

## B. Funktionale Joins

- ❑ In OO-Datenbanken werden Beziehungen zwischen "Relationen" durch die Abspeicherung von OID umgesetzt.
- ❑ Ein funktionaler Join verknüpft zwei in Beziehung stehende Relationen (z. B. in OQL durch Angabe eines Pfadausdrucks)
- ❑ Beispiel

```

class R_T {
    char[100] RData;
    Set<Ref<S_T>> SRefSet;
    ...
};
create table R of R_T;

class S_T {
    int SAttr;
    char[100] SData;
    ...
};
create table S of S_T;

```

Anfrage in OQL-Syntax:

```

select r.RData, (select s.SAttr from r.SRefSet s)
from R r;

```

Im Gegensatz zu SQL darf ein Attribut des Resultats mengenwertig sein!

396.

### Skizze des Algorithmus

1. Führe Unnest(SRefSet, SRef) auf der Relation R aus.  
Ergebnis: Relation Tmp1
2. Berechne den funktionalen Join von Tmp1 mit der Umsetzungstabelle M.  
Relation Tmp2.
3. Berechne den funktionalen Join von Tmp2 mit der Relation S.  
Ergebnis: Relation Tmp3.
4. Führe Nest(SRef, SRefSet) auf der Relation Tmp3 aus.

### Effiziente Unterstützung des Schritts 2:

- ❑ Partitionierung
  - Bereichspartitionierung der Relation Tmp1 bzgl. des OID.  
Eine Partitionierung von Tmp1 entspricht einem zusammenhängenden Bereich der Umsetzungstabelle M.
- ❑ Sortieren
  - Sortieren der Relation Tmp1 nach OID.
  - Verschmelzen der Relationen Tmp1 und M.

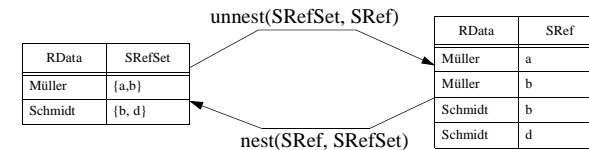
398.

### B.1 Naiver Ansatz:

- ❑ Jede Referenz in R.SRefSet wird zunächst mittels der Übersetzungstabelle in eine physische Adresse umgewandelt. Danach wird mit der physischen Adresse auf das Objekt zugegriffen.
- ❑ Dieser Ansatz hat extrem hohe I/O-Kosten falls die Umsetzungstabelle und die Relation S nicht komplett in den Hauptspeicher passen.

### B.2 Unnest-Algorithmus

- ❑ Durch den Unnest-Operator wird das mengenwertige Attribut SRefSet "flach geklopft".



- ❑ Man beachte hierbei, daß ein mengenwertiges Tupel zu einer physisch **zusammenhängenden** Folge von Tupeln in der flachen Relation transformiert wird.

397.

### Effiziente Unterstützung des Schritts 3:

- ❑ Dieser Schritt kann ähnlich zu Schritt 2 verarbeitet werden.
  - Partitionieren oder Sortieren der Relation Tmp2 bzgl. der physischen Adresse der Objekte.

### Effiziente Unterstützung des Schritts 4:

- ❑ Der Nest-Operator ist sehr ähnlich zum Gruppierungsoperator und kann auch mit den gleichen Techniken (Hashing, Sortieren) ausgeführt werden.

### B.3 Wertebasierter Join

- ❑ Jedes Objekt kennt seinen OID, so daß ein Zugriff auf die Umsetzungstabelle nicht notwendig ist.
- ❑ Skizze des Algorithmus
  1. Führe Unnest(SRefSet, SRef) auf der Relation R aus.  
Ergebnis: Relation Tmp1
  2. Berechne den Join von Tmp1 und der Relation S bzgl. des Prädikats Sref = OID.  
Ergebnis: Relation Tmp2.
  3. Führe Nest(SRef, SRefSet) auf der Relation Tmp2 aus.

399.

