s d &m
software design & management

The Paradigm of Relational Indexing

A Survey

Hans-Peter Kriegel, Martin Pfeifle (LMU München),
Marco Pötke (sd&m AG), Thomas Seidl (RWTH Aachen)

BTW 2003
Leipzig, 27.02.2003

sd&m AG
Thomas-Dehler-Straße 27
81737 München
Telefon (089) 6 38 12-0

www.sdm.de

Agenda

- **Benutzerdefinierte Indexstrukturen in ORDBMS**

  Physische Implementierung

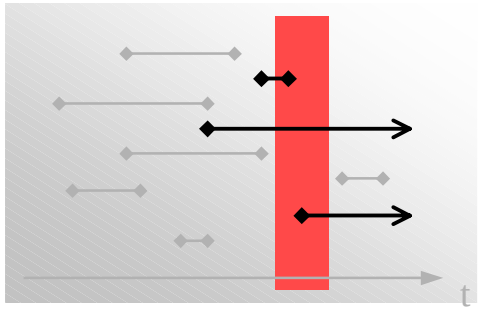
  Speicherschemata für Relationale Indexe

  Operationen auf Relationalen Indexen

  Zusammenfassung

Beispiel: Ausgedehnte Objekte und Anfragen

Intervall-Anfrage



1D-Objekte:

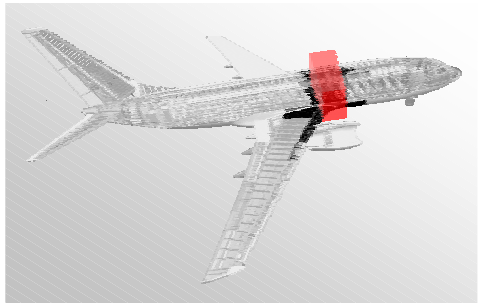
- Temporale Daten
- Unscharfe Messwerte
- Intervall-Constraints
- …

Fenster-Anfrage

2D-Objekte:

- Geographische Daten
- VLSI-Design
- Bitemporale Daten
- …

Box-Anfrage

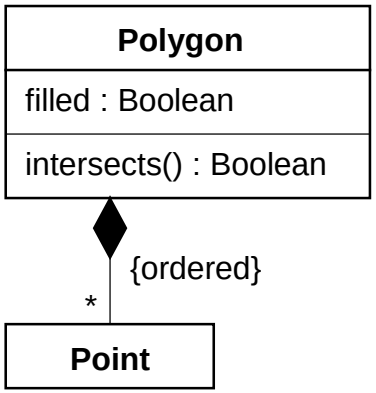


3D-Objekte:

- CAD-Dokumente
- Digitaler Zusammenbau
- Haptische Simulation
- …

Objekt-relationale DDL & DML

Räumliche Datentypen und Prädikate:




```
create type PointList as array of Point;
create type Polygon as object (
   filled Boolean,
   points PointList,
   member function intersects (p Polygon) return Bool
);

create operator intersects (a Polygon, b Polygon) return Bool
   begin return a.intersects(b); end;
```

id	geom
A	**Polygon**(…)
B	**Polygon**(…)
…	…

```
create table my_polygons (
   id Varchar primary key,
   geom Polygon
);
```

Räumliche Anfragebearbeitung:

