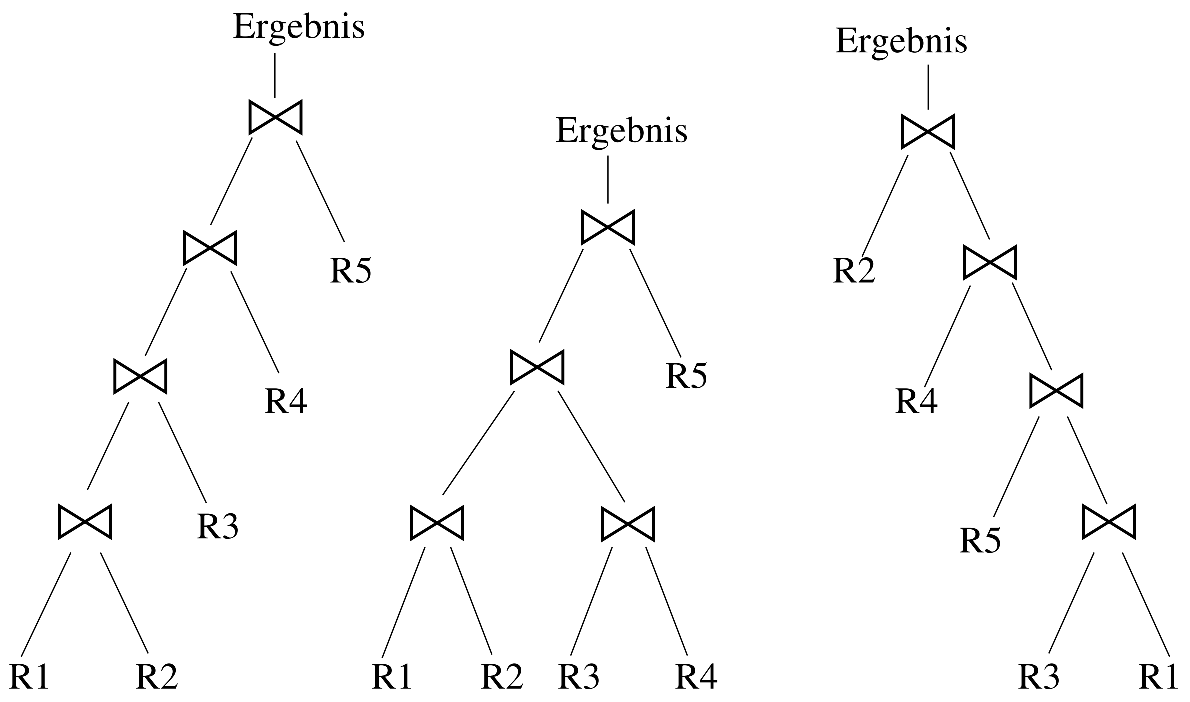
  - **Hash-Methode** (*hashing*)

    Partitionierung der inneren Relation R_i. Laden der p Partitionen in eine Hash-Tabelle HT im HSP. „Probing" der äußeren Relation R_a oder ihrer entsprechenden Partitionen mit HT: ➥ $O(p \cdot N_a + N_i)$

Operatoren über mehrere Relationen (2)

- **n-Wege-Verbunde**
  - Zerlegung in n-1 Zwei-Wege-Verbunde
  - Anzahl der Verbundreihenfolgen ist abhängig von den gewählten Verbundattributen
  - n! verschiedene Reihenfolgen möglich
  - Einsatz von Pipelining-Techniken
  - Optimale Auswertungsreihenfolge abhängig von
    - Planoperatoren
    - „passende" Sortierordnungen für Verbundattribute
    - Größe der Operanden usw.

- **Verschiedene Verbundreihenfolgen mit Zwei-Wege-Verbunden (n=5)**



- **Analoge Vorgehensweise bei Mengenoperationen**

Planoperatoren für den Verbund

- **Verbund**
  - satztypübergreifende Operation: gewöhnlich sehr teuer
  - häufige Nutzung: wichtiger Optimierungskandidat
  - typische Anwendung: Gleichverbund
  - allgemeiner Θ-Verbund selten

- **Implementierung der Verbundoperation kann gleichzeitig Selektionen auf den beteiligten Relationen R und S ausführen**

```
SELECT   *
FROM     R, S
WHERE    R.VA Θ S.VA
         AND P_R
         AND P_S
```

  - VA: Verbundattribute
  - P_R und P_S: Prädikate definiert auf Selektionsattributen (SA) von R und S

- **Mögliche Zugriffspfade**
  - Scans über R und S (immer)
  - Scans über I_R(VA), I_S(VA) (wenn vorhanden)
    ➥ liefern Sortierreihenfolge nach VA
  - Scans über I_R(SA), I_S(SA) (wenn vorhanden)
    ➥ ggf. schnelle Selektion für P_R und P_S
  - Scans über andere Indexstrukturen (wenn vorhanden)
    ➥ ggf. schnelleres Auffinden aller Sätze

Nested-Loop-Verbund

- **Annahmen:**
  - Sätze in R und S sind nicht nach den Verbundattributen geordnet
  - es sind keine Indexstrukturen $I_R(VA)$ und $I_S(VA)$ vorhanden

- **Algorithmus für Θ-Verbund:**

  Scan über S,
  für jeden Satz s, falls P_S:
      Scan über R,
      für jeden Satz r, falls P_R AND (r.VA Θ s.VA):
          führe Verbund aus,
          d. h., übernehme kombinierten Satz (r, s) in die Ergebnismenge.

