



10 Mehrdimensionale Zugriffspfade



Allgemeines Ziel physischer Zugriffspfade

- möglichst effizienter Zugriff auf Datensätze, die sich für eine Anfrage qualifizieren
- Qualifizierung erfolgt über ein Attribut (1-dimensionaler Zugriffspfad) oder über mehrere Attribute gleichzeitig (mehrdimensionaler Zugriffspfad)

Beispiele für 1-dimensionale Zugriffspfade

- B/B*-Baum
- Hashing

Beispiele für mehrdimensionale Zugriffspfade

- mehrdimensionales Hashing
- MDB-/KdB-Bäume
- Grid Files
- R-/R*-Bäume



Motivation

- Selektion der Sätze durch Angabe von Schlüsselwerten für mehrere Felder (z.B. Name und Wohnort)
- Wichtigstes Beispiel: räumliche Koordinaten (x, y) oder (x, y, z)

Anforderungen

- Organisation räumlicher Daten
- Erhaltung der Topologie (Cluster-Bildung)
- raumbezogener Zugriff

Definitionen

- Zu einem Satztyp $R = (A_1, \dots, A_n)$ gebe es N Sätze, von denen jeder ein n -Tupel $t = (a_1, \dots, a_n)$ von Werten ist. Die Attribute A_1, \dots, A_k ($k \leq n$) seien Schlüssel
- Eine Anfrage Q spezifiziert die Bedingungen, die von den Schlüsselwerten der Sätze in der Treffermenge erfüllt sein müssen



❖ ABGRENZUNG

- ◆ Nutzung von eindimensional Indexstrukturen -> Mischen von RowIDs
- ◆ Linearisierung multidimensionaler Räume
- ◆ Indexierung von Summendaten

❖ MULTIDIMENSIONALER STRUKTUREN

- ◆ Anfragetypen
- ◆ MDB-Bäume
- ◆ KdB-Bäume
- ◆ Grid-File
- ◆ Multidimensionales Hashing

❖ RAUMZUGRIFFSSTRUKTUREN

- ◆ Anfragetypen
- ◆ Punkttransformation
- ◆ Clipping
- ◆ Überlappende Regionen
- ◆ R-/R⁺-Baum

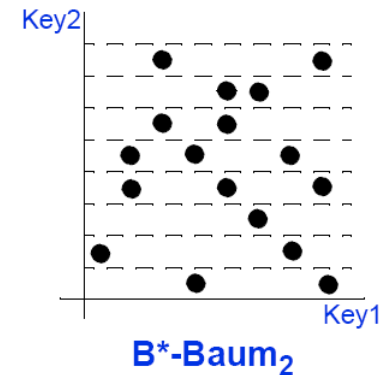
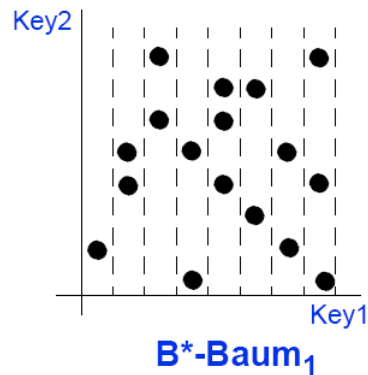


Abgrenzung



Bisher: Nutzung eindimensionaler Indexe (non-composite index)

- Indexierung einer Dimension (z.B. über B*-Baum)
- Beispiel
 - 2-dimensionaler Suchraum
 - Zerlegungsprinzip über Key1 oder Key2



- zwei B*-Bäume notwendig
- Zugriff nach Key₁ --> B*-Baum₁ ... nach Key₂ -> B*-Baum₂
- ... aber: Zugriff nach (Key₁ AND Key₂) ???



Zugriff nach (Key_1 AND Key_2) oder (Key_1 OR Key_2)

- Zeigerliste für Werte von Key_1
- Zeigerliste für Werte von Key_2
- Mischen und Zugriff auf Ergebnistupel

Simulation des mehrdimensionalen Zugriffs mit einem B^ -Baum*

- Konkatenierte Schlüsselwerte (composite index)
- Unterstützung für Suchoperationen
 - (Key_1 AND Key_2) -> OK!
 - (Key_1) -> OK bei B_1
 - (Key_2) -> OK bei B_2
 - ...OR-Verknüpfung???
 - ...Skalierung bei n Dimensionen

B^* -Baum B_1

Key ₁	Key ₂
A ₁	B ₁
A ₁	B ₂
A ₁	B _m
A ₂	B ₁
A ₂	B _m
...	...
A _n	B ₁
A _n	B _m

B^* -Baum B_2

Key ₂	Key ₁
B ₁	A ₁
B ₁	A ₂
B ₁	A _n
B ₂	A ₁
B ₂	A _n
...	...
B _m	A ₁
B _m	A _n



TPC-H Relation Customer mit Indizes

```
CREATE INDEX cust_nationkey ON customer(c_nationkey);  
CREATE INDEX cust_mktsegment ON customer(c_mktsegment);
```

SELECT-Anweisung (würde auch ohne Hint funktionieren!)

```
SELECT /*+AND_EQUAL(t cust_mktsegment cust_nationkey)*/ *  
FROM CUSTOMER t  
WHERE C_NATIONKEY = 7  
AND C_MKTSEGMENT = 'AUTOMOBILE';
```

Query-Plan

```
SELECT STATEMENT Optimizer Mode=CHOOSE  
  TABLE ACCESS BY INDEX ROWID TPCH.CUSTOMER  
    AND-EQUAL  
      INDEX RANGE SCAN TPCH.CUST_MKTSEGMENT  
      INDEX RANGE SCAN TPCH.CUST_NATIONKEY
```

AND-EQUAL Operation

- Mischen von ROWIDs gleichartiger Indexe
--> Indexkonvertierung !

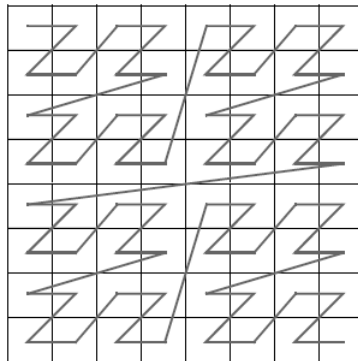


Idee

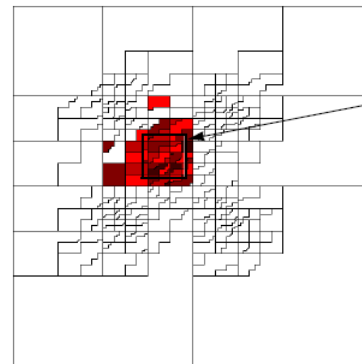
- Linearisierung eines multidimensionalen Raumes und Nutzung eindimensionaler Indexstrukturen

Beispiel: UB-Baum

- Linearisierung durch raumfüllende Z-Kurve; möglichst nachbarschaftserhaltend



a) Linearisierung über Z-Kurve



Von der Anfrage
adressierter
Bereich des
Datenraums

b) Zugriffe bei Bereichsanfragen

- ... beim Zugriff wird “etwas” mehr als nötig gelesen!



Multidimensionale Strukturen



exact-match-Anfrage

- Alle Schlüsselattribute sind in der Anfrage spezifiziert
- $Q = (A_1 = a_1) \wedge (A_2 = a_2) \wedge \dots \wedge (A_k = a_k)$

partial-match-Anfrage

- Nur einige der Schlüsselattribute sind spezifiziert ($s < k$)
- z.B. Abteilung = "K55" \wedge Ort = "Dresden"

range-Anfrage

- Für einige Attribute ist ein Werteintervall angegeben
- z.B. Abteilung \leq "K10" \wedge Abteilung \leq "K60"

Merke

- Das Problem ist mit herkömmlichen Zugriffspfadmethoden, wie z.B. B/B*-Baum, Hashing nicht effizient zu lösen



Multidimensionale B-Bäume (MDB-Bäume)

- Ziel und Idee
 - Speicherung multidimensionaler Schlüssel (a_k, \dots, a_1) in einer Indexstruktur
 - Hierarchie von Bäumen, wobei jede Hierarchiestufe jeweils einem Attribut entspricht.
- Prinzip
 - k-stufige Hierarchie von B-Bäumen.
 - Jede Hierarchiestufe (**Level**) entspricht einem Attribut.
Werte des Attributs a_i werden in Level i ($1 \leq i \leq k$) gespeichert.
 - Die B-Bäume des Levels i haben die Ordnung m_i ;
 m_i hängt von der Länge der Werte des i -ten Attributs ab.

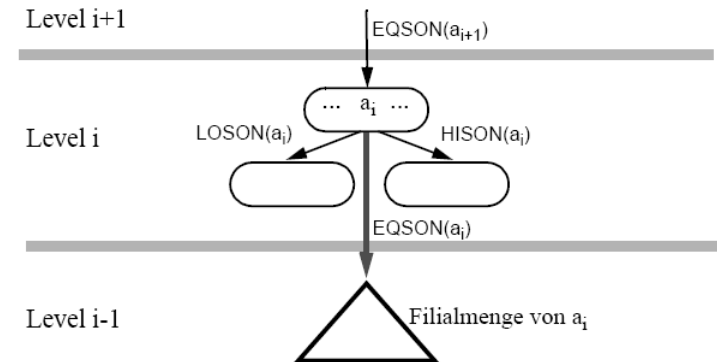
Verzeigerung in einem B-Baum

- linker Teilbaum bzgl. a_i : **LOSON**(a_i)
- rechter Teilbaum bzgl. A_i : **HISON**(a_i)



Verknüpfung der Level

- Jeder Attributwert a_i hat zusätzlich einen EQSON-Zeiger: **EQSON(a_i)** zeigt auf den B-Baum des Levels $i-1$, der die verschiedenen Werte abspeichert, die zu Schlüsseln mit Attributwert a_i gehören (Filialmenge)



Filialmenge

- M sei Menge aller Schlüssel (a_k, \dots, a_1) mit dem Präfix (a_k, \dots, a_i) $i < k$
- Die Menge der $(i-1)$ -dimensionalen Schlüssel, die man aus M durch Weglassen des gemeinsamen Präfixes erhält, heißt Filialmenge von (a_k, \dots, a_i) (oder kurz: Filialmenge von a_i).
- $\text{EQSON}(a_i)$ zeigt auf einen B-Baum, der die Filialmenge von a_i abspeichert