full table scan

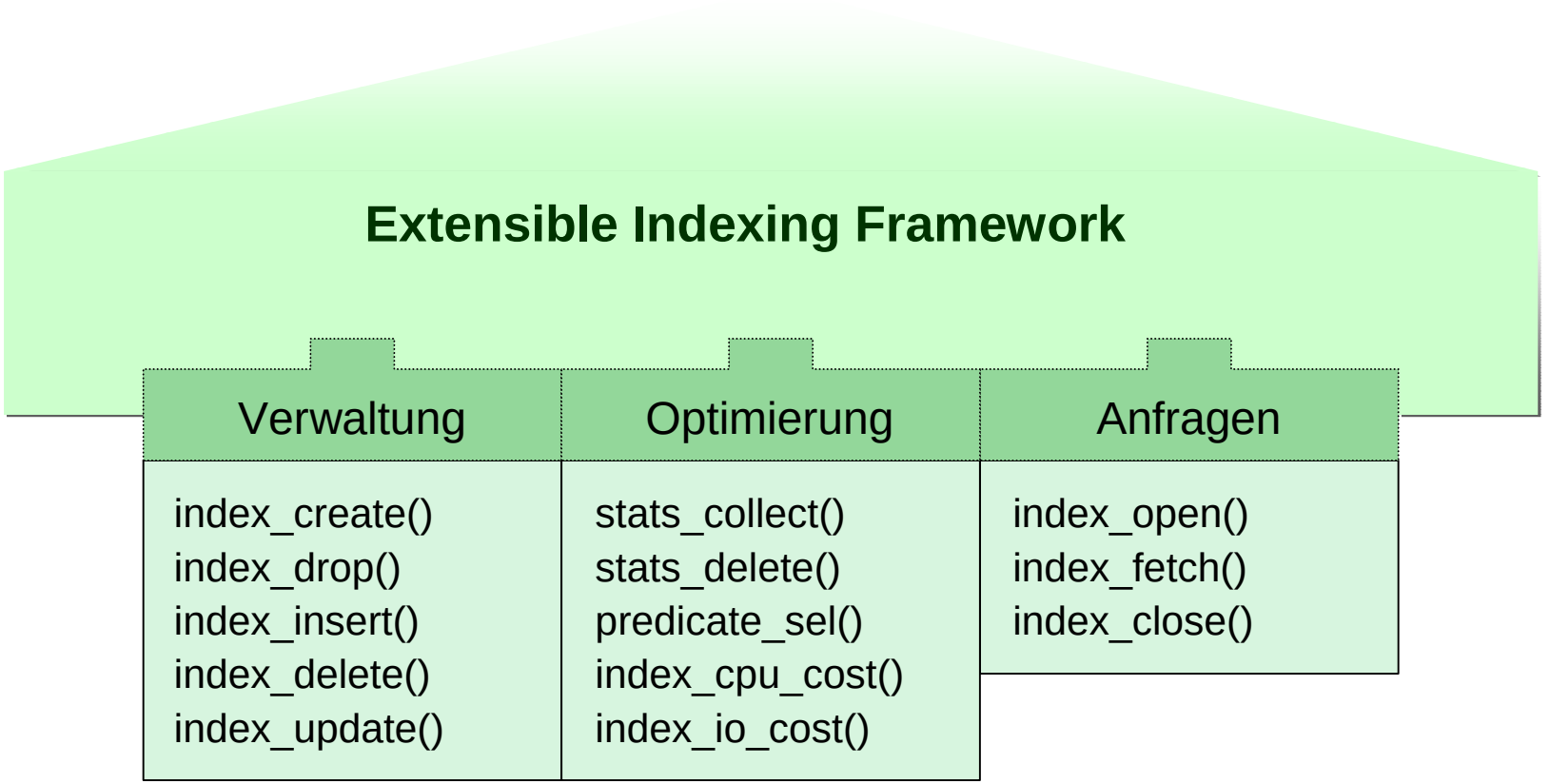
index region scan

?

```
select id
from my_polygons
where intersects(geom,:query) = true;
```

Deklarative Einbettung

```
SQL> create index ...
SQL> insert into ...
SQL> select name from ...
```



Integration von Indexstrukturen

SQL:1999 (bzw. SQL:2003)
Rahmenwerke für objekt-relationale DDL, DML

„Schnittstelle nach oben“

Deklarative Einbettung

Benutzerdefinierte Indexstruktur

Physische Implementierung

„Schnittstelle nach unten“

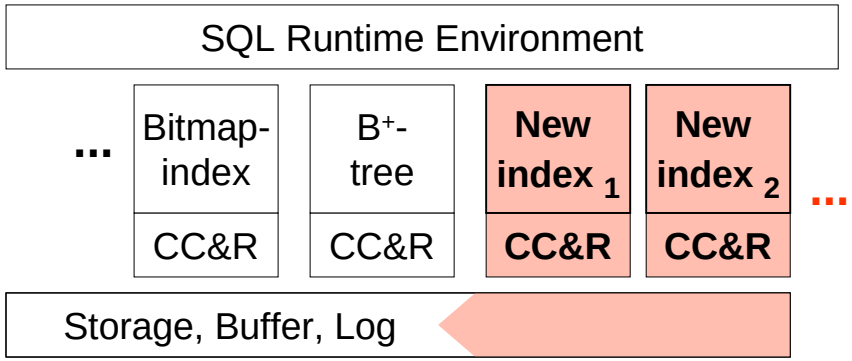
Datenbank-Kern
Block-Manager, Caches, Locking, Logging, …

