

Mehrdimensionale Zugriffspfade und Dateiorganisation

Gliederung

- Begriffe, Anforderungen und Probleme, Anwendungsgebiete
- Konkrete Techniken
 - Bäume
 - Grid-File
 - Mehrdimensionales Hashing

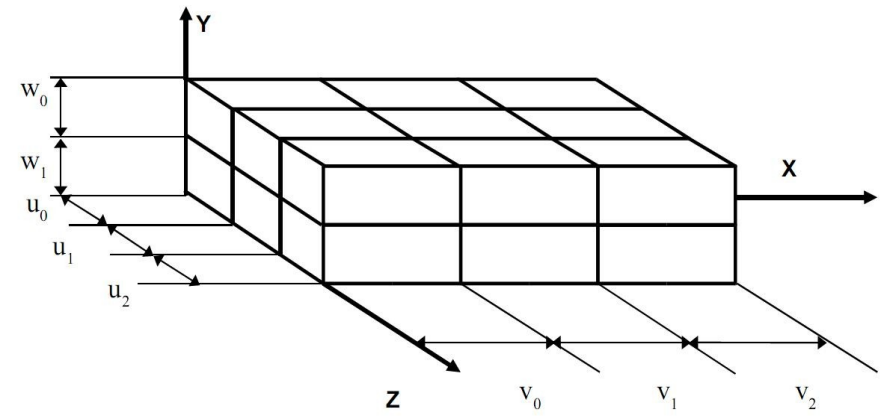
Begriffe

- Eindimensionale Zugriffe
 - Suchschlüssel besteht aus genau einem Attribut
- Mehrdimensionale Zugriffe
 - Suchschlüssel aus k Attributen (z.B. $k=2$: „Finde Angestellte aus Abteilung X mit dem Alter Y“)
 - Unterstützung durch k -dimensionalen Index für effizientere Lösung (mehrdim. Suche/Clusterung)
 - Lösungen mit eindimensionalen Indexen in vielen Fällen zu langsam/kompliziert
 - Anfrage für jedes Attribut und anschließend Schnittmengenbildung
 - Index mit konkatenierten Attributen

Begriffe(2)

- Datenraum

- aufgespannt durch k Attribute, z.B. $k=3$:
- Mehrdimensionale Dateiorganisation bedeutet Anordnung der Datensätze in diesem Raum
- Abbildung auf einen linearen Speicherbereich
- Speicherbereich in Buckets organisiert



Anforderungen

- Erhaltung der topologischen Struktur
 - benachbarte Objekte im Datenraum sollten auch benachbart auf den Speicher abgebildet werden, also in gleichen Buckets (mehrdimensionale Clusterung innerhalb dieser) -> „lokale Ordnungshaltung“
- Dynamische Reorganisation
 - um z.B. bei Inserts, Deletes Entartungen des Index zu verhindern
 - balancierte Zugriffsstruktur für gleichförmig schnellen Zugriff

Anfragetypen

- Anfragen die für jeden Schlüssel einen Wert vorgeben („exact match query“), Anfragen für Teile des Schlüssels („partial match query“)
- Bereichsanfragen („range / partial range“)
- nearest-neighbour-Anfragen („best match query“)
- point-, region-, und containment-query