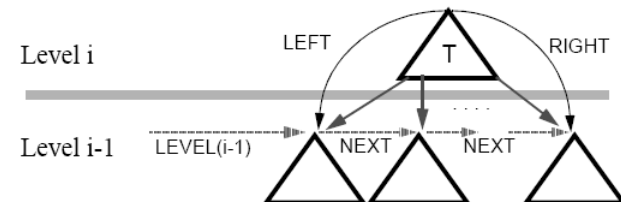


Unterstützung von Anfragetypen

- Exact Match Queries:
 - > können effizient beantwortet werden
- Range, Partial Match und Partial Range Queries:
 - > erfordern zusätzlich sequentielle Verarbeitung auf jedem Level
- Unterstützung von Range Queries:
 - > Verkettung der Wurzeln der B-Bäume eines Levels: NEXT-Zeiger

Unterstützung von Partial Match Queries und Partial Range Queries

- Einstiegszeiger für jedes Level
 - LEVEL(i) zeigt für Level i auf den Beginn der verketteten Liste aus NEXT-Zeigern
- Überspringzeiger von jeder Level-Wurzel eines Level-Baumes T zum folgenden Level:
 - LEFT(T) zeigt auf die Filialmenge des kleinsten Schlüssels von T
 - RIGHT(T) zeigt auf die Filialmenge des größten Schlüssels von T

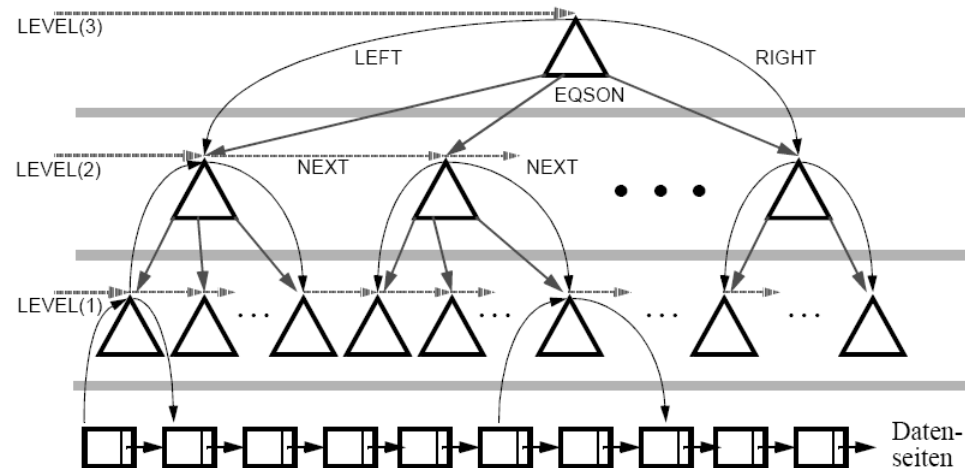




Annahme:

- Für jedes Attribut gilt:
 - Gleichverteilung der Werte im zugehörigen Wertebereich
 - Stochastische Unabhängigkeit der Attribute
- Alle Bäume auf demselben Level haben dieselbe Höhe
- Maximale Höhe: $O(\log N + k)$

Beispiel: MDB-Baum



Einwand

- Die Annahmen sind in realen Dateien äußerst selten erfüllt
- Gilt obige Annahme nicht dann beträgt die maximale Höhe: $O(k \cdot \log N)$

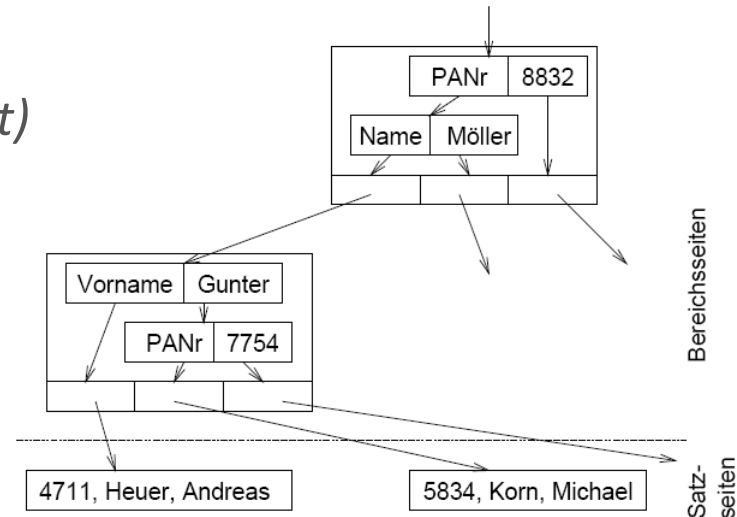


Idee

- B*-Baum, bei dem Indexseiten als binäre Bäume mit Zugriffsattributen, Zugriffsattributwerten und Zeigern realisiert werden
- Unterstützung von Primär- und Sekundärschlüssel
--> keine zusätzlichen Sekundärindexe

Struktur eines KdB-Baumes vom Typ (b, t)

- pro Indexseite: Teilbaum, der nach mehreren Attributen hintereinander verzweigt
 - innere Knoten (Bereichsseiten) enthalten einen kd-Baum mit maximal b internen Knoten
 - Blätter (Satzseiten) enthalten bis zu t Tupel





Bereichsseiten (innere Knoten)

- Anzahl der Schnitt- und Adressenelemente der Seite
- Zeiger auf Wurzel des in der Seite enthaltenen kd-Baumes
- Schnitt- und Adressenelemente

Schnittelement

- Zugriffsattribut
- Zugriffsattributwert
- zwei Zeiger auf Nachfolgerknoten des kd-Baumes dieser Seite (können Schnitt- oder Adressenelemente sein)
- Adressenelemente
- Adresse eines Nachfolgers der Bereichsseite im KdB-Baum (Bereichs- oder Satzseite)



Komplexität: lookup, insert und delete

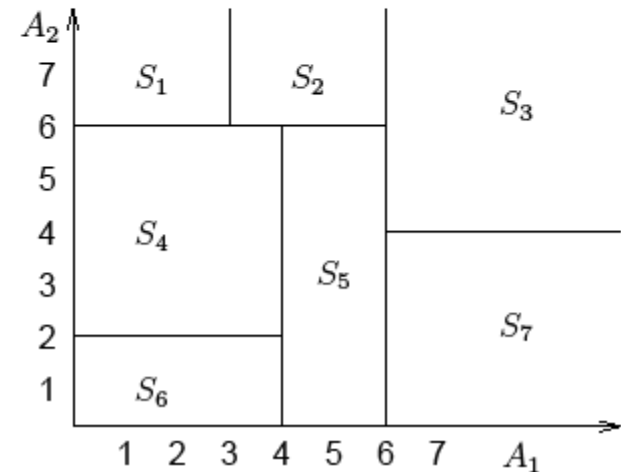
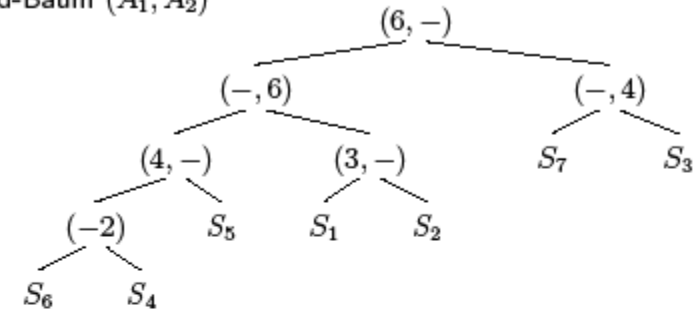
- exact-match: $O(\log n)$
- partial-match: besser als $O(n)$
- bei t von k angegebenen Attributen in der Anfrage: $O(n^{1-t/k})$

Trennattribute

- zyklische Festlegung
- Einbeziehung der Selektivitäten
-> Zugriffsattribut mit hoher Selektivität sollte früher und häufiger als Schnittelement eingesetzt werden

Brickwall (2d-Baum)

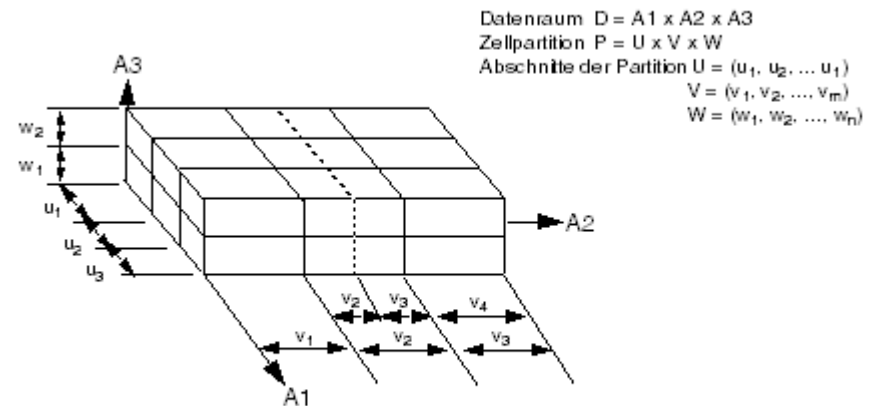
2d-Baum (A_1, A_2)





Prinzip der Organisation des umgebenden Datenraums durch Dimensionsverfeinerung

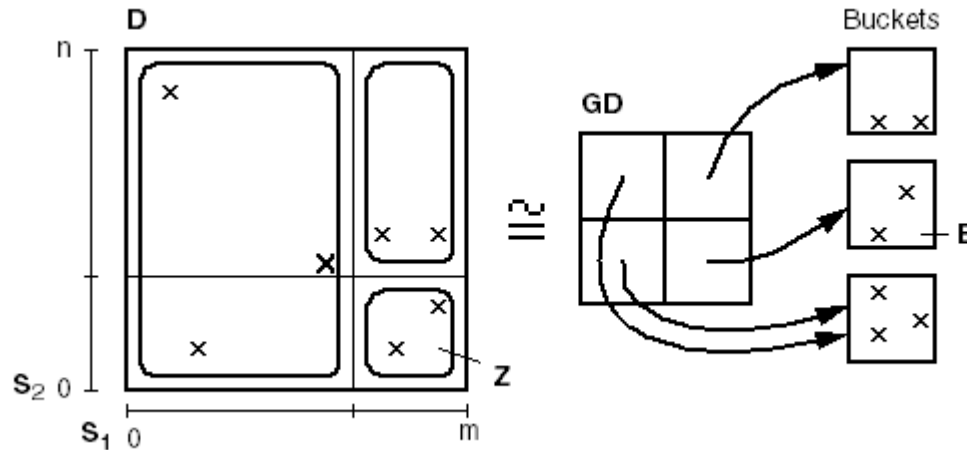
- Datenraum D wird dynamisch durch **ein orthogonales Raster** (grid) partitioniert, so dass k-dimensionale Zellen (Grid-Blöcke) entstehen
- die in den Zellen enthaltenen Objekte werden Buckets zugeordnet
- es muss eine eindeutige Abbildung der Zellen zu den Buckets gefunden werden
- die klassenbildende Eigenschaft dieser Verfahren ist das Prinzip der Dimensionsverfeinerung, bei dem ein Abschnitt in der ausgewählten Dimension durch einen vollständigen Schnitt durch D verfeinert wird



Dreidimensionaler Datenraum D mit Zellpartition P;
Veranschaulichung eines Split-Vorganges im Intervall v_2



Zerlegungsprinzip durch Dimensionsverfeinerung



Komponenten

- k Skalierungsvektoren (Scales) definieren die Zellen (Grid) auf k -dim. Datenraum D
- Zell- oder Grid-Directory GD : dynamische k -dim. Matrix zur Abbildung von D auf die Menge der Buckets
- Bucket: Speicherung der Objekte einer oder mehrerer Zellen (Bucketbereich BB)



Eigenschaften

- 1:1-Beziehung zwischen Zelle Z_i und Element von GD
- Element von Grid-Directory ist ein Zeiger auf Bucket B
- n:1-Beziehung zwischen Z_i und B (mehrere Zellen können auf das gleiche Bucket zeigen)

Ziele

- Erhaltung der Topologie
- effiziente Unterstützung aller Fragetypen
- vernünftige Speicherplatzbelegung

Anforderungen

- Prinzip der zwei Plattenzugriffe unabhängig von Werteverteilungen, Operationshäufigkeiten und Anzahl der gespeicherten Sätze
- Split- und Mischoperationen jeweils nur auf zwei Buckets
- Speicherplatzbelegung
- durchschnittliche Belegung der Buckets nicht beliebig klein
- schiefe Verteilungen vergrößern nur Grid-Directory

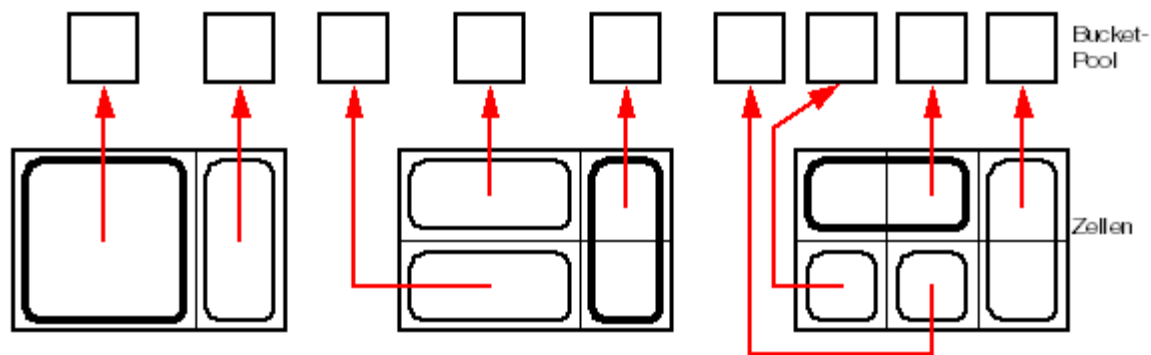


Entwurf einer Directory-Struktur

- dynamische k-dim. Matrix GD (auf Externspeicher)
- k eindimensionale Vektoren S_i (im Hauptspeicher)

Operationen auf dem Gitterverzeichnis

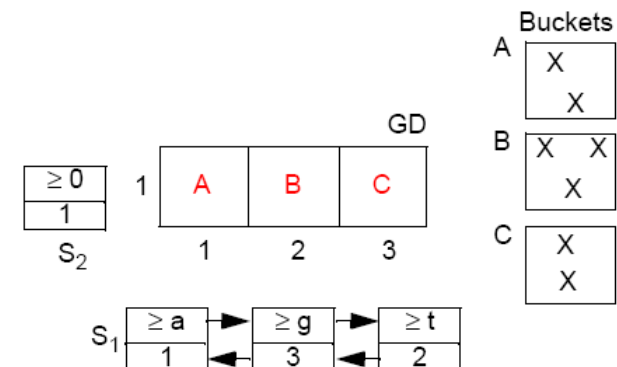
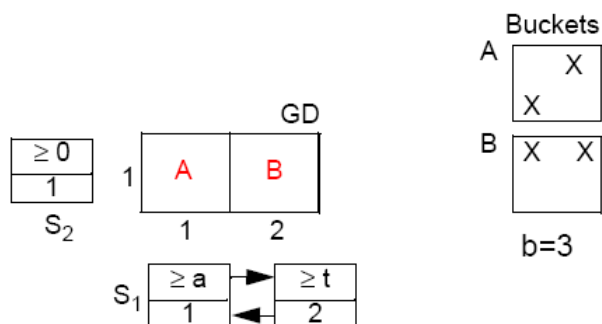
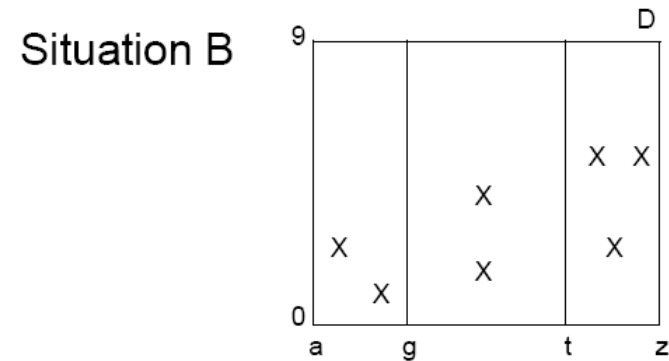
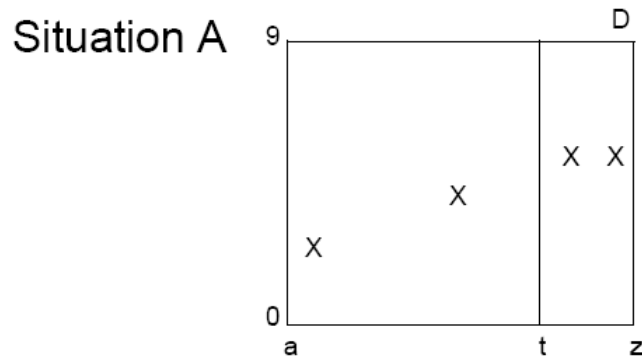
- direkter und relativer Zugriff (NEXTABOVE, NEXTBELOW) auf einen GD-Eintrag
- Mischen zweier benachbarter Einträge einer Dimension (mit Umbenennung der betroffenen Einträge)
- Splitten eines Eintrages einer Dimension (mit Umbenennung)
- Schachtelförmige Zuweisung von Zellen zu Buckets





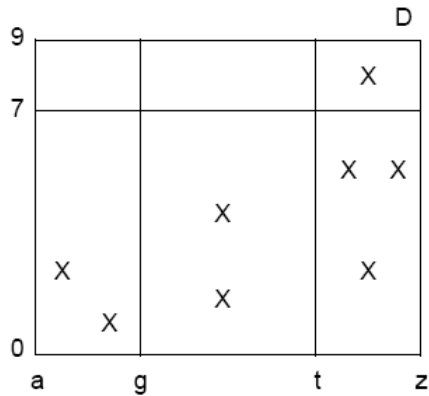
Skalierungsvektoren als zweifach gekettete Listen

- Indirektion erlaubt es, das Gitterverzeichnis an den Rändern wachsen zu lassen.
- Minimierung des Änderungsdienstes am Gitterverzeichnis
- Stabilität der Einträge im Gitterverzeichnis

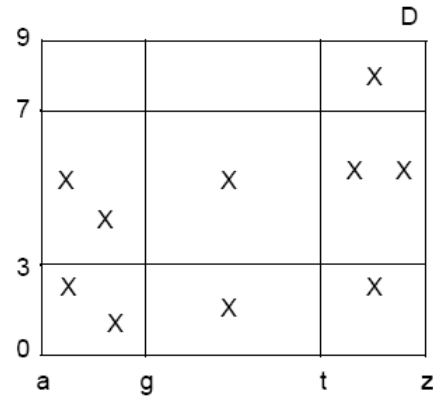




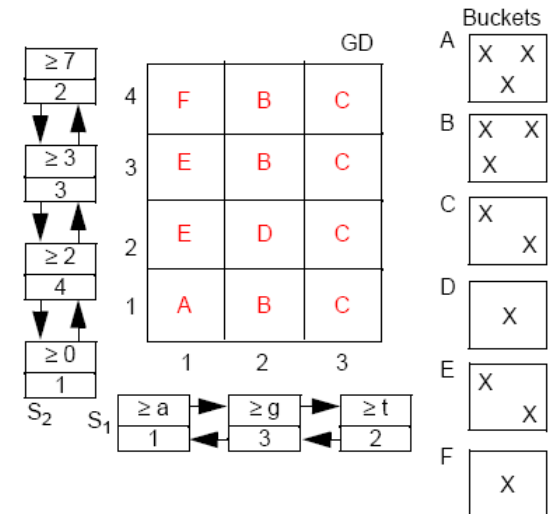
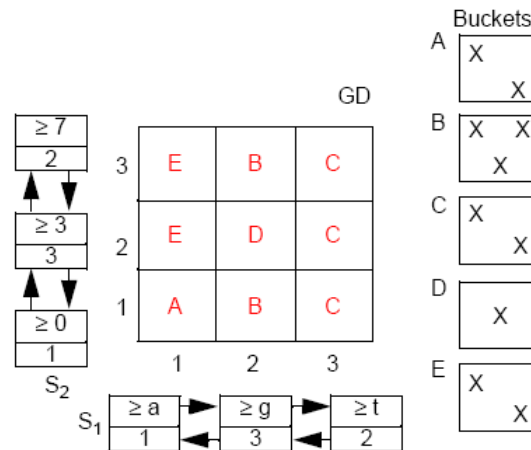
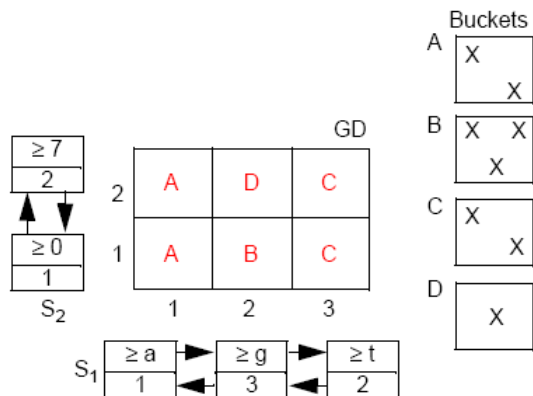
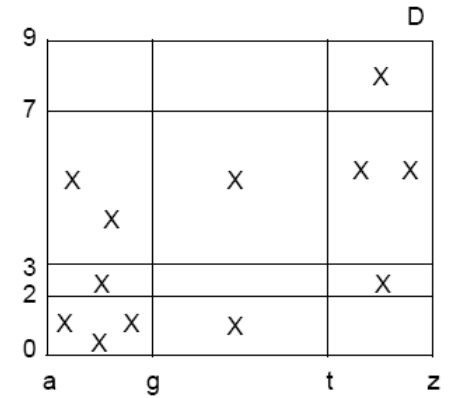
Situation C



Situation D



Situation E





Durch das Grid-File Konzept sind nur folgende Aspekte vorgegeben

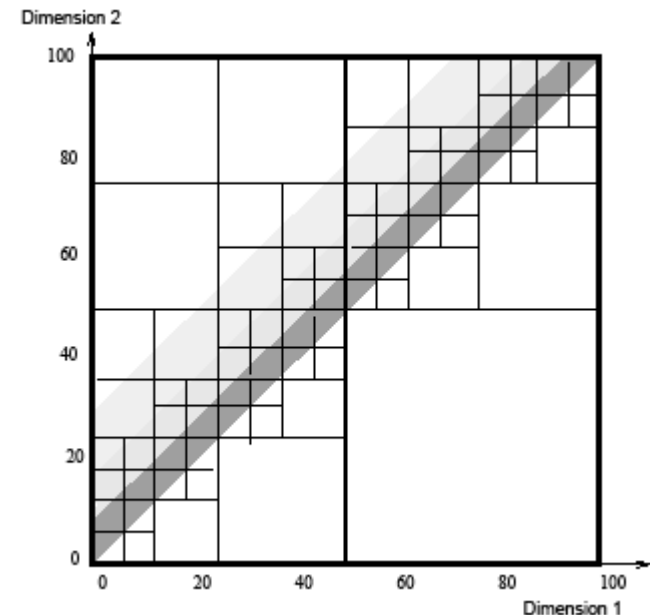
- Die Partitionierung des Suchraums durch die Zuordnung von Grid-Blöcken zu Buckets
- Die Verwendung eines Grid-Verzeichnisses

*Weitere Entscheidungen
(abh. von Implementierung)*

- Bestimmung des Split-Verfahrens
- Bestimmung des Merge-Verfahrens
- Implementierung des Grid-Verzeichnisses
- weitere Aspekte wie Mehrbenutzerbetrieb, usw.

Probleme

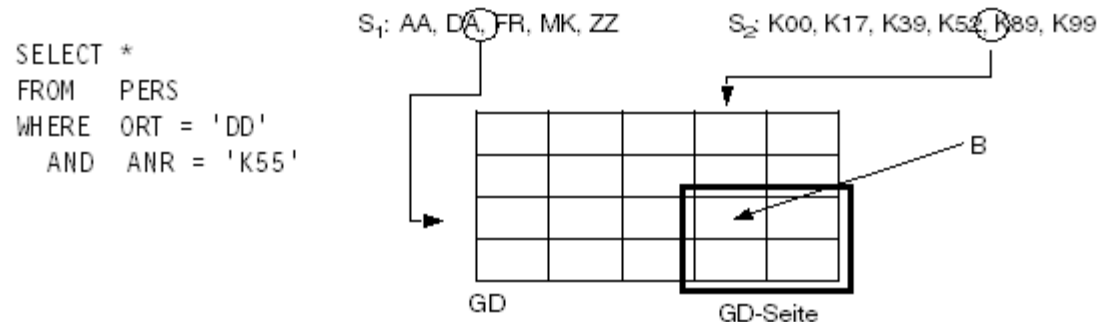
- ungleichmäßige Verteilung
z.B. nach Eck-/Mittentransformation



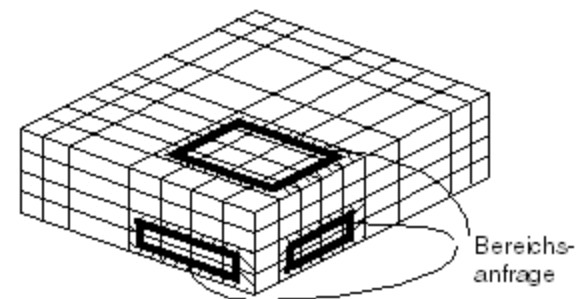


Beantwortung der mehrdimensionalen Anfragetypen mit einem Gridfile

- Exakte Übereinstimmung (exact match)
 - Benutzung der Skalen zur Ermittlung von Zeile und Spalte der Matrix; Zugriff auf den im Matricelement genannten Bucket



- Teilweise Übereinstimmung (partial match)
 - Benutzung einer Skala, z. B. der Spaltenskala, zur Ermittlung einer ganzen Matrixspalte; Zugriff auf alle Buckets in der Spalte
- Bereichsanfrage (range)
 - Ermittlung einer Teilmatrix mit Hilfe der Skalen





Zerlegungsprinzip

- Partitionierung durch Hashfunktionen in jeder Dimension
- Aufteilung der Bucketadresse in k Teile (k = Anzahl der Schlüssel)
 - eigene Hash-Funktion $h_i()$ für jeden Schlüssel i , die Teil der Bucketadresse bestimmt
 - Anzahl der Buckets sei 2^B (Adresse = Folge von B Bits)
 - jede Hash-Funktion h_i liefert b_i Bits mit
 - Satz $t = (a_1, a_2, \dots, a_k, \dots)$ gespeichert in Bucket mit $\text{Adr.} = [h_1(a_1) \mid h_2(a_2) \mid \dots \mid h_k(a_k)]$
- Vorteile
 - kein Index, geringer Speicherplatzbedarf und Änderungsaufwand

Anfrageunterstützung

- Exact-Match-Anfragen:
Gesamtadresse bekannt \Rightarrow Zugriff auf 1 Bucket
- Partial-Match-Anfrage: Eingrenzung des Suchraumes ($A_i = a_i$):
Anzahl zu durchsuchender Buckets reduziert sich um 2^{b_i} auf $N_B = 2^B / 2^{b_i} = 2^{B-b_i}$



Anwendungsbeispiel

PNR: INT (5) $b_1 = 4$
 SVNR: INT (9) $b_2 = 3$
 ANAME: CHAR (10) $b_3 = 2$
 ➔ B=9 (512 Buckets)

$h_1(\text{PNR}) = \text{PNR} \bmod 16$
 $h_1(58651) = 11 \textcircled{R} 1011$

$h_2(\text{SVNR}) = \text{SVNR} \bmod 8$
 $h_2(130326734) = 6 \textcircled{R} 110$

$h_3(\text{ANAME}) = L(\text{ANAME}) \bmod 4$
 $h_3(\text{XYZ55}) = 1 \textcircled{R} 01$

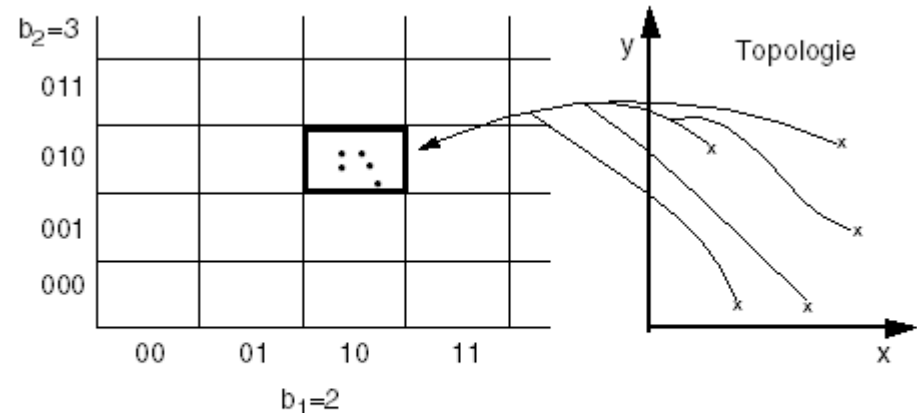
▪ B-Adr. = 101111001

Anzahl der Zugriffe

- Exact-Match-Anfragen: Zugriff auf 1 Bucket
- Partial-Match-Anfragen:

PNR = 58651 ➔ $N_B = 2^9 / 2^4 = 32$

(PNR = 73443) AND (SVNR = 2332) ➔ $N_B = 2^9 / 2$



Bucket mit Adresse '10010' enthält alle Sätze mit

$h_1(a_1) = '10'$ und $h_2(a_2) = '010'$



Idee

- Abwechselnd von verschiedenen Zugriffsattributen werden die Bits der Adresse berechnet

Ansatz von Kuchen

- Hash-Werte sind Bit-Folgen, von denen jeweils ein Anfangsstück als aktueller Hash-Wert dient
- Berechnung von je einem Bit-String pro beteiligtem Attribut
- Zyklische Bestimmung der Anfangsstücke nach dem Prinzip des Bit-Interleaving
- Bestimmung des Hash-Wertes reihum aus den Bits der Einzelwerte



Gegeben

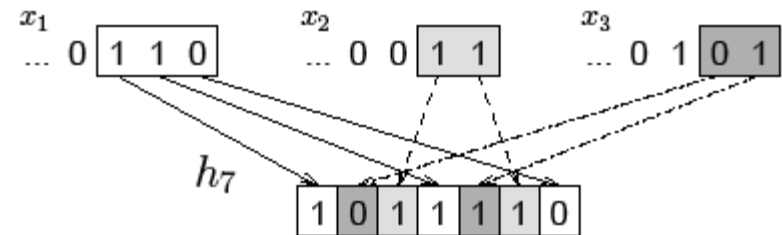
- mehrdimensionaler Schlüsselwert: $x = (x_1, \dots, x_k) \in D = D_1 \times \dots \times D_k$
- Folge von Hashfunktionen unter Berücksichtigung der jeweils i-ten Anfangsstücke der lokalen Hashwerte $h_{ij}(x_j)$
 $h_i(x) = h_i(h_{i1}(x_1), \dots, h_{ik}(x_k))$
- Lokale Hashfunktion $h_{ij}()$ ergeben Bitvektor der Länge z_{ij}
 $h_{ij} : D_j \rightarrow \{0, \dots, z_{ij}\}, j \in \{1, \dots, k\}$
- Kompositionsfunktion h_i setzt lokale Bitvektoren zu einem Bitvektor der Länge i zusammen
 $h_i : \{0, \dots, z_{i1}\} \times \dots \times \{0, \dots, z_{ik}\} \rightarrow \{0, \dots, 2^{i+1}-1\}$
- Vergrößerung der Längen zyklisch bei jedem Erweiterungsschritt

$$z_{ij} = \begin{cases} 2^{\lfloor \frac{i}{k} \rfloor + 1} - 1 & \text{für } j-1 \leq (i \bmod k) \\ 2^{\lfloor \frac{i}{k} \rfloor} - 1 & \text{für } j-1 > (i \bmod k) \end{cases}$$



Veranschaulichung

- 3 Dimensionen, Schlüsselwert $i = 7$
- unterlegte Teile des Bitvektors: $h_{71}(x_1)$, $h_{72}(x_2)$ und $h_{73}(x_3)$
- Schritt zu $i=8$: Verwendung eines weiteren Bits von x_2 , d.h. $h_{82}(x_2)$



Komplexität

- Exact-Match-Anfragen: $O(1)$
- Partial-Match-Anfragen, bei t von k Attributen festgelegt: $O(n^{1-t/k})$
- bestimmte Bits sind “unknown”
- Spezialfälle: $O(1)$ für $t = k$, $O(n)$ für $t = 0$



Raumzugriffsstrukturen



Eigenschaften räumlicher Objekte

- allgemeine Merkmale wie Name, Beschaffenheit, . . .
- Ort und Geometrie (Kurve, Polygon, . . .)

Indexierung räumlicher Objekte

- genaue Darstellung?
- Objektapproximation durch schachtelförmige Umhüllung - effektiv!
-> dadurch werden Fehltreffer möglich

Probleme

- neben Objektdichte muss Objektausdehnung bei der Abbildung und Verfeinerung berücksichtigt werden
- Objekte können andere enthalten oder sich gegenseitig überlappen

Klassifikation der Lösungsansätze

- Punkttransformation
- Clipping
- Überlappende Regionen



Point Query

- gegeben ein Punkt P; finde alle geometrischen Objekte, die P enthalten

Window Query

- gegeben ein Rechteck R; finde alle geometrischen Objekte, die R schneiden

Region Query (Intersection Query)

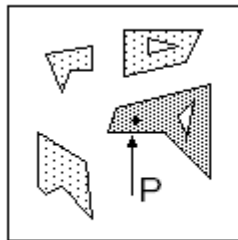
- gegeben ein EPL E; finde alle geometrischen Objekte, die E schneiden

Enclosure Query

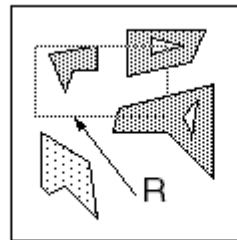
- gegeben ein EPL E; finde alle geometrischen Objekte, die in E enthalten sind

Containment Query

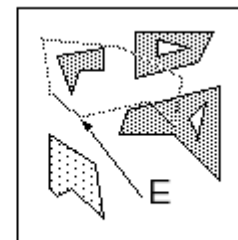
- gegeben ein EPL E; finde alle geometrischen Objekte, die E vollständig enthalten



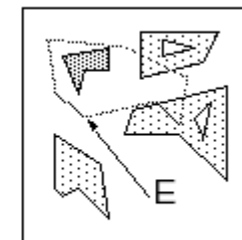
Point Query



Window Query



Region Query



Enclosure Query



Fig. 3. Point Query



Fig. 4. Window Query



Fig. 5. Intersection Query



Fig. 6. Enclosure Query



Fig. 7. Containment Query



Fig. 8. Adjacency Query

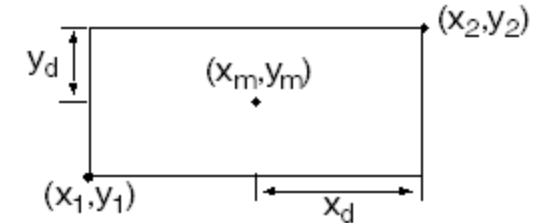


Prinzip

- Überführung der minimal umgebenden n-dimensionalen Rechtecke in 2n-dimensionale Punkte
- Abspeicherung in einer 2n-dimensionalen Punktzugriffsstruktur

Mittentransformation

- Beschreibung des Rechtecks durch den Mittelpunkt (x_m, y_m) und die jeweilige halbe Ausdehnung x_d und y_d
- Anfragegrenzen verlaufen nicht mehr orthogonal zur Datenraumpartitionierung
=> komplexere Implementierung der Anfrage
=> mehr Datenseiten werden geschnitten

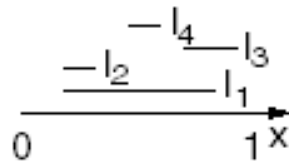


Eckentransformation

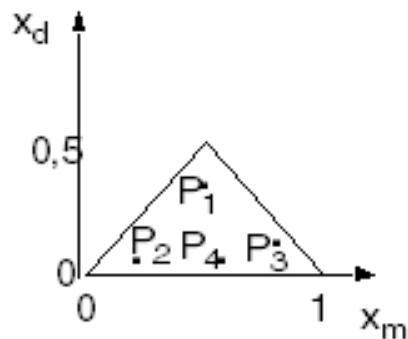
- Repräsentation des Rechtecks durch diagonal gegenüberliegende Eckpunkte (x_1, y_1) und (x_2, y_2)
- die meisten Daten liegen auf einer Diagonalen durch den Datenraum
=> starke Korrelation der Daten



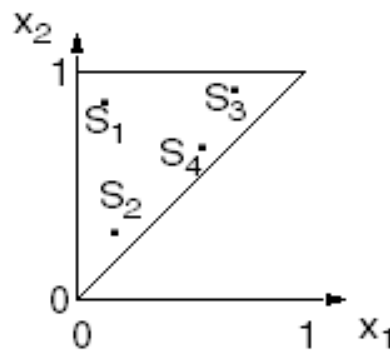
Transformation
von Intervallen I_j



in 2-dim. Punkte:

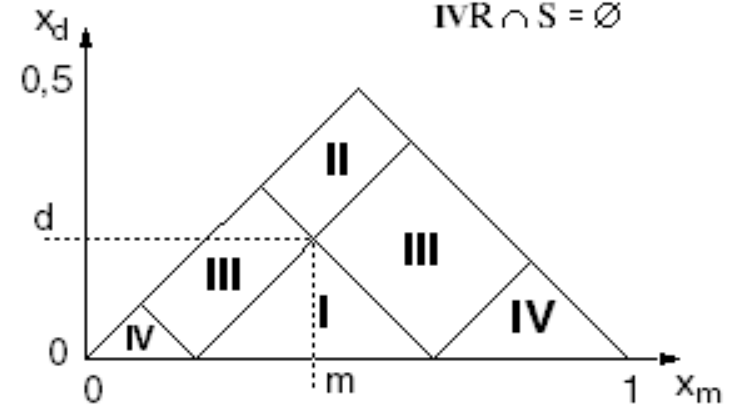


$P_j = (x_m, x_d)$
(Mittentransformation)



$S_j = (x_1, x_2)$
(Eckentransformation)

- I $R \subseteq S$
- II $R \supseteq S$
- III $R \cap S \neq \emptyset$
- IV $R \cap S = \emptyset$



Anfrageräume zur Suche von
Intervallen R bzgl. eines Intervalls
 $S = (m, d)$ (Mittentransformation)

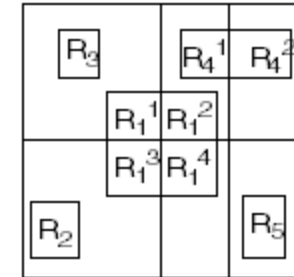


Idee

- Rechteck wird in jede Datenregion eingefügt, die es schneidet

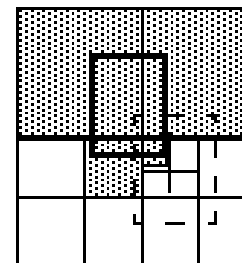
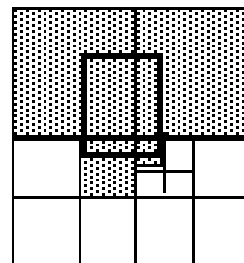
Eigenschaften

- Zahl der Datensätze in der Indexstruktur steigt stärker als die Zahl der gespeicherten Rechtecke (Anzahl der Rechtecke in Verhältnis zum Datenraum oder Partitionierung bereits sehr fein)
- Bedarf an Überlaufseiten, so dass ein Gebiet, in dem sich mehr Rechtecke überlappen, als in eine Datenseite passen, nicht weiter partitioniert werden muss
- Einfüge- und Löschoperationen werden erheblich aufwändig
- Bereichsanfragen degenerieren in der Leistung aufgrund der hohen Zahl von mehrfach eingelesenen identischen Rechtecken



Beispiel

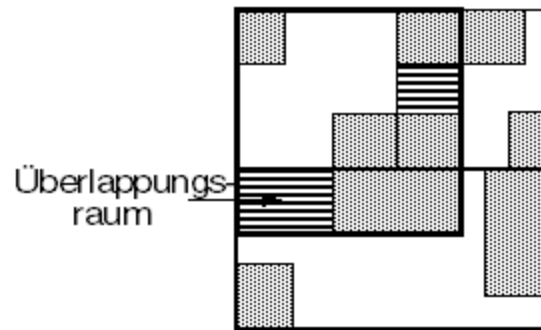
- gespeichertes Rechteck wird viermal gefunden





Idee

- Partitionierung des Datenraumes ist nicht mehr disjunkt
- Seitenregionen können überlappen und ein Clipping der Rechtecke wird überflüssig



Problem

- Überlappung der Directory-Regionen
(fällt eine Anfrage in einen Überlappungsraum, müssen mehrere Pfade im Suchbaum untersucht werden)
-> Überlappung sollte möglichst klein gehalten werden

Beispiel: R-Baum



Ziel

- Effiziente Verwaltung räumlicher Objekte (Punkte, Polygone, Quader, ...)

Hauptoperationen

- Punktanfragen (point queries)
Finde alle Objekte, die einen gegebenen Punkt enthalten
- Gebietsanfragen (region queries)
Finde alle Objekte, die mit einem gegebenen Suchfenster überlappen (in ... vollständig enthalten sind)

Ansatz

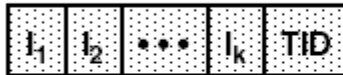
- Speicherung und Suche von achsenparallelen Rechtecken
- Objekte werden durch Datenrechtecke repräsentiert und müssen durch kartesische Koordinaten beschrieben werden
- Repräsentation im R-Baum erfolgt durch minimale begrenzende (k-dimensionale) Rechtecke/Regionen
- Suchanfragen beziehen sich ebenfalls auf Rechtecke/Regionen



R-Baum ist höhenbalancierter Mehrwegbaum

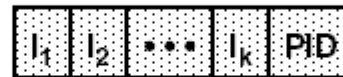
- jeder Knoten entspricht einer Seite
- pro Knoten maximal M , minimal m ($\geq M/2$) Einträge

Blattknoteneintrag:



kleinstes umschreibendes Rechteck
(Datenrechteck) für TID

Zwischenknoteneintrag:



Intervalle beschreiben kleinste
umschreibende Datenregion für
alle in PID enthaltenen Objekte

I_j = geschlossenes Intervall
bzgl. Dimension j

PID: Verweis auf Sohn

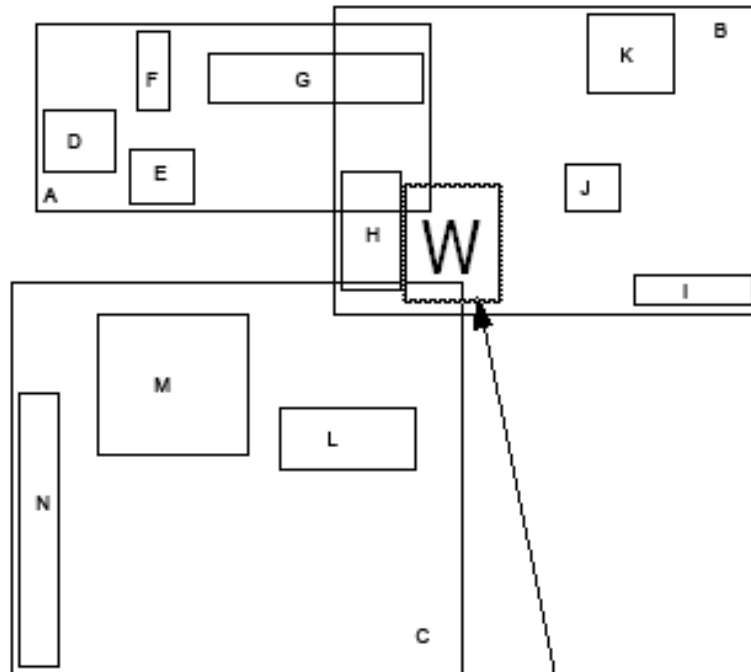
TID: Verweis auf Objekt

Eigenschaften

- starke Überlappung der umschreibenden Rechtecke/Regionen auf allen Baumebenen möglich
- bei Suche nach Rechtecken/Regionen sind ggf. mehrere Teilbäume zu durchlaufen
- Änderungsoperationen ähnlich wie bei B-Bäumen

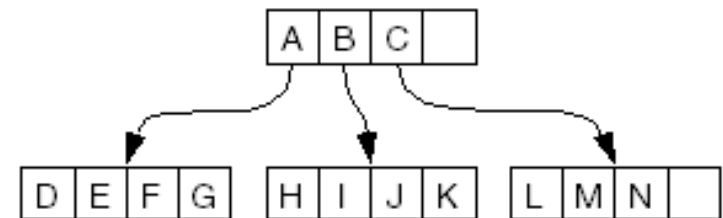


Lage der Datenobjekte



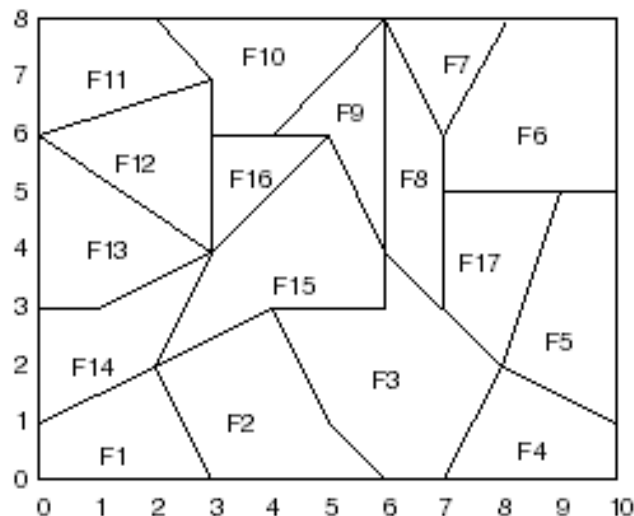
"schlechtes"
Suchfenster

korrespondierende
R-Baum-Struktur

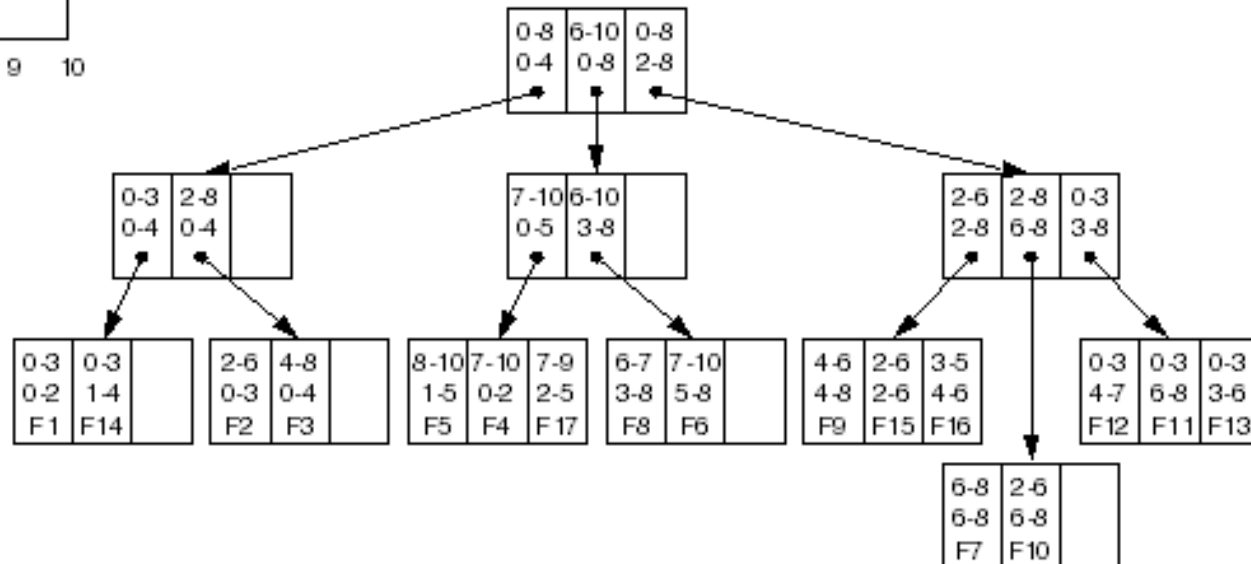




Abzuspeichernde Flächenobjekte



korrespondierender R-Baum



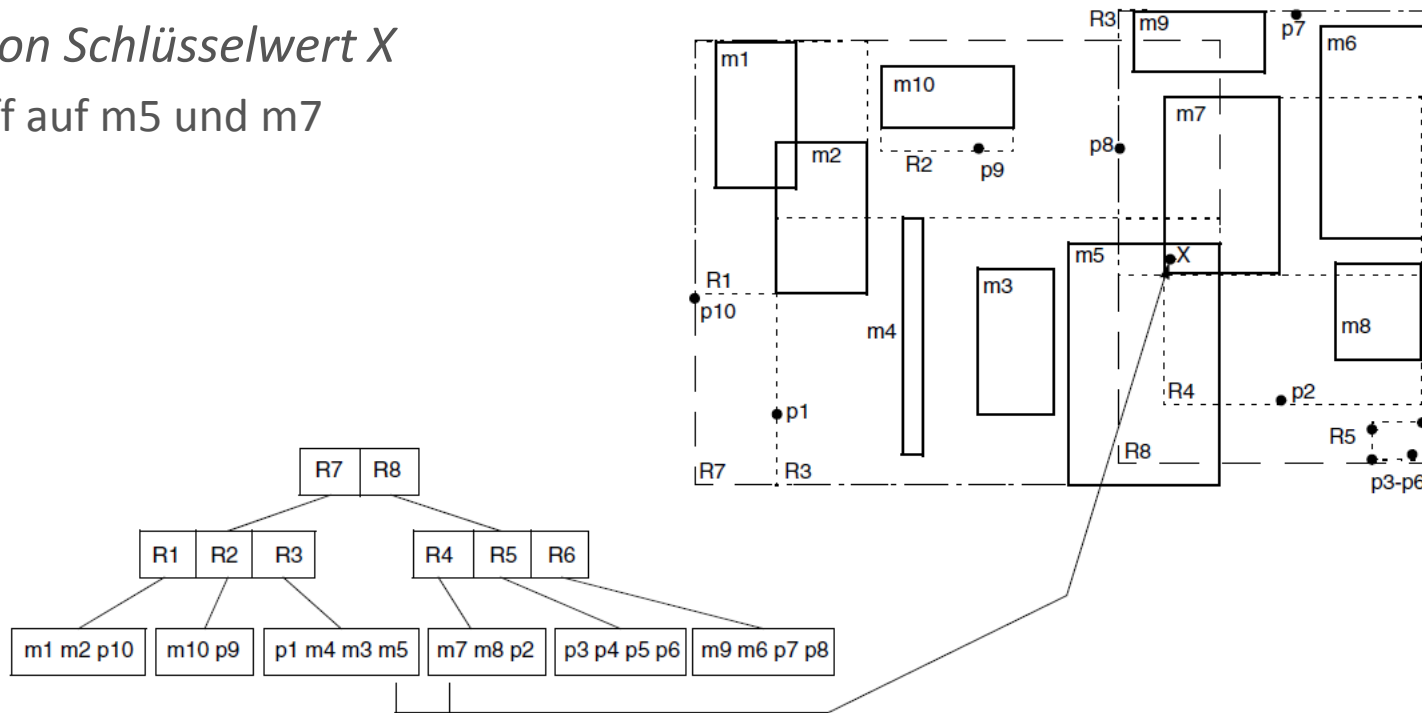


Beispiel

- Einzelne Datenpunkte p_i ($1 \leq i \leq 10$) bzw. Teilräume m_i ($1 \leq i \leq 10$)
- Überlappende Rechtecke R_i ($1 \leq i \leq 8$)

Suche von Schlüsselwert X

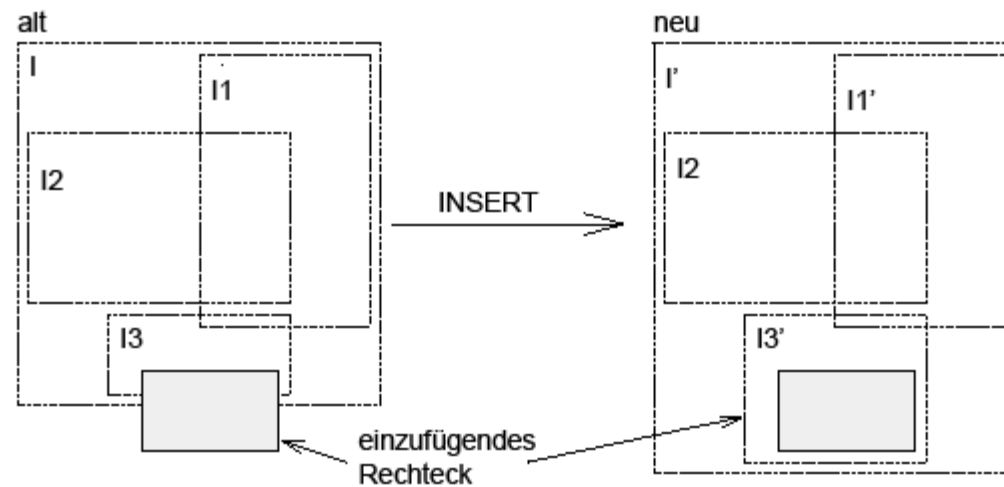
- Zugriff auf m_5 und m_7





Problem

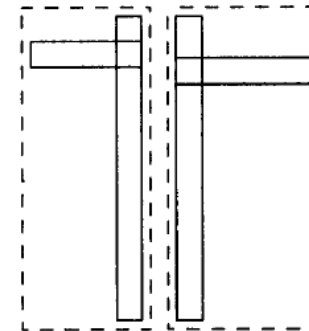
- Vergrößerung der Regionen
- aber: Einfache Strategie



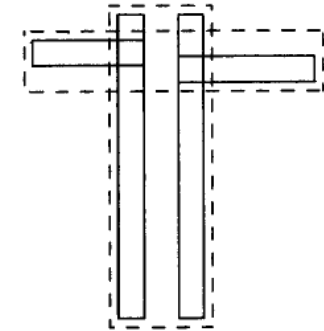


Beim Einfügen eines Eintrags in einen vollen Knoten

- Einträge des Knotens müssen auf zwei neue Knoten aufgeteilt werden
- Kriterium für die Unterteilung
 - Jeder Knoten dessen umschließenden Rechteck mit einer Suchanfrage überlappt muss durchsucht werden
 - Daher Aufteilung in Knoten mit möglichst kleinen umschließenden Rechtecken
- Betrachtung jeder möglich Aufteilung ist zu aufwändig
 - Bei M Einträgen, etwa 2^{M-1} Möglichkeiten
- Split-Suche mit quadratischem Aufwand
 - Initial wählt man das Paar von Einträgen des umschließendes Rechteck am größten ist, also die zwei Einträge die am wenigsten zueinander passen
 - Beide Einträge bilden jeweils einen neuen Knoten
 - Restlichen Einträge werden schrittweise den Knoten zugeteilt
 - Der nächstes zuzuteilenden Eintrag, ist der der auf beiden Knoten möglich unterschiedlich große umschließende Rechtecke erzeugt, der als möglich klar zuzuordnen ist
 - Der Eintrag wird dem Knoten zugeteilt, dessen umschließendes Rechtecke er am geringsten vergrößert, also der Knoten zu dem der Eintrag am besten passt



Bad split



Good split

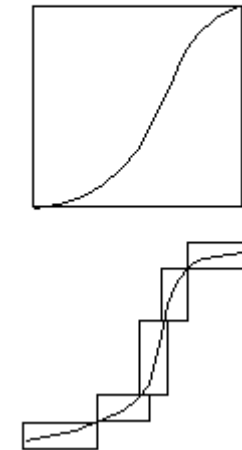


Überdeckung (coverage)

- Gesamter Bereich, um alle zugehörigen Rechtecke zu überdecken
- Minimale Überdeckung reduziert die Menge des „toten Raumes“ (leere Bereiche), der von den Knoten des R-Baumes überdeckt wird

Idee

- Anwendung des Clipping und Vermeidung von Überlappungen
- Ein Daten-Rechteck wird ggf. in eine Sammlung von disjunkten Teilrechtecken zerlegt und auf der Blattebene in verschiedenen Knoten gespeichert





Überlappung (overlap)

- Gesamter Bereich, der in zwei oder mehr Knoten enthalten ist -> effiziente Suche erfordert minimale Überdeckung und Überlappung
- Minimale Überlappung reduziert die Menge der Suchpfade zu den Blättern (noch kritischer für die Zugriffszeit als minimale Überdeckung)

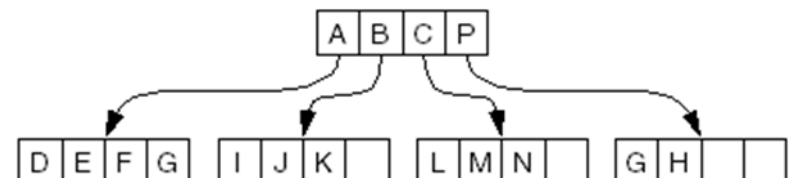
Partitionierung der Datenobjekte



Idee

- Eintrag von Objekten in mehreren Knoten
- Überdeckungsproblematik wird entschärft
- Komplexere Algorithmen für Enthaltenseins-Anfragen
- Schwierigere Wartung, keine Leistungsvorteile gegenüber R-Baum

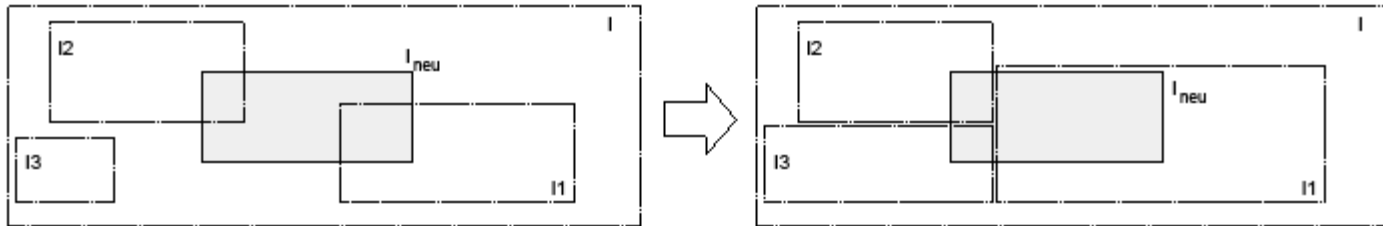
korrespondierende R⁺-Baum-Struktur



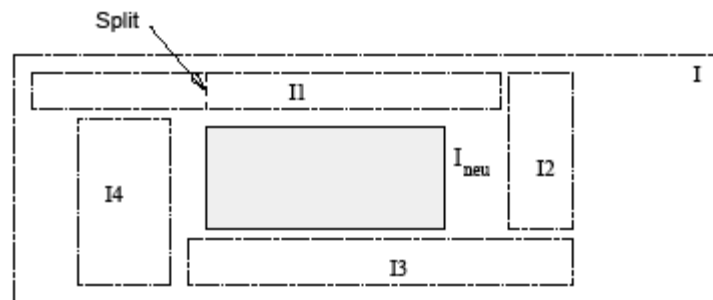


Problem

- Modifikation mehrerer Rechteckregionen
- sehr umfassende Erweiterung von Bereichen möglich



- Gefahr unvermeidbarer Splits durch gegenseitige Blockierung für eine Erweiterung
 - Änderungsoperationen sind nicht mehr auf den Pfad vom Blatt zur Wurzel beschränkt
 - keine obere Grenze für Einträge im Blattknoten





Ziel

- Effiziente Verwaltung räumlicher Objekte (Punkte, Polygone, Quader, ...)
- Besonders gut für Polygone und Anfrage gegen deren Struktur




Ansatz

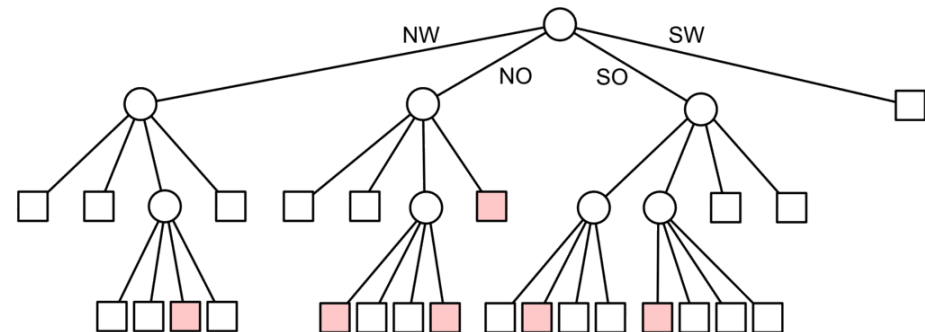
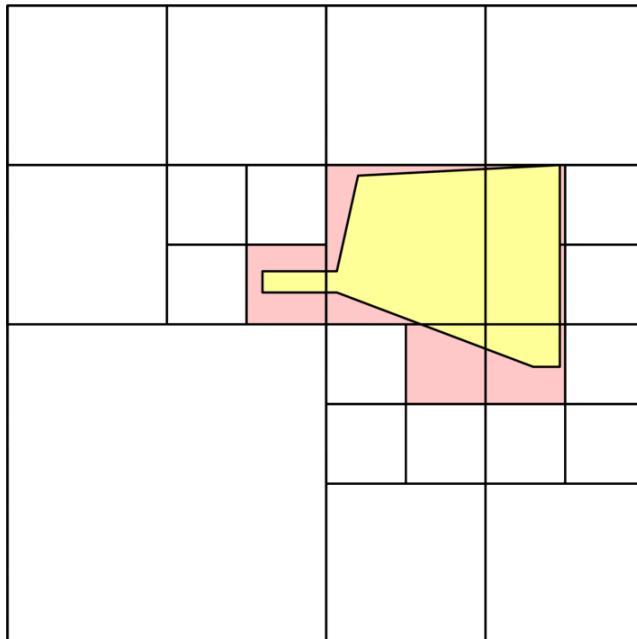
- Zwei-dimensional
- Einzelne Objekte werden rekursive dekomponiert
- Ein quadratischer Raum wird rekursive in jeweils vier gleichgroße, nicht-überlappende Quadrate unterteilt
- Bis der Inhalt eine Quadrat ausreichend einfach ist um ihn mit herkömmlichen Datenstrukturen zu speichern
- Untergliederung nähert das Objekt an

Voraussetzung

- Objekte liegen in einem klar und dauerhaft begrenzten Raum
- Gesamtmenge der Objekte füllt diesen Raum aus
- Trifft auf GEO-Objekte und Kartendaten zu, gegebene Begrenzung (Deutschland, Europa, Erde) werden gut ausgefüllt



- Der Bereich ist in vier gleichgroße Teile unterteilt – jedes Teil kann rekursiv wiederum bis zu einer minimalen Größe weiter unterteilt werden
- Die minimale Teilegröße gibt die Genauigkeit vor, mit der die Geometrie angenähert wird
- Ein Teil ist entweder
 - belegt 
 - nicht belegt 
 - ist in genau vier weitere Teile unterteilt 

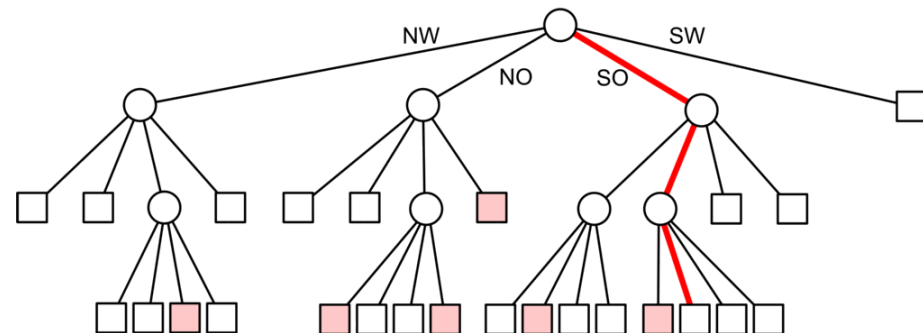
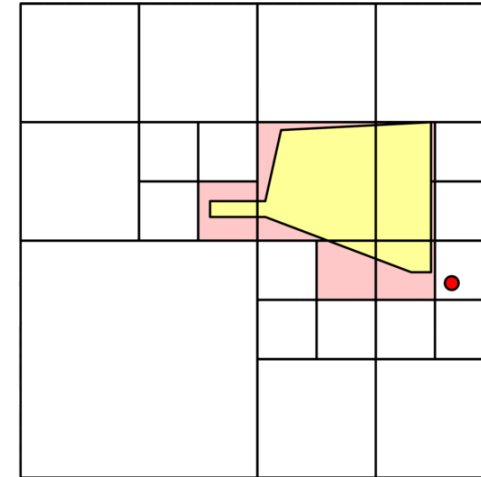




Für jede Geometrie wird ein eigener Quadtree angelegt

Grob-Prüfung Punkt-in-Bereich:

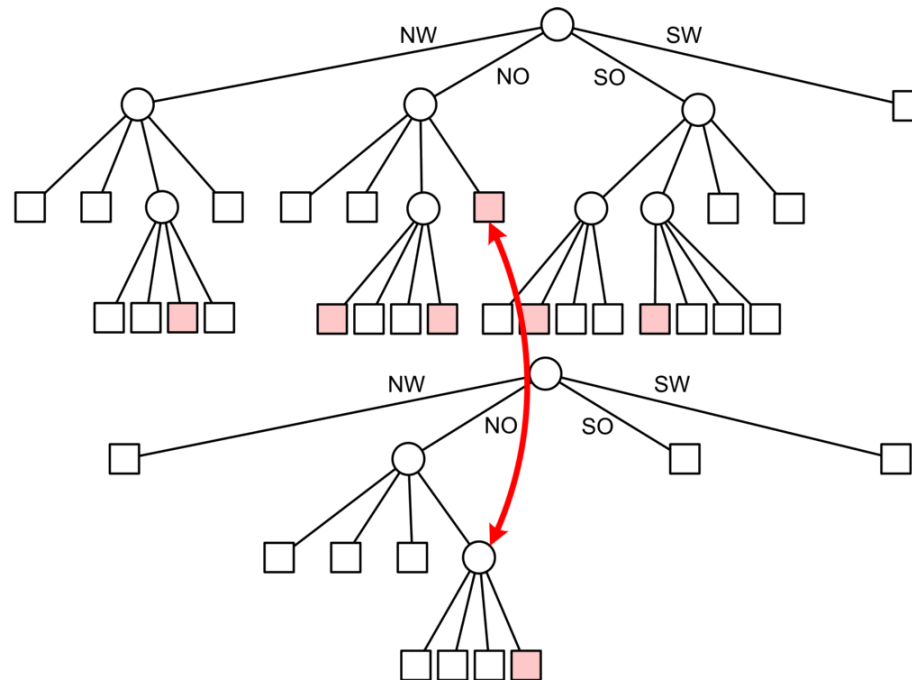
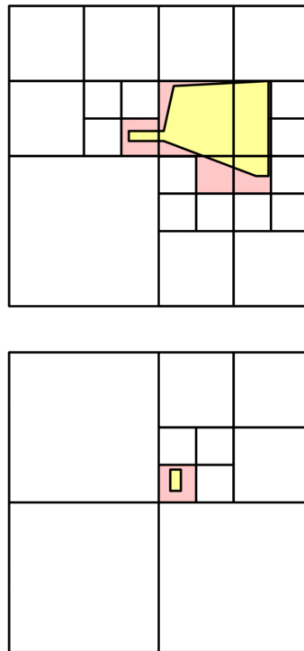
- Steige den Baum gemäß der Punktlage ab
- Ist ein Blatt erreicht – prüfe ob belegt





Grob-Prüfung Überlappung:

- Zwei Quadrees beschreiben eine Überlappung, wenn es mindestens zwei Knoten in den jeweiligen Bäumen gibt, die die gleiche Lage haben *und*
 - beide Knoten sind belegte Blätter *oder*
 - ein Knoten ist ein belegtes Blatt, einer ein innerer Knoten





Zwei Datentypen

- GEOMETRY für planare Daten (a), erfordert Definition von Bounding Box
- GEOGRAPHY für geographische Daten (b), Geoid definiert Gesamtbereich
- Zur Indizierung werden Quadrees benutzt

Anlegen

```
CREATE TABLE Cities (  
    CityId int IDENTITY (1,1),  
    CityName nvarchar(20),  
    CityGeo geometry);  
CREATE TABLE Streets (  
    StreetId int IDENTITY (1,1),  
    StreetName nvarchar(20),  
    StreetGeo geometry);
```



(a)



(b)

Einfügen

```
INSERT INTO Cities (CityName, CityGeo) VALUES ('MyCity',  
    geometry::STGeomFromText('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));  
INSERT INTO Streets (StreetName, StreetGeo) VALUES ('First Avenue',  
    geometry::STGeomFromText('LINESTRING (100 100, 20 180, 180 180)', 0));
```

Anfragen

```
SELECT s.StreetName, c.CityName FROM Cities c, Streets s  
WHERE s.StreetGeo.STIntersects(c.CityGeo)=1
```

[Fang, Y.; Friedman, M.; Nair, G.; Rys, M. & Schmid, A.-E. Spatial indexing in Microsoft SQL Server 2008. *SIGMOD'08, ACM*, **2008**, 1207-1216]



Datentyp

- *sdo_geometry* für 2-4 dimensionale räumliche Daten
- Implementiert als Oracle Object Datatype
- Zur Indizierung werden R-Trees oder Quadrees benutzt

Unterstützte Operatoren

- *anyinteract*: identify data geometries that intersect the specified query geometry
- *inside*: identify data geometries that are "completely inside" the query geometry
- *coveredby*: identify data geometries that "touch" on at least one boundary and are inside the query geometry otherwise
- *contains*: reverse of *inside*
- *covers*: reverse of *coveredby*
- *touch*: identify geometries whose boundary "touches" the boundary of the query geometry but disjoint otherwise
- *equal*: identify geometries that are exactly the same as the query geometry
- *overlapbdyintersect*: identify geometries that overlap each other and boundaries intersect.
- *overlapbdydisjoint*: identify geometries that overlap with the query geometry but the boundaries are disjoint

[Kanth, K. V. R.; Ravada, S. & Abugov, D. Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. *SIGMOD'02, ACM*, **2002**, 546-557]



Nutzung eindimensionaler Zugriffspfade

- Kompromiss und schlechte Skalierung
- Verlust der physischen Nachbarschaft

Multidimensionale Strukturen

- MDB-Bäume
- KdB-Bäume
- Grid-File
- Multidimensionales Hashing

Raumzugriffsstrukturen

- R- und R⁺-Baum
- Quadrees

Literatur

- Härder, T. & Rahm, E. Datenbanksysteme: Konzepte und Techniken der Implementierung. Springer-Verlag, 1999
- Saake, G.; Heuer, A. & Sattler, K.-U. Datenbanken: Implementierungstechniken. MITP-Verlag, 2005
- Hellerstein, J. M.; Stonebraker, M. & Hamilton, J. R. Architecture of a Database System. Foundations and Trends in Databases, 2007, 1, 141-259
- Gaede, V. & Günther, O. Multidimensional Access Methods ACM Computing Surveys, 1998, 30, 170-231
- Lu, H. & Ooi, B. C. Spatial Indexing: Past and Future, IEEE Data(base) Engineering Bulletin, 1993, 16, 16-21