Agenda

- Benutzerdefinierte Indexstrukturen in ORDBMS
- **Physische Implementierung**
- Speicherschemata für Relationale Indexe
- Operationen auf Relationalen Indexen
- Zusammenfassung

Physische Implementierung (1/3): Nativer Ansatz

Direkte Integration mit Management-Komponenten (CC&R):



Beispiele:

R-Link-tree (Informix) [Inf99]

UB-Tree (TransBase/HC) [RMF+00]

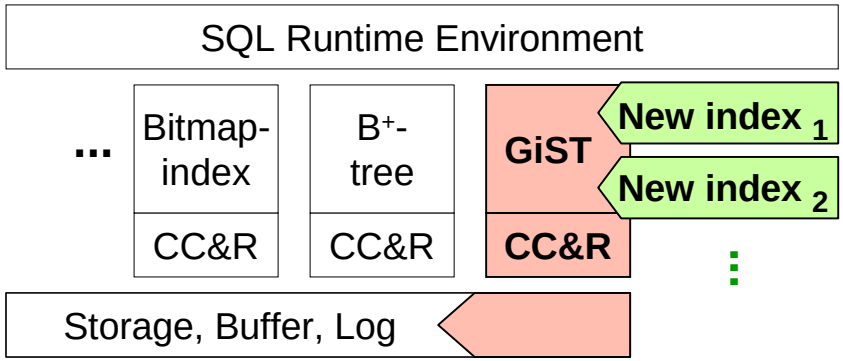
\+ Laufzeitverhalten und Nebenläufigkeit potentiell **optimal**

– **Plattform-abhängig:** Spezifische Erweiterung jedes DB-Kernels

– **Hoher Implementierungs- und Wartungsaufwand**
(Abhilfe: bestehenden Index erweitern, siehe z.B. UB-Tree)

Physische Implementierung (2/3): Generischer Ansatz

Nutzung eines generischen Rahmenwerks mit Index-Basisdiensten:



Beispiel:

GiST - Generalized Search Tree [HNP95]

\+ Laufzeitverhalten und Nebenläufigkeit potentiell **optimal**

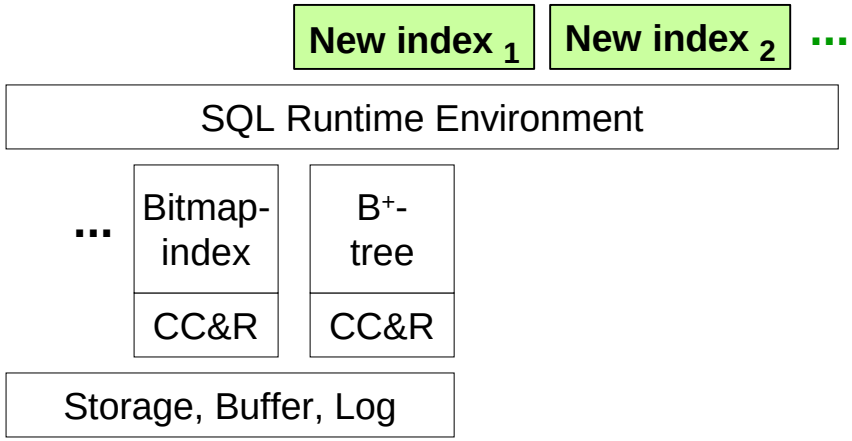
\+ GiST-basierende Indexe im Idealfall **Plattform-unabhängig**

– GiST-Framework selbst aber **spezifisch für jeden DB-Kernel**

– **Hoher Implementierungs- und Wartungsaufwand**

Physische Implementierung (3/3): Relationaler Ansatz

Nutzung von Standard-SQL als „Virtuelle Maschine“:



Beispiele:

Linear Quadtrees (IBM, Oracle)
Relational R-tree (Oracle)
Relational X-tree [BBKM99]
Relational Interval Tree [KPS00]

\+ **Plattform-unabhängig:** Keine Erweiterung des DB-Kernels
\+ **Geringer Implementierungs-/Wartungsaufwand** (SQL-Schnittstelle)
\+ Laufzeitverhalten und Nebenläufigkeit potentiell **sehr gut**
(Erforderlich: Katalog mit zentralen Designkriterien)

