Join Strategien

Data Warehousing und Data Mining

Mit Materialien von Ulf Leser und Sebastian Wandelt

Patrick Schäfer

Berlin, 23. Oktober 2017

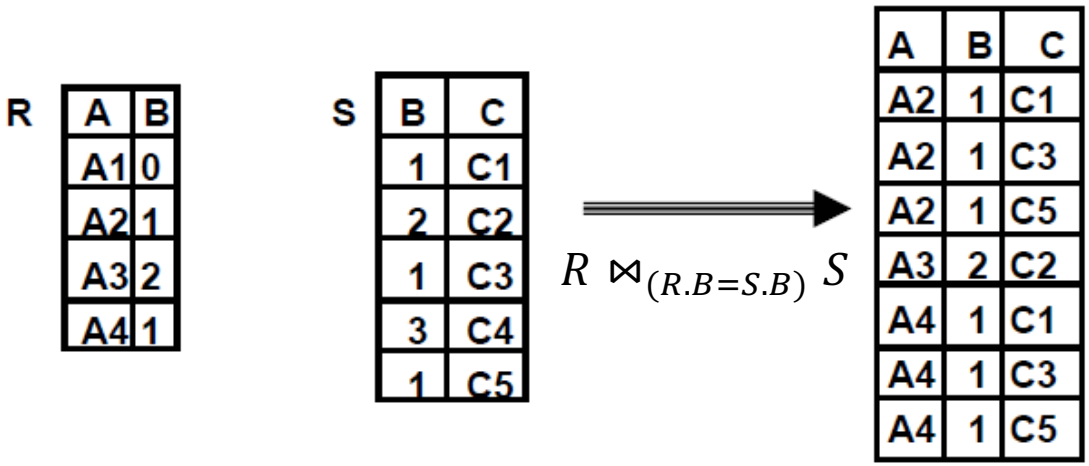
patrick.schaefer@hu-berlin.de

Vorlesung: https://hu.berlin/vl_dwhdm17
Übung: https://hu.berlin/ue_dwhdm17

Join ($R \bowtie S$)

- Wichtiger relationaler Operator: $R \bowtie S$
  - Entspricht kartesischem Produkt + Selektion
  - Kann sehr teuer sein: $O(|R| \cdot |S|)$
  - SQL: Häufig mehrere Joins in einer Anfrage
- Beispiel: Relationen R(A,B) und S(B,C)

```
SELECT * FROM R,S WHERE R.B = S.B
```



Nested Loop Join (Naïve)

- Naive

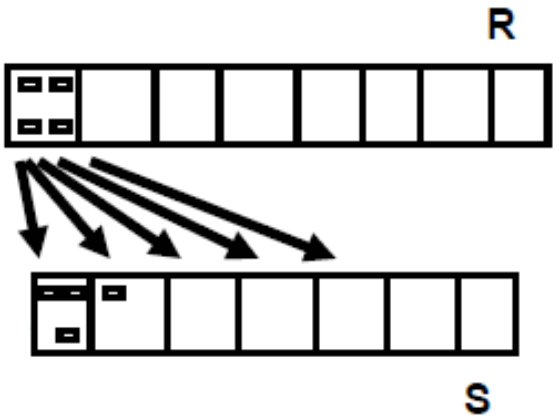
```
FOR EACH r IN R DO
      FOR EACH s IN S DO
            IF (r.B=s.B) THEN OUTPUT (r,s)
```

- Blocked ~

```
FOR EACH block X IN R DO
      FOR EACH block Y IN S DO
            naive-join on X and Y
```

- Indexed ~

```
FOR EACH r IN R DO
     retrieve Y using index on S
     FOR EACH s in Y
          OUTPUT (s,r)
```



- Vorteile/Nachteile?
  - Blocked ~ gut, falls Speicherplatz limitiert ist.
    Aber S wird |R| mal durchlaufen: $O(|R| \cdot |S|)$
  - Für Indexed ~ muss der Index in den Hauptspeicher passen