- **Komplexität: O (N^2)**

  (Kardinalität N für R und S)

- **Nested-Loop-Verbund mit Indexzugriff**

  Scan über S,
  für jeden Satz s, falls P_S:
      ermittle mittels Zugriff auf $I_R(VA)$ alle TIDs für Sätze mit r.VA = s.VA,
      für jedes TID:
          hole Satz r, falls P_R:
              übernehme kombinierten Satz (r, s) in die Ergebnismenge.

- **Nested-Block-Verbund**

  Scan über S,
  für jede Seite (bzw. Menge aufeinanderfolgender Seiten) von S:
      Scan über R,
      für jede Seite (bzw. Menge aufeinanderfolgender Seiten) von R:
          für jeden Satz s der S-Seite, falls P_S:
              für jeden Satz r der R-Seite,
                falls P_R AND (r.VA Θ s.VA):
                    übernehme kombinierten Satz (r, s) in die Ergebnismenge.

Sort-Merge-Verbund

- **Algorithmus besteht aus 2 Phasen:**

  - *Phase* 1: Sortierung von R und S nach R(VA) und S(VA) (falls nicht bereits vorhanden), dabei frühzeitige Eliminierung nicht benötigter Sätze (➥ P_R, P_S)

  - *Phase 2:* schritthaltende Scans über sortierte R- und S-Sätze mit Durchführung des Verbundes bei r.VA = s.VA

- **Komplexität: O(N log N)**

- **Spezialfall**

  Falls I_R(VA) und I_S(VA) oder verallgemeinerte Zugriffspfad-struktur über R(VA) und S(VA) (Join-Index) vorhanden:

  ➥ *Ausnutzung von Indexstrukturen auf Verbundattributen*:

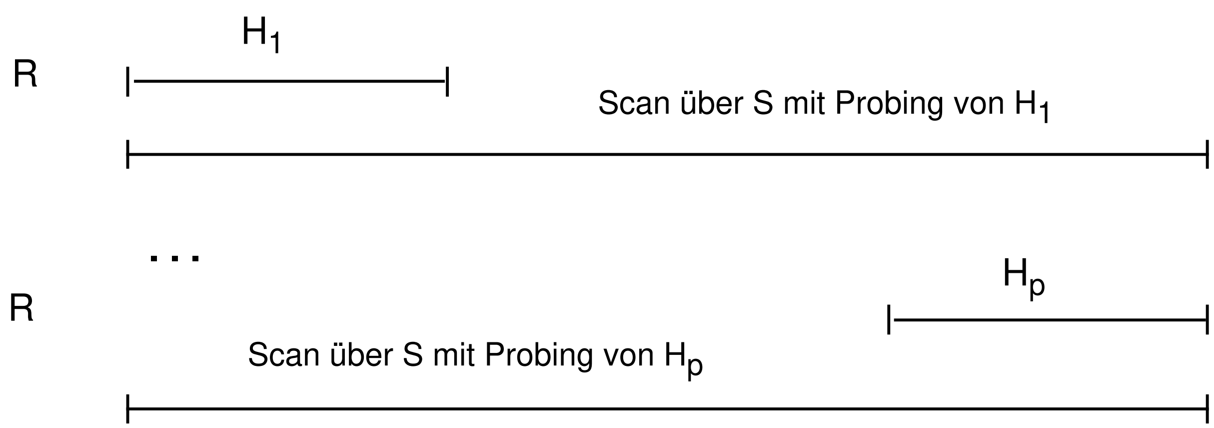
  Schritthaltende Scans über I_R(VA) und I_S(VA):
      für jeweils zwei Schlüssel aus I_R(VA) und I_S(VA), falls r.VA = s.VA:
      hole mit den zugehörigen TIDs die Tupel,
          falls P_R und P_S:
          übernehme kombinierten Satz (r, s) in die Ergebnismenge.

Hash-Verbund

- **Einfachster Fall (classic hashing):**
  - *Schritt 1*: Abschnittsweises Lesen der (kleineren) Relation R und Aufbau einer Hash-Tabelle mit $h_A(r(VA))$ nach Werten von R(VA) entsprechend den Abschnitten R_i ($1 \leq i \leq p$), so daß jeder der p Abschnitte in den verfügbaren Hauptspeicher paßt und jeder Satz P_R erfüllt
  - *Schritt 2*: Überprüfung (Probing) für jeden Satz von S mit P_S; im Erfolgsfall Durchführung des Verbundes
  - *Schritt 3*: Wiederhole Schritt 1 und 2 solange, bis R erschöpft ist.