Definition: Relationale Indexstruktur

- **Eine *Relationale Indexstruktur* ist**
  - eine Indexstruktur
  - deren persistente Repräsentation ausschließlich in Relationen eines RDBMS abgelegt und verwaltet wird
- **Folgende Relationen treten dabei auf:**
  - *Daten-Tabelle t:* Relation mit den indexierten Anwendungsdaten
  - *Index-Tabellen r_i:* Relationen mit den abgeleiteten Indexdaten
  - *Meta-Tabelle m:* Relation mit den Parametern jedes Indexes

- **Designkriterien:**
  - *Speicherschemata für relationale Indexe*
  - *Operationen auf relationalen Indexen*

  } → Effizienz & Nebenläufigkeit

Agenda

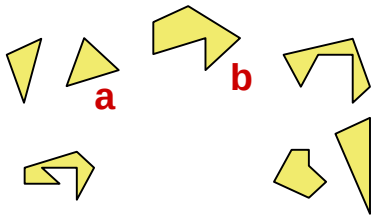
- Benutzerdefinierte Indexstrukturen in ORDBMS
- Physische Implementierung
- **Speicherschemata für Relationale Indexe**
- Operationen auf Relationalen Indexen
- Zusammenfassung

Eigenschaften der relationalen Abstraktion

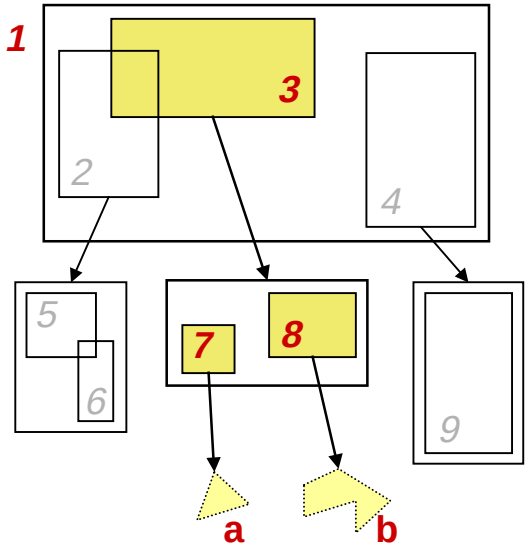
- **Keine dedizierte Zuordnung von relationalen Index-Einträgen zu Plattenblöcken (z.B. Blattknoten)**
  - Kein Block-basiertes Medium, sondern ein „relationales Medium“
  - Indirekte Kontrolle durch Einsatz von Cluster-Mechanismen auf den Index-Tabellen (Primär-B^+-Baum, Cluster-DDL, BLOBs)
- **Änderungen im relationalen Index (z.B. Knoten-Splits) laufen innerhalb von Benutzertransaktionen**
  - Index-Änderungen einer laufenden TA können andere schreibende TA (beliebig) lange blockieren (striktes 2-Phasen-Locking!)
  - Index-Änderungen werden auf der selben Granularität geloggt wie Änderungen an Nutzer-Daten (aufwändigere Rollbacks!)

Navigationales Speicherschema (1/2)

Beispiel:



Relational R-Tree (Oracle)



t

id	geom
a	Polygon(…)
b	Polygon(…)
c	Polygon(…)
d	Polygon(…)
…	…

$\{\tau_1, \ldots, \tau_m\} = t$

r_1

page-id	son-id	son-mbr	…
root	***1***	…	
1	***2***	…	
1	***3***	…	
1	***4***	…	
…	…	…	

ρ = (*root*, ***1***, …)

Definition:

Sei $P = (T, R_1, \ldots, R_n)$ ein relationaler Index auf dem Datenschema T mit den Indexschemata $R_1, \ldots, R_n$.

Dann ist *P navigational* $\Leftrightarrow$

$(\exists\, t \subseteq T)\ (\exists\, r_i \subseteq R_i,\ 1 \le i \le n)$:

mindestens ein $\rho \in r_i$ ist assoziiert mit Tupeln $\{\tau_1, \ldots, \tau_m\} \subseteq t$ und $\boldsymbol{m > 1}$.