Begrenzter Hauptspeicher

- Was können wir besser machen, wenn wir nur begrenzt viel Hauptspeicher haben?
  a) Sorted Merge Join
     Idee: Vorsortierung
  b) Hash-Join
     Idee: Hashing

Sorted Merge Join

- Idee ähnlich wie bei MergeSort
  - Sortiere beide Relationen nach dem Join-Attribut
  - Füge sortierte Relationen paarweise zusammen
- 1. Phase: Externes Sortieren jeder Relation
- 2. Phase: Join sortierter Relationen

Phase 1: Externes Sortieren

- Problem: Sortiere R Einträge mit Hauptspeicher für maximal m Einträge

- Idee: Erstelle |R|/m Blöcke:
  - Lese m Einträge aus R in den Hauptspeicher
  - Sortiere Einträge in-memory
  - Schreibe die sortierten Daten auf Platte
  - Füge jeweils zwei Dateien (Blöcke) sortiert zusammen bis nur noch eine Datei bleibt

Beispiel: Extern Sortieren

g	24
a	19
d	31
c	33
b	14
e	16
r	16
d	21

m Einträge von Platte lesen

m Einträge von Platte lesen

Beispiel: Extern Sortieren

g	24
a	19
d	31
c	33
b	14
e	16
r	16
d	21

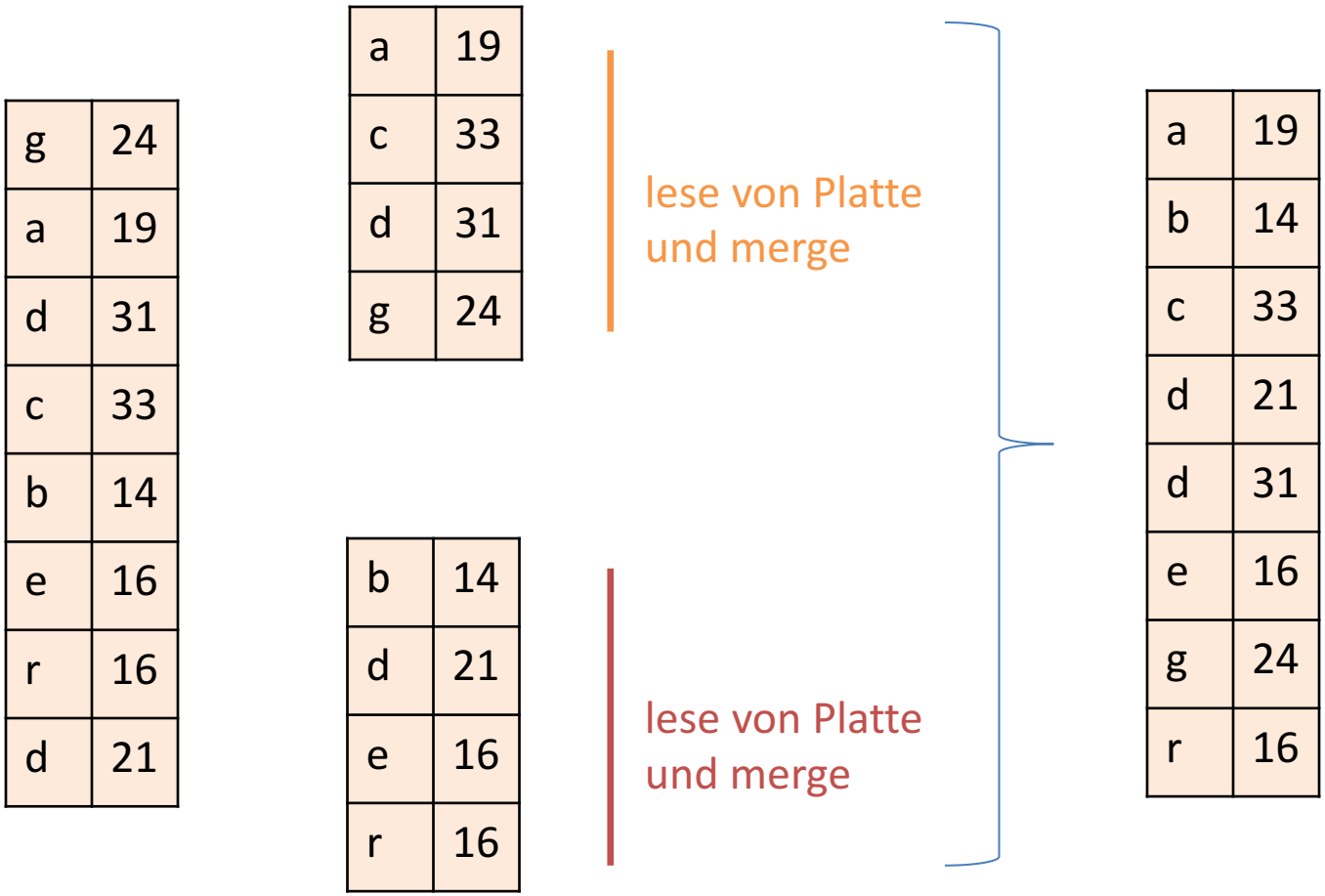
a	19
c	33
d	31
g	24

sortiere in-memory
und schreibe auf Platte

b	14
d	21
e	16
r	16

sortieren in-memory
und schreibe auf Platte

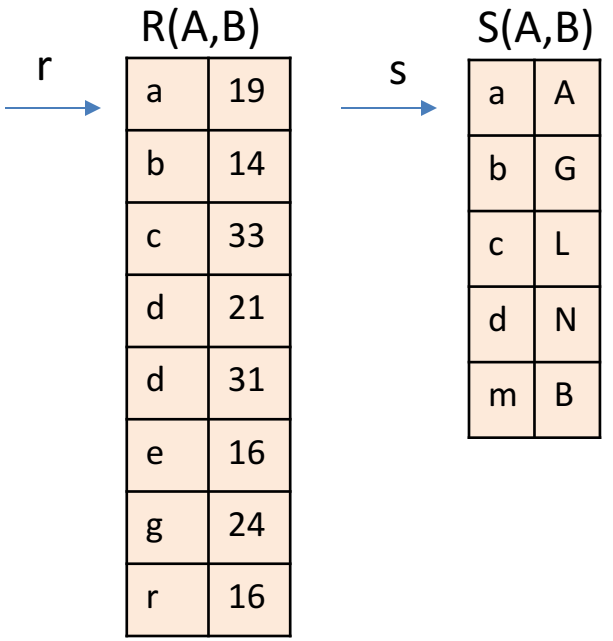
Beispiel: Extern Sortieren



Phase 2: Join sortierter Relationen

- Füge zwei sortierte Relationen R und S zusammen

```
r := first(R); s := first(S);
WHILE hasNext(R) AND hasNext(S) DO
       IF r.A < s.A THEN
         r := next(R);
       ELSEIF r.A > s.A THEN
         s := next (S)
       ELSE /* r.A = s.A */
         key := r.A;
         B := ∅;
         WHILE hasNext(S) AND s.A = key DO
           B := B∪{s};
           s = next(S);
         END DO;
         WHILE hasNext(R) and r.A = key DO
           FOR EACH e in B DO
             OUTPUT(r, e); // (R⋈r.B=keyS)
           END FOR;
           r := next(R);
         END DO;
       ENDIF;
END DO;
```