## B.4 P(PM)\*M-Algorithmus

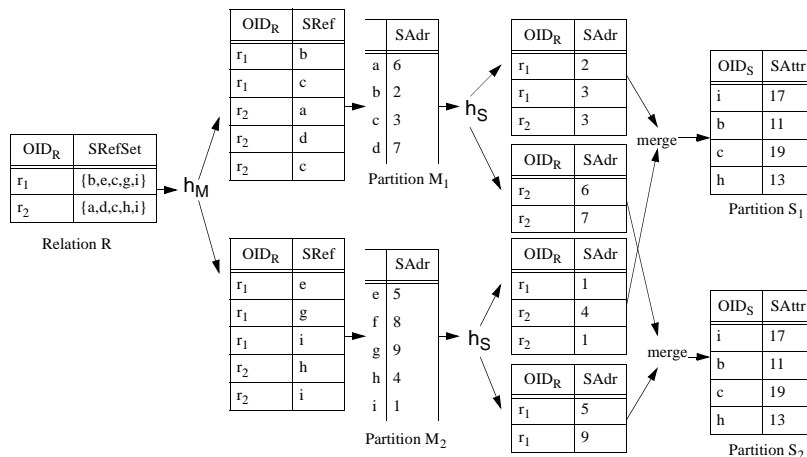
- Ähnlich zum Unnest-Algorithmus mit Partitonierung. Es wird aber nun nach jeder Partitionierung eine **Merge-Phase** durchgeführt.
- Durch die Merge-Phase wird die Gruppierung der Tupel bewahrt.
- Im folgenden beschränken wir uns auf den Spezialfall  $P(PM)^1M$

### Idee:

- Es werden zwei Partitionierungsfunktionen  $h_M$  und  $h_S$  benutzt.
  - Partitionierungsfunktionen unterteilen den Datenraum in zusammenhängende Bereiche. Jeder Bereich gehört zu genau einer Relation.
  - $h_M$  unterteilt die Umsetzungstabelle bzgl. der logischen OID in N Partitionen  $M_1, \dots, M_N$ , die jeweils vollständig in den Hauptspeicher passen.
  - $h_S$  unterteilt die Relation S bzgl. der physischen Adresse (SAdr) in K Partitionen  $S_1, \dots, S_K$ , die jeweils vollständig in den Hauptspeicher passen.

400.

### Beispiel:



402.

## Algorithmus

1. Führe Unnest(SRefSet, SRef) auf der Relation R aus und teile das Ergebnis durch eine Partitionierungsfunktion  $h_M$  bzgl. des Attributs SRef in N Partitionen  $R_1, \dots, R_N$  auf.
2. FOR ( $i=1$ ;  $i < N$ ;  $i++$ )
  - Ersetze in jedem Tupel aus  $R_i$  den logischen OID durch die physische Adresse (SAdr) (Suche in  $M_i$ ).
  - Teile  $R_i$  nun durch eine Partitionierungsfunktion  $h_S$  bzgl. SAdr in K Partitionen  $R_{i,1}, \dots, R_{i,K}$  auf.
3. FOR ( $j=1$ ;  $j < K$ ;  $j++$ )
  - Verschmelze die Partitionen  $R_{1,j}, \dots, R_{N,j}$  in eine Partition, so daß die Ordnung durch den OID der Relation R bestimmt ist. Ergebnis  $Tmp_j$ .
  - Ersetze in jedem Tupel aus  $Tmp_j$  die physische Adresse (SAdr) durch die entsprechende Information aus  $S_j$ .
4. Verschmelze die Relationen  $Tmp_1, \dots, Tmp_K$ , so daß die Ordnung durch den OID der Relation R bestimmt ist und erzeuge die gewünschten Tupel in ihrer geschachtelten Repräsentation.

401.

## Zusammenfassung

- In einem experimentellen Leistungsvergleich wurde gezeigt (Baumandl, Claussen & Kemer, 1998), daß  $P(PM)^*M$  tatsächlich den anderen Verfahren überlegen ist.
  - Vorteil gegenüber den anderen Verfahren ist dabei, daß die Gruppierung der Daten bei der Berechnung des Joins annähernd bewahrt bleibt.
- einige Details
  - Verfahren läßt sich verallgemeinern auf funktionale Joins, die sich auf längere Pfadausdrücke beziehen.
  - Mögliche Aggregate können bereits frühzeitig berechnet werden.
  - Benötigt man im Ergebnis viele Attribute aus R, so kann es effizienter sein, diese Attribute beim Partitionieren erstmal nicht zu berücksichtigen und erst später diese Attribute wieder an das entsprechende Tupel im Ergebnis zu hängen.

403.