Eigenschaften:

+ Clusterung von r_1 über *page-id*
– geringe Nebenläufigkeit (strict 2-PL)
– aufwändiges Logging

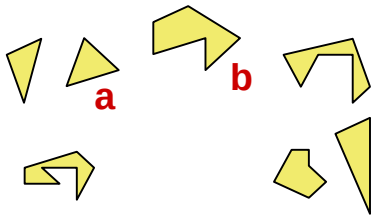
→ **Einbenutzerbetrieb / Read-Only**

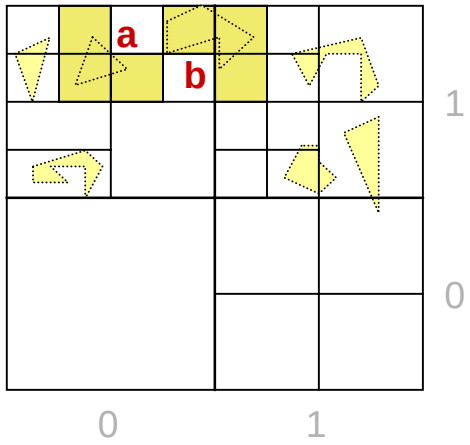
Navigationales Speicherschema (2/2)

- **Weitere Eigenschaften:**
  - Natürliche Abbildung von hierarchischen Indexstrukturen
  - Clusterung der Hierarchieknoten („Seiten”) kann leicht gesteuert werden, z.B. über Sortierung der page-id im Relational R-tree:
    - Breitendurchlauf = Sibling-Clustering [KC 98]
    - Tiefendurchlauf = Hierarchisches Clustering, „positive Pruning“
  - Variabler Verzweigungsgrad möglich („Supernode“-Konzept), z.B. Relational R-tree: kein Split ohne Überlappung
- **Beispiele:**
  - Relational R-tree (Oracle)
  - Relational X-tree [BBKM99]

Direktes Speicherschema (1/2)

Beispiel: Linear Quadtree (IBM, Oracle)





t

id	geom
a	Polygon(…)
b	Polygon(…)
c	Polygon(…)
d	Polygon(…)
…	…

τ = (**a**, Polygon(…))

r_1

code	geom-id
24	**a**
25	**a**
28	**a**
31	**b**
…	…

ρ = (24, **a**)

Definition:

Sei $P = (T, R_1, \ldots, R_n)$ ein relationaler Index auf dem Datenschema T mit den Indexschemata $R_1, \ldots, R_n$.

Dann ist *P direkt* ⇔

$(\forall\ t \subseteq T)\ (\forall\ r_i \subseteq R_i, 1 \leq i \leq n)$:

jedes $\rho \in r_i$ ist assoziiert mit **genau einem** $\tau \in t$.

Eigenschaften:

+ Clusterung von r_1 über *code*
+ hohe Nebenläufigkeit möglich
+ kein Logging-Overhead

→ **Schreibender Mehrbenutzerbetrieb**

Direktes Speicherschema (2/2)

- **Beispiele:**
  - Linear Quadtree (IBM, Oracle)
  - Relational Interval Tree [KPS 00]

Agenda

- Benutzerdefinierte Indexstrukturen in ORDBMS
- Physische Implementierung
- Speicherschemata für Relationale Indexe
- **Operationen auf Relationalen Indexen**
- Zusammenfassung

Definition: Cursor-getriebene Operation

- **Eine Cursor-getriebene Operation ist**
  - eine Operation (*select*, *insert*, *update*, *delete*),
  - die *höchstens eine* Anfrage (Cursor) auf den Indextabellen absetzt.
  - Bei *select*: Der Cursor liefert bereits das Ergebnis der Operation (keine prozedurale Nachbearbeitung).

- **Vorteile:**
  - Anzahl von verwendeten Cursors wird minimiert
  - Anfrageoptimierung des Datenbanksystems wird ausgenutzt
  - Direkte Integration in größere Ausführungspläne
  - Robuster und fehlerfreier Code über deklaratives SQL