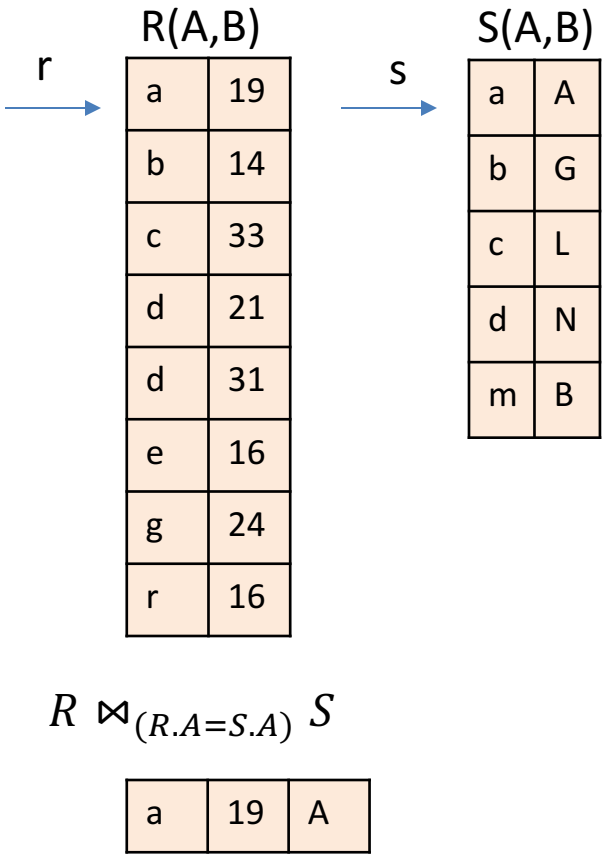


$R \bowtie_{(R.A=S.A)} S$

Phase 2: Join sortierter Relationen

- Füge zwei sortierte Relationen R und S zusammen

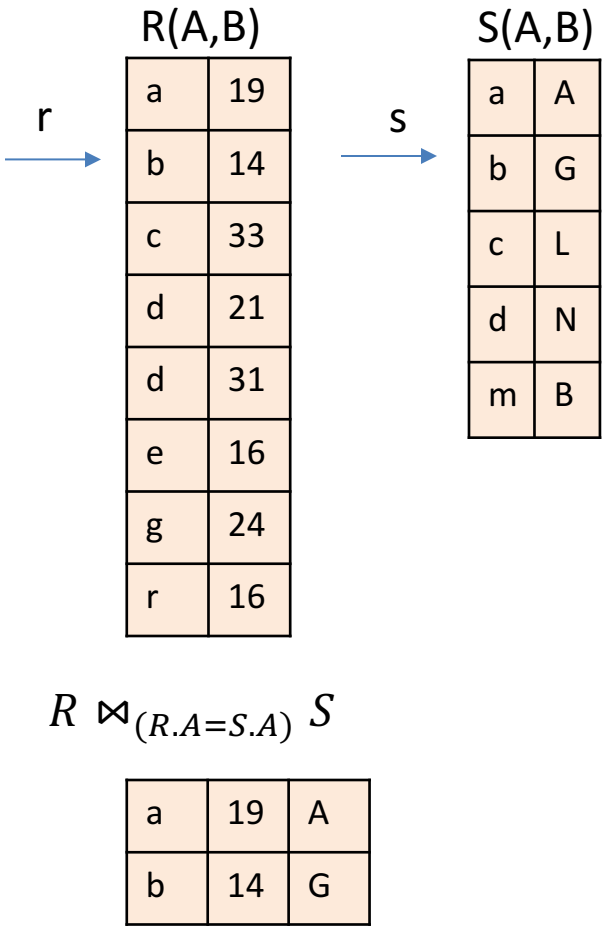
```
r := first(R); s := first(S);
WHILE hasNext(R) AND hasNext(S) DO
       IF r.A < s.A THEN
         r := next(R);
       ELSEIF r.A > s.A THEN
         s := next (S)
       ELSE /* r.A = s.A */
         key := r.A;
         B := ∅;
         WHILE hasNext(S) AND s.A = key DO
           B := B∪{s};
           s = next(S);
         END DO;
         WHILE hasNext(R) and r.A = key DO
           FOR EACH e in B DO
             OUTPUT(r, e); // (R⋈r.B=keyS)
           END FOR;
           r := next(R);
         END DO;
       ENDIF;
END DO;
```