- **Aufbau der Hash-Tabelle und Probing**

  Es erfolgt ein Scan über R; dabei wird die Hash-Tabelle H_i ($1 \leq i \leq p$) der Reihe nach im HSP aufgebaut



- **Komplexität: O(p · N)**

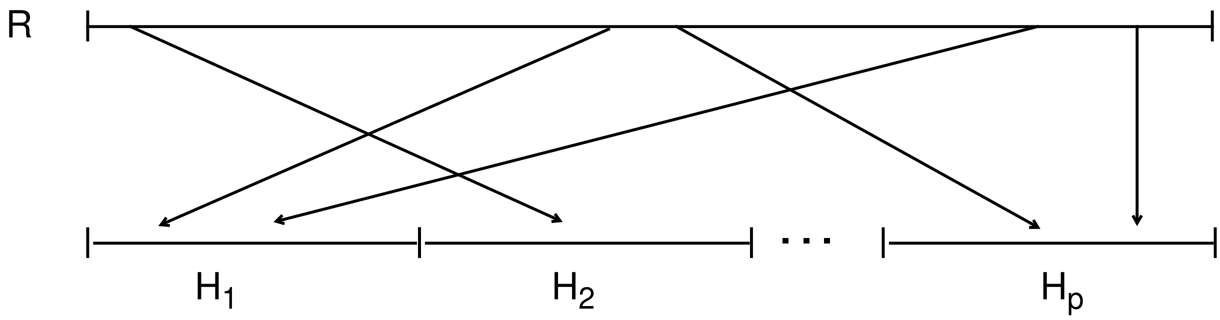
- **Spezialfall:**

  R paßt in den Hauptspeicher: eine Partition (p = 1)

  ➥ ein Scan über S genügt

Hash-Verbund (2)

- **Partitionierung von R mit Hash-Funktion h_P**



  - Partitionierung von R in Teilmengen R_1, R_2, ..., R_p:
    Ein Satz r von R ist in R_i, wenn h(r) in H_i ist.

  ➥ Warum ist diese Partitionierung eine kritische Operation ?
  Welche Hilfsoperationen können erforderlich sein ?
  Ist für die Partitionierung der Einsatz einer Hash-Funktion notwendig ?

  - Relation S wird mit derselben Funktion h_P
    unter Auswertung von P_S partitioniert

- **Varianten des Hash-Verbundes:**
  Sie unterscheiden sich vor allem durch die Art der Partitionsbildung

- **Simple-Hash-Join**

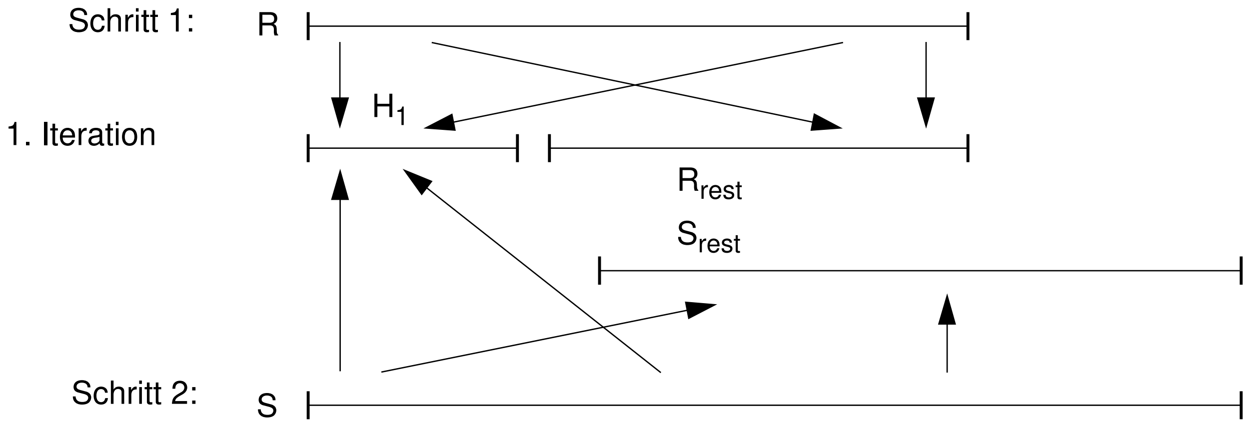
  - *Schritt 1*: Führe Scan auf kleinerer Relation R aus, überprüfe P_R und wende auf jedes qualifizierte Tupel r die Hash-Funktion h_P an. Fällt h_P(r(VA)) in den gewählten Bereich, trage es in H_i ein. Anderenfalls schreibe r in einen Puffer für die Ausgabe in eine Datei für „übergangene" r-Tupel.