  → **Maximale Delegation an das Datenbanksystem**

Cursor-getriebene Operation: Anfrage auf R-tree

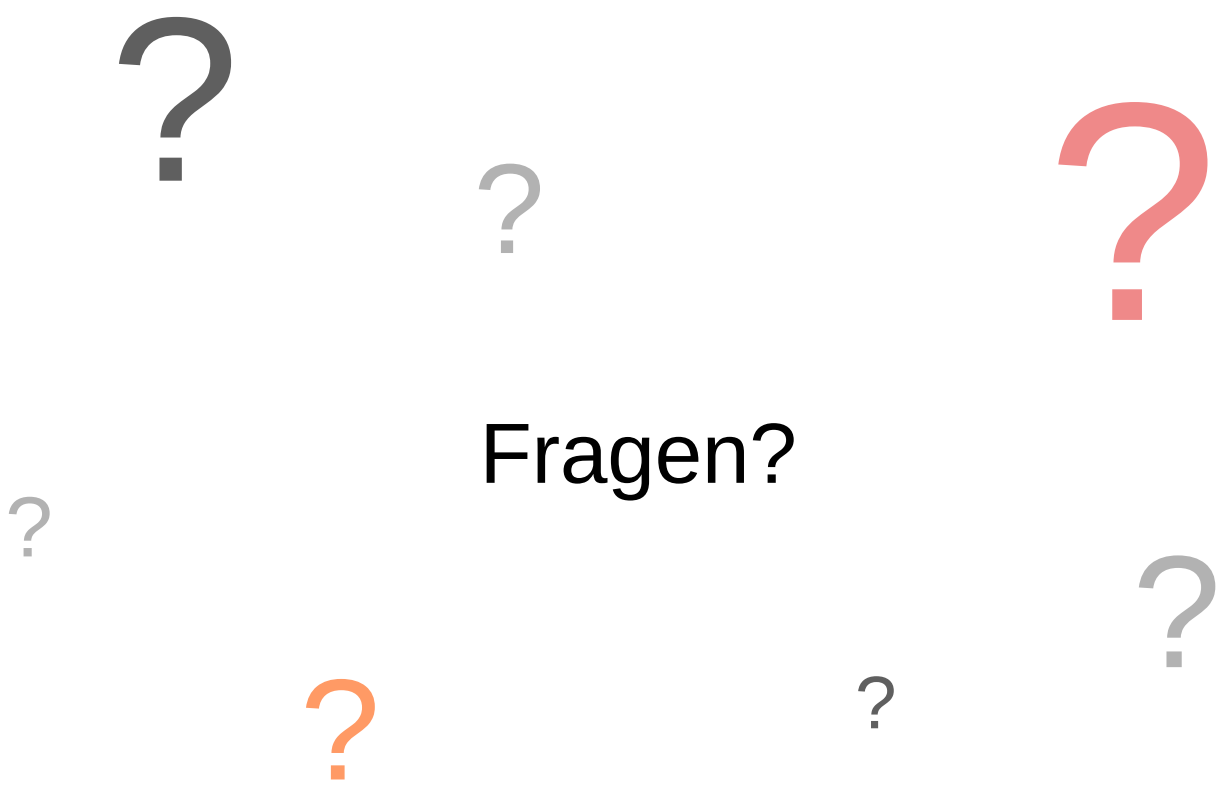
```
WITH RECURSIVE tree_traversal (son_id, son_mbr, page_lev) AS (
      SELECT son_id, son_mbr, page_lev
      FROM rtree_index_table
      WHERE page_id = ROOT
      UNION ALL
      SELECT next.son_id, next.son_mbr, next.page_lev
      FROM tree_traversal prior, rtree_index_table next
      WHERE prior.son_mbr INTERSECTS BOX((0,0),(100,100))
      AND prior.son_id = next.page_id
)                                                    // deklarativer Baumdurchlauf
SELECT son_id AS id
FROM tree_traversal
WHERE page_lev = 0;                                  // Selektion der Datenobjekte
```

Agenda

- Benutzerdefinierte Indexstrukturen in ORDBMS
- Physische Implementierung
- Speicherschemata für Relationale Indexe
- Operationen auf Relationalen Indexen
- **Zusammenfassung**

Zusammenfassung

- **Relationale Indexstrukturen haben Potential:**
  - Nutzen bestehende Datenbank-Dienste aus
  - Sehr hohe Effizienz und Nebenläufigkeit machbar
  - Erwecken Extensible Indexing Frameworks zum Leben
- **Aber Vorsicht:**
  - Anforderungen der Anwendung müssen berücksichtigt werden
  - ORDBMS darf nicht zum heimlichen Application Server werden



Fragen?