Phase 2: Join sortierter Relationen

- Füge zwei sortierte Relationen R und S zusammen

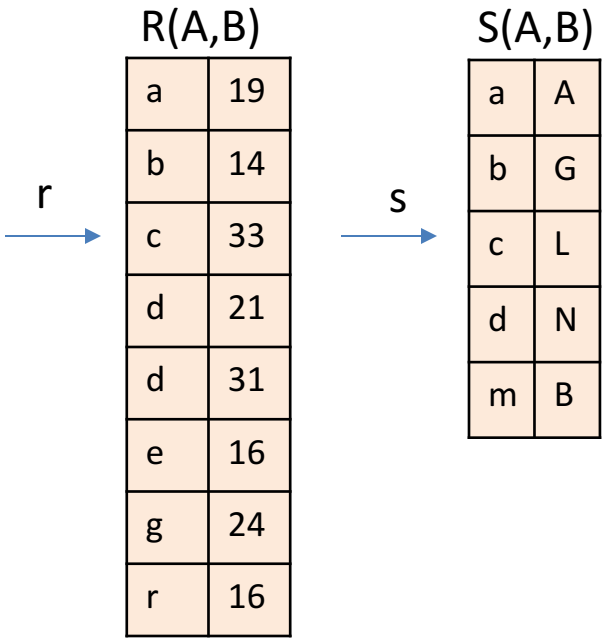
```
r := first(R); s := first(S);
WHILE hasNext(R) AND hasNext(S) DO
       IF r.A < s.A THEN
         r := next(R);
       ELSEIF r.A > s.A THEN
         s := next (S)
       ELSE /* r.A = s.A */
         key := r.A;
         B := ∅;
         WHILE hasNext(S) AND s.A = key DO
           B := B∪{s};
           s = next(S);
         END DO;
         WHILE hasNext(R) and r.A = key DO
           FOR EACH e in B DO
             OUTPUT(r, e); // (R⋈r.B=keyS)
           END FOR;
           r := next(R);
         END DO;
       ENDIF;
END DO;
```



Phase 2: Join sortierter Relationen

- Füge zwei sortierte Relationen R und S zusammen

```
r := first(R); s := first(S);
WHILE hasNext(R) AND hasNext(S) DO
       IF r.A < s.A THEN
         r := next(R);
       ELSEIF r.A > s.A THEN
         s := next (S)
       ELSE /* r.A = s.A */
         key := r.A;
         B := ∅;
         WHILE hasNext(S) AND s.A = key DO
           B := B∪{s};
           s = next(S);
         END DO;
         WHILE hasNext(R) and r.A = key DO
           FOR EACH e in B DO
             OUTPUT(r, e); // (R⋈r.B=keyS)
           END FOR;
           r := next(R);
         END DO;
       ENDIF;
END DO;
```