  - *Schritt 2:* Führe Scan auf S aus, überprüfe P_S und wende auf jedes qualifizierte Tupel s die Hash-Funktion h_P an. Fällt h_P(s(VA)) in den gewählten Bereich, suche in H_i einen Verbundpartner (Probing). Falls erfolgreich, bilde ein Verbundtupel und ordne es dem Ergebnis zu. Anderenfalls schreibe s in einen Puffer für die Ausgabe in eine Datei für „übergangene" s-Tupel.

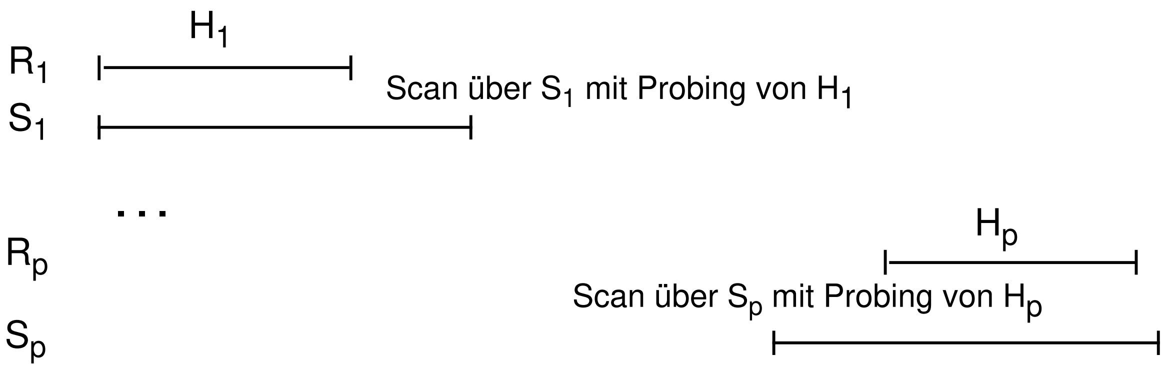
  - *Schritt 3*: Wiederhole Schritt 1 und 2 mit den bisher übergangenen Tupeln solange, bis R erschöpft ist. Dabei ist die Überprüfung von P_R und P_S nicht mehr erforderlich.

Hash-Verbund (3)

- **Partitionierungstechnik beim Simple-Hash-Join:**
  gezeigt an Aufbau und Probing von H_1



- **Verbesserung der Partitionsbildung**
  (Bei Grace-Join und Hybrid-Hash-Join)
  - Partitionsbildung findet vor dem Verbund statt
  - Partitionen R_i und S_i sind in Dateien zwischengespeichert
  - Aufbau von H_i im HSP mit R_i und Probing mit S_i

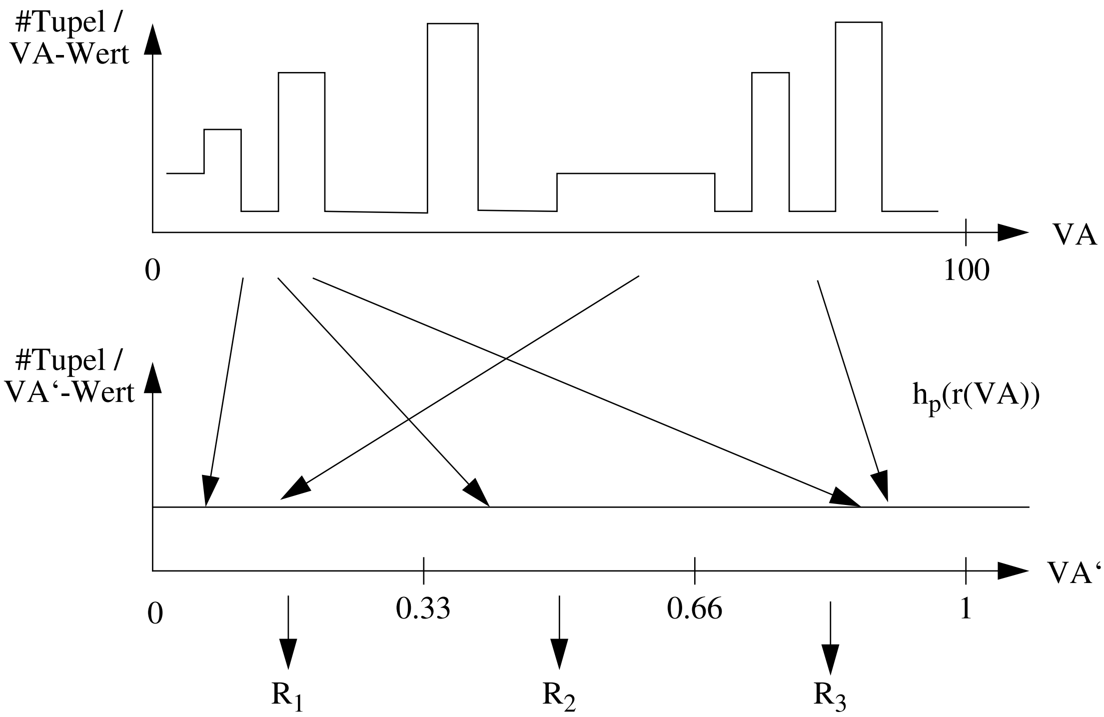


  - Hybrid-Hash-Join optimiert das Verfahren dadurch, daß während der Partitionsbildung Aufbau und Probing von H_1 erfolgt

Hash-Verbund – Beispiel

I. Partitionieren

a) Partitionieren von R mit $h_p(r(VA))$



b) Partitionieren von S mit $h_p(s(VA))$

II. Verbund

1)

H_1

R1 — in M mit $h_H(r(VA))$

VA‘: 0.0 – 0.33

S1

VA‘: 0.0 – 0.33

Lesen, Probing mit $h_H(s(VA))$

2)

H_2

R2

VA‘: 0.34 – 0.66

S2

VA‘: 0.34 – 0.66

3)

H_3

R3

VA‘: 0.67 – 1.0

S3

VA‘: 0.67 – 1.0

Nutzung von typübergreifenden Zugriffspfaden

- **Verbund über Link-Strukturen**

  Ausnutzung hierarchischer Zugriffspfade für den Gleichverbund

  Scan über R (Owner-Relation),
  für jeden Satz r, falls P_R:
  Scan über zugehörige Link-Struktur $L_{R\text{-}S}(VA)$,
  für jeden Satz s, falls P_S:
  übernehme kombinierten Satz (r, s) in die Ergebnismenge.

- **Weitere Verfahren**

  - **Verbundindexe**, die für bestimmte Θ-Verbunde eingerichtet sind

R	S
TID_{r2}	TID_{s4}
TID_{r1}	TID_{s3}
TID_{r2}	TID_{s2}
TID_{r2}	TID_{s6}

logische Sicht

VI_R:

R	S
TID_{r1}	TID_{s3}
TID_{r2}	TID_{s2}
TID_{r2}	TID_{s4}
TID_{r2}	TID_{s6}

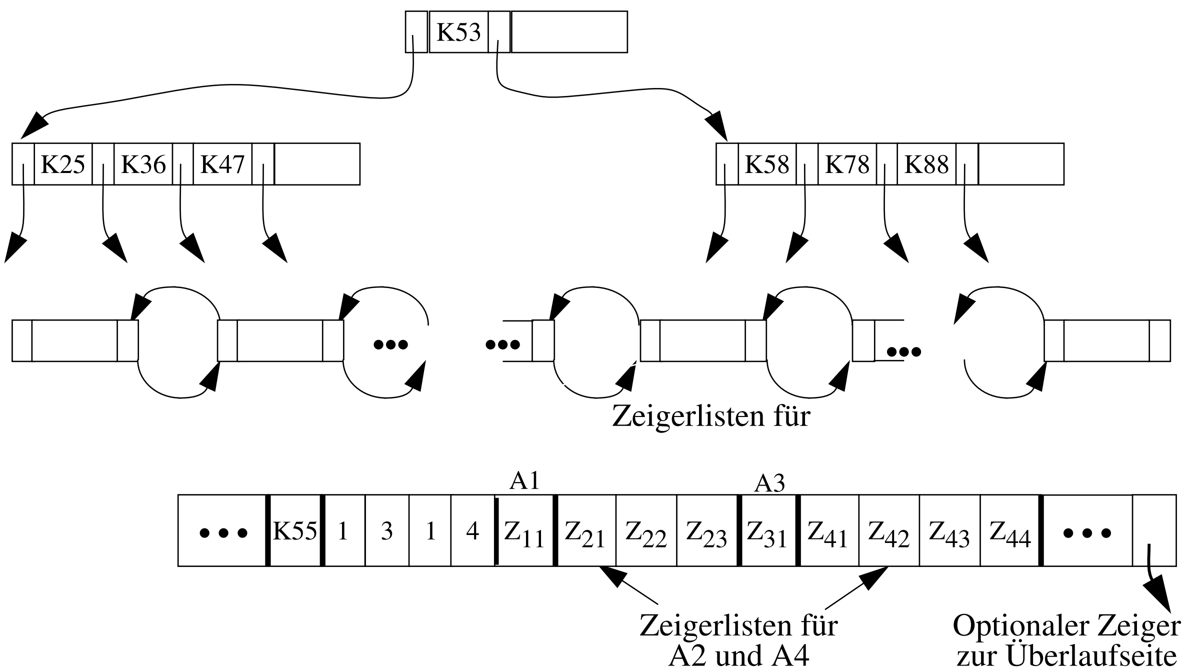
Index auf TID_R

VI_S:

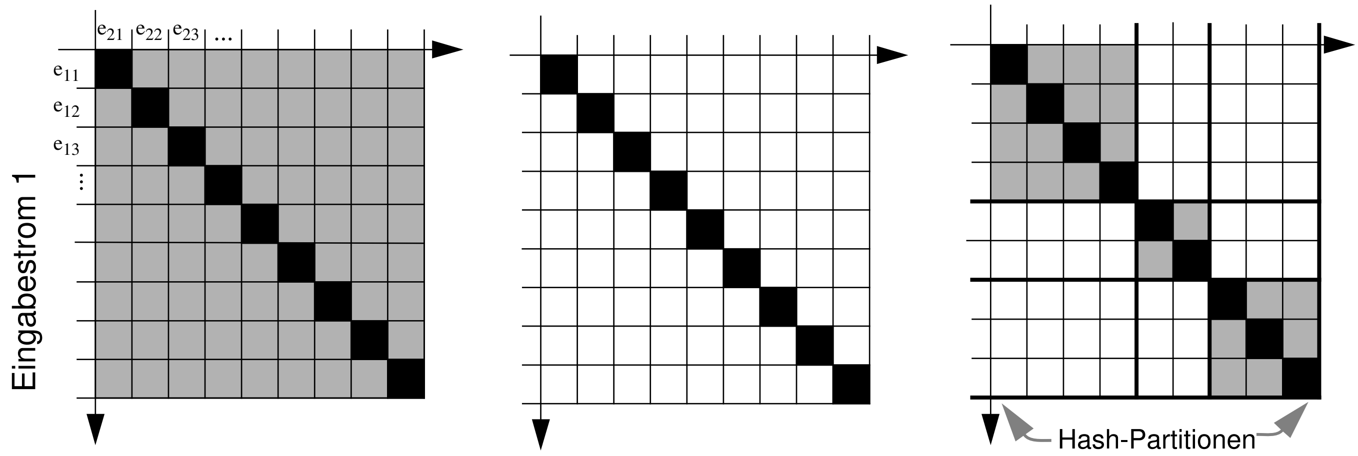
S	R
TID_{s2}	TID_{r2}
TID_{s3}	TID_{r1}
TID_{s4}	TID_{r2}
TID_{s6}	TID_{r2}

Index auf TID_S

  - Nutzung von **verallgemeinerten Zugriffspfadstrukturen**



Verbundalgorithmen – Vergleich



(a) Nested-Loop-Verbund (b) Merge-Verbund (c) Hash-Verbund

Elementvergleich

Elementvergleich, der zu einemVerbundelement führt

- Nested-Loop-Verbund ist immer anwendbar, jedoch ist dabei stets das vollständige Durchsuchen des gesamten Suchraums in Kauf zu nehmen

- Merge-Verbund benötigt die geringsten Suchkosten, verlangt aber, daß die Eingabeströme bereits sortiert sind. Indexstrukturen auf beiden Verbundattributen erfüllen diese Voraussetzung. Sonst reduziert das Sortieren beider Relationen nach den Verbundattributen den Kostenvorteil in erheblichem Maße. Ein Sort-Merge-Verbund kann dennoch zusätzliche Vorteile besitzen, falls das Ergebnis in sortierter Folge verlangt wird und das Sortieren des großen Ergebnisses aufwendiger ist als das Sortieren zweier kleiner Ergebnismengen.

- Beim Hash-Verbund wird der Suchraum partitioniert. In Bild c ist unterstellt, daß die gleiche Hash-Funktion h auf die Relationen R und S angewendet worden ist. Die Partitionsgröße (bei der kleineren) Relation richtet sich nach der verfügbaren Puffergröße im Hauptspeicher. Eine Verkleinerung der Partitionsgröße, um den Fall b anzunähern, verursacht höhere Vorbereitungskosten und ist deshalb nicht zu empfehlen

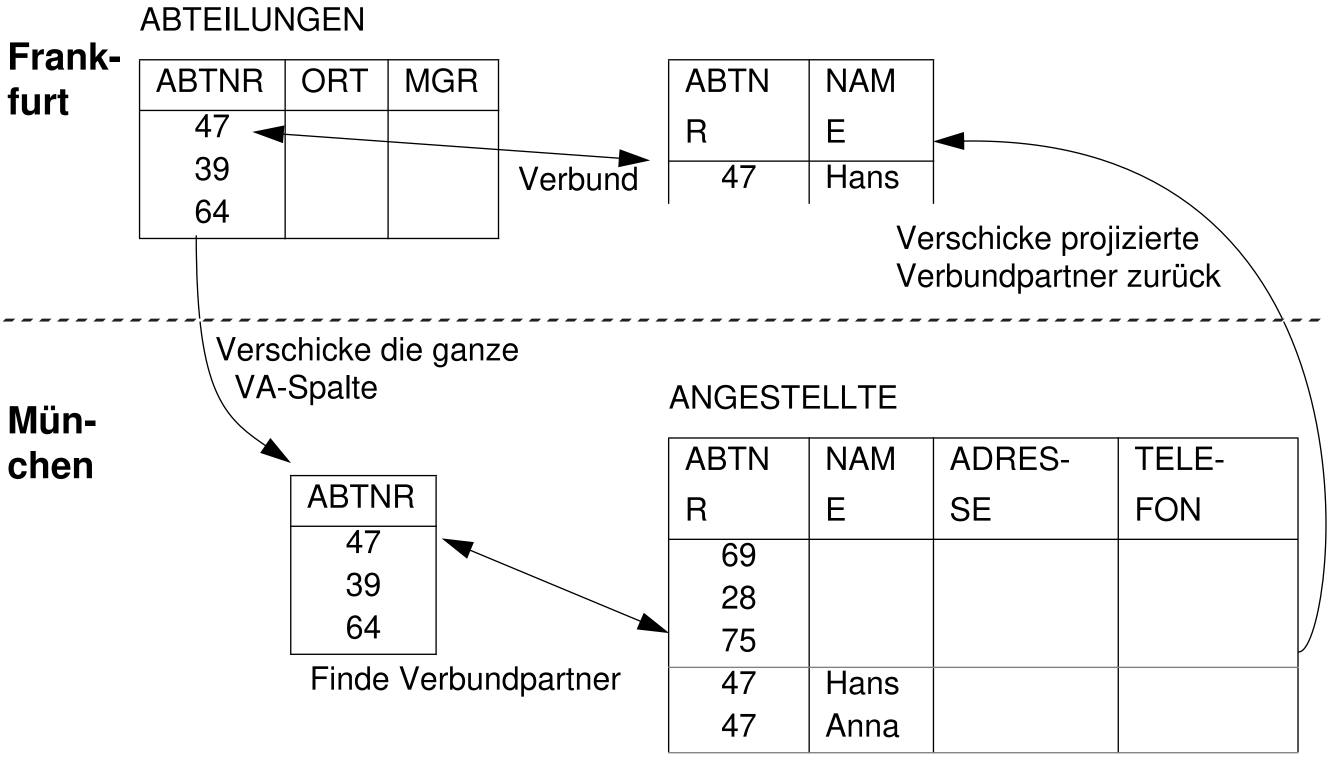
Verbundalgorithmen in verteilten DBS

- **Problemstellung:** Anfrage in Knoten K, die einen Verbund zwischen (Teil-) Relationen R am Knoten K_R und (Teil-) Relation S am Knoten K_S erfordert
- Festlegung des Ausführungsknotens: K, K_R oder K_S
- **Bestimmung der Auswertestrategie**
  - Sende beteiligte Relationen vollständig an einen Knoten und führe lokale Verbundberechnung durch (*„Ship Whole“*)
    - minimale Nachrichtenanzahl
    - sehr hohes Übertragungsvolumen
  - Fordere für jeden Verbundwert der ersten Relation zugehörige Tupel der zweiten an (*“Fetch as Needed“*)
    - hohe Nachrichtenanzahl
    - nur relevante Tupel werden berücksichtigt
  - Kompromißlösung:
    - Semi-Verbund bzw. Erweiterungen wie Bit-Vektor-Verbund (*Hash-Filter-Join*)

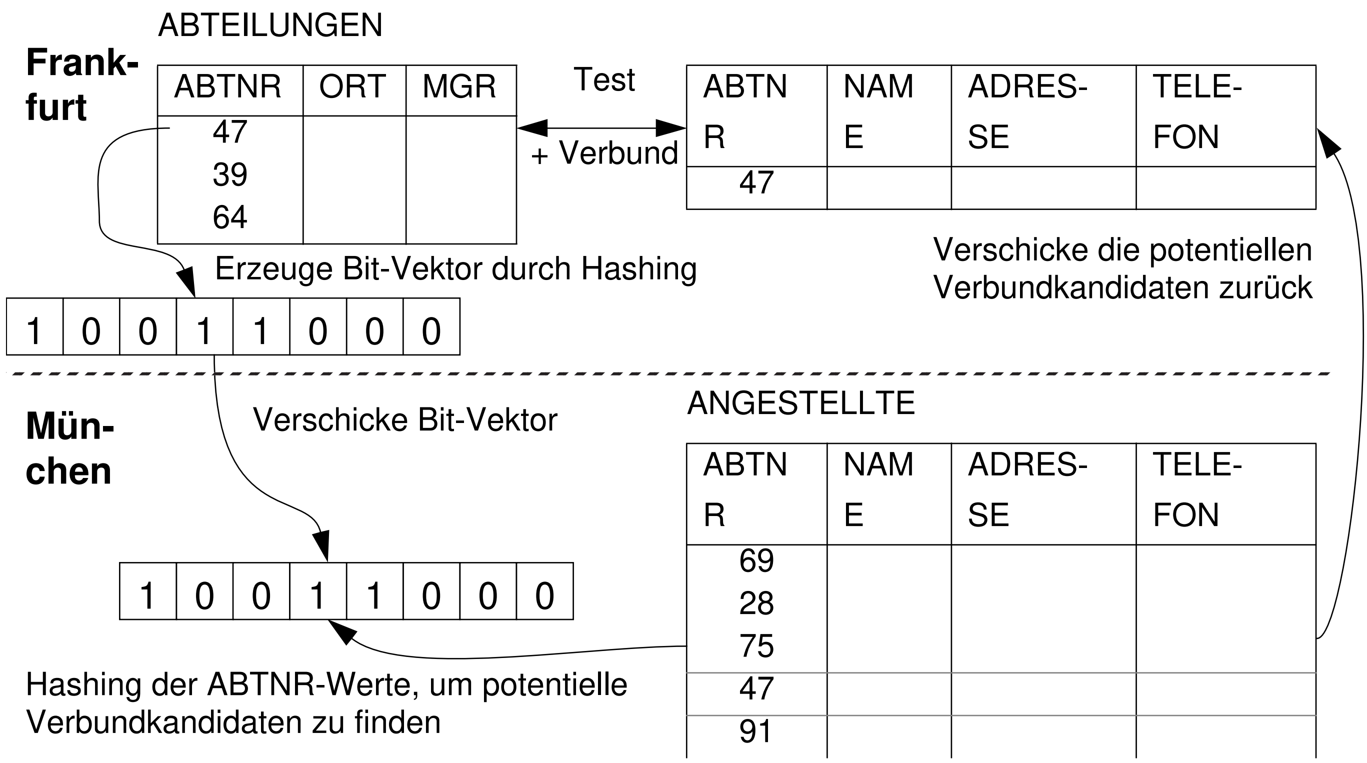
- **Semi-Verbund**
  - Versenden einer Liste der VA von R zum Knoten S
  - Ermitteln der Verbundpartner in S und Zurückschicken zum Knoten von R
  - Durchführung des Verbundes

- **Bit-Vektor-Verbund**
  - ähnlich wie Semi-Verbund, nur Versenden eines durch Hashfunktion erstellten Bitvektors (Bloom-Filter)
  - Rücksenden einer Obermenge der Verbundpartner in S

Semi-Verbund

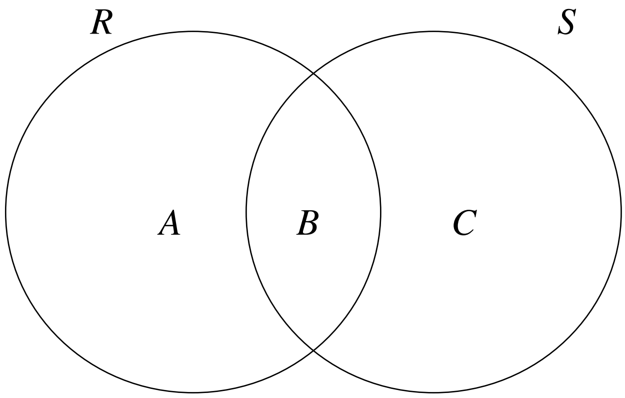


Bit-Vektor-Verbund



Mengenoperationen*

- **Welche Mengenoperationen werden benötigt?**



R, S *vereinigungsverträgliche Eingabeströme*
A, B, C *Elementmengen*

Operations-ergebnis	Übereinstimmung in allen Attributen	Übereinstimmung in einem oder mehreren Attributen
A	Differenz (R–S)	Anti-Semiverbund (S, R)
B	Durchschnitt	Verbund, Semiverbund (S, R)
C	Differenz (S–R)	Anti-Semiverbund (R, S)
A, B		linksseitiger Äußerer Verbund
A, C	Anti-Differenz	Anti-Verbund
B, C		rechtsseitiger Äußerer Verbund
A, B, C	Vereinigung	symmetrischer Äußerer Verbund

- **Welche Algorithmen lassen sich für diese Mengenoperationen heranziehen?**
  - Was muß jeweils verglichen werden?
  - Wie läßt sich eine Verbindung zu den Verbundalgorithmen herstellen?

* Graefe, G.: Query evaluation techniques for large databases. ACM Computing Surv. 25:2. 1993. pp. 73-170

Zusammenfassung

- **Selektionsoperationen**
  - vorhandene Zugriffspfadtypen erfordern zugeschnittene Operationen und effiziente Abbildung
  - Kombination verschiedener Zugriffspfade möglich (TID-Algorithmus)

- **Allgemeine Klassen von Auswertungsverfahren für binäre Operationen**
  - **Schleifeniteration** (nested iteration)
  - **Mischmethode** (merge method)
  - **Hash-Methode** (hashing)

- **Viele Optionen zur Durchführung von Verbundoperationen**
  - Nested-Loop-Verbund
  - Sort-Merge-Verbund
  - Hash-Verbund
  - und Variationen

- **Mengenoperationen**
  - prinzipiell Nutzung der gleichen Verfahrensklassen
  - Variation der Vergleichsdurchführung

- **Erweiterungsinfrastruktur in objekt-relationalen DBS**
  - Einbringen von benutzerdefinierten Funktionen und Operatoren
  - Verallgemeinerung: benutzerdefinierte Tabellenoperatoren mit n Eingabetabellen und m Ausgabetabellen