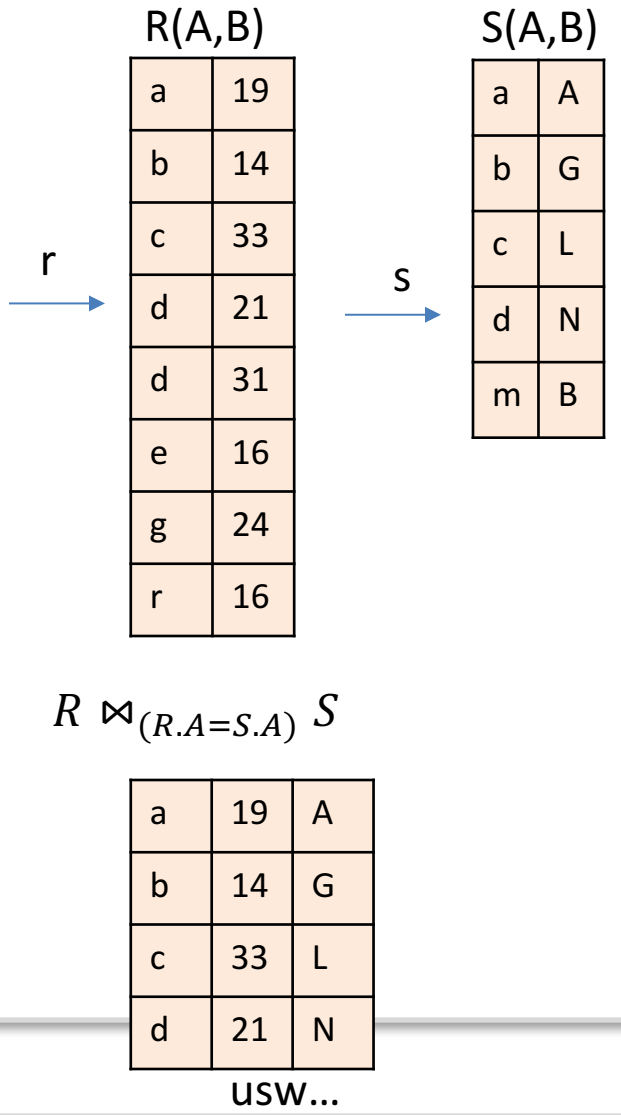
$R \bowtie_{(R.A=S.A)} S$

a	19	A
b	14	G
c	33	L

Phase 2: Join sortierter Relationen

- Füge zwei sortierte Relationen R und S zusammen

```
r := first(R); s := first(S);
WHILE hasNext(R) AND hasNext(S) DO
       IF r.A < s.A THEN
         r := next(R);
       ELSEIF r.A > s.A THEN
         s := next (S)
       ELSE /* r.A = s.A */
         key := r.A;
         B := ∅;
         WHILE hasNext(S) AND s.A = key DO
           B := B∪{s};
           s = next(S);
         END DO;
         WHILE hasNext(R) and r.A = key DO
           FOR EACH e in B DO
             OUTPUT(r, e); // (R⋈r.B=keyS)
           END FOR;
           r := next(R);
         END DO;
       ENDIF;
END DO;
```



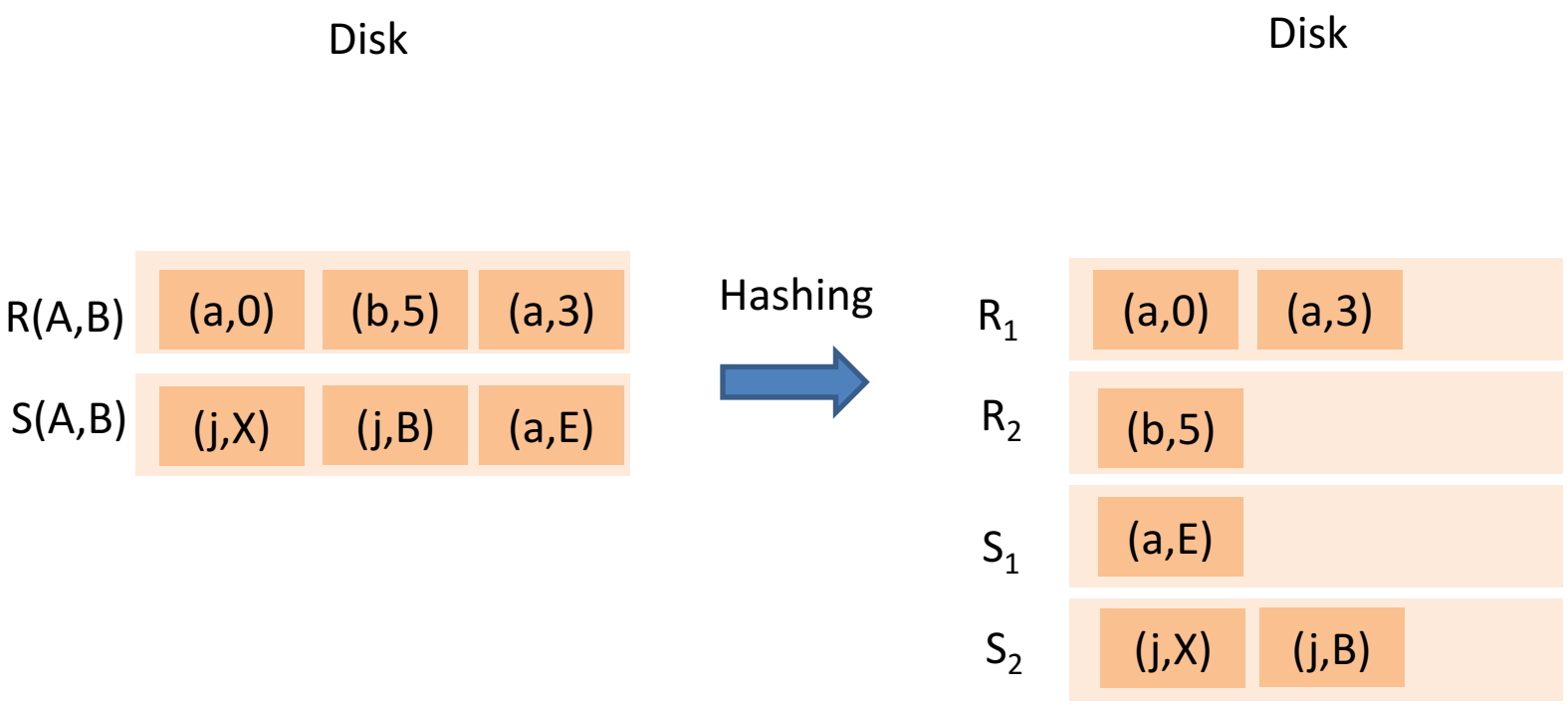
Hash Join

- Idee:
  - Eine Hashfunktion wird verwendet, um die Einträge anhand des Hash-Werts des Join-Attributs zu partitionieren
  - Dann müssen nur Einträge korrespondierender Partitionen verglichen werden

Hash Join

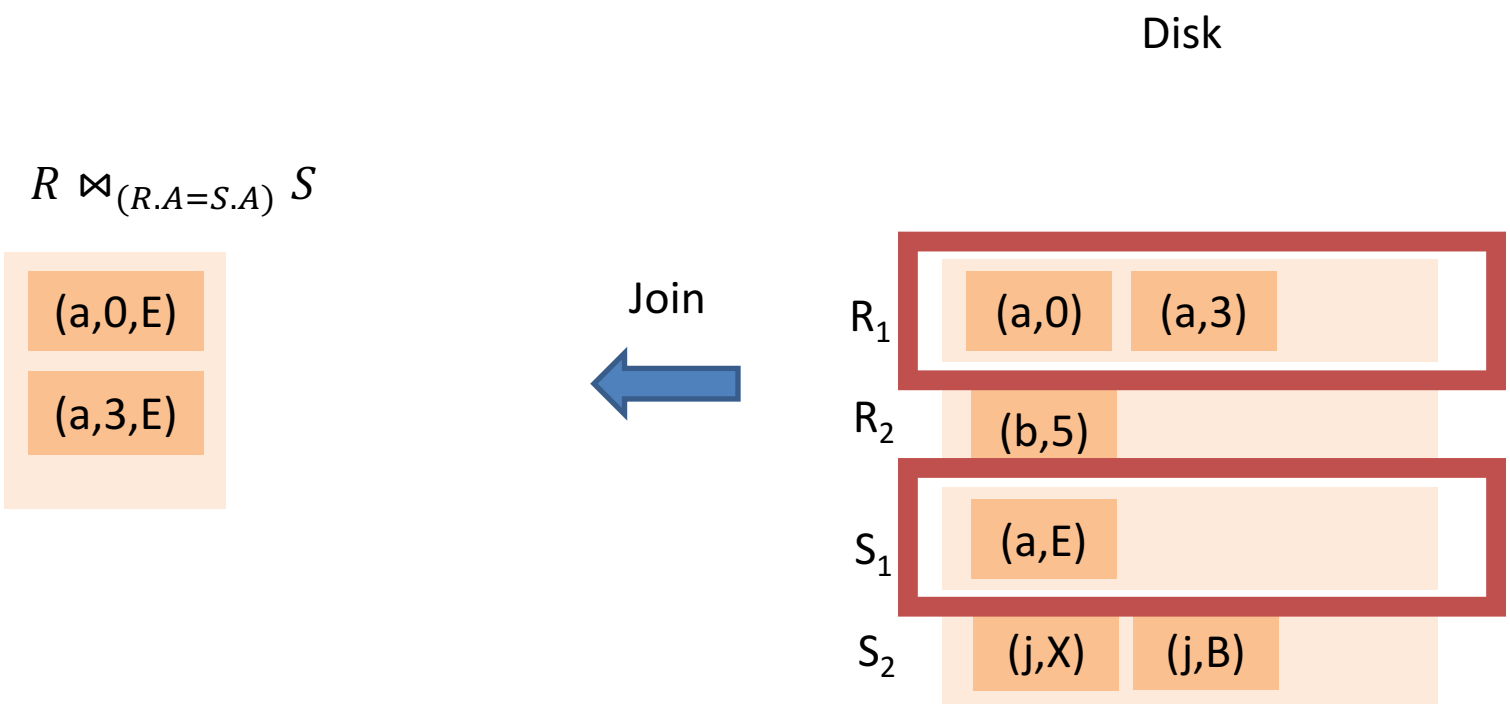
- Benutze die Join-Attribute als Hash-Schlüssel
- Wähle eine Hash-Funktion, die die Elemente möglichst gleichmäßig auf Partitionen (Blöcke) verteilt
- 3 Schritte:
  1. Scanne R, berechne dabei die Hashtabelle, und schreibe alle Blöcke auf die Festplatte
  2. Scanne S, berechne dabei die Hashtabelle, und schreibe alle Blöcke auf die Festplatte
  3. Führe Join über korrespondierende Blöcke schrittweise durch

Beispiel: Hash-Join



1. Verwende Hash-Funktion, um Eingabedaten zu partitionieren

Beispiel: Hash-Join



2. Join korrespondierende Partitionen

Hash Join - Hinweise

- Die erste Hashfunktion sollte die Tupel gleichmäßig auf die Buckets verteilen
- Es sollten nicht zu viele Buckets entstehen, aber zwei sollten in den Hauptspeicher passen
  - Buckets benötigen jeweils ein Filehandle
  - Datei öffnen & schließen ist langsam
- Also: maximale Größe der Buckets abschätzen

Hash Join vs Sorted Merge Join

- Sorted Merge Join:
  - Geeignet bei limitiertem Hauptspeicher
  - Nicht sensitiv bzgl. Datenverteilung (Partitionsgrößen)
  - Vorteilhaft, wenn beide Relationen vorsortiert sind
  - Worst Case: $O(|R| \log |R| + |S| \log |S|)$
- Hash-Join
  - Geeignet bei limitiertem Hauptspeicher
  - Benötigt keine (teure) Vorsortierung
  - Worst Case: $O(|R| \cdot |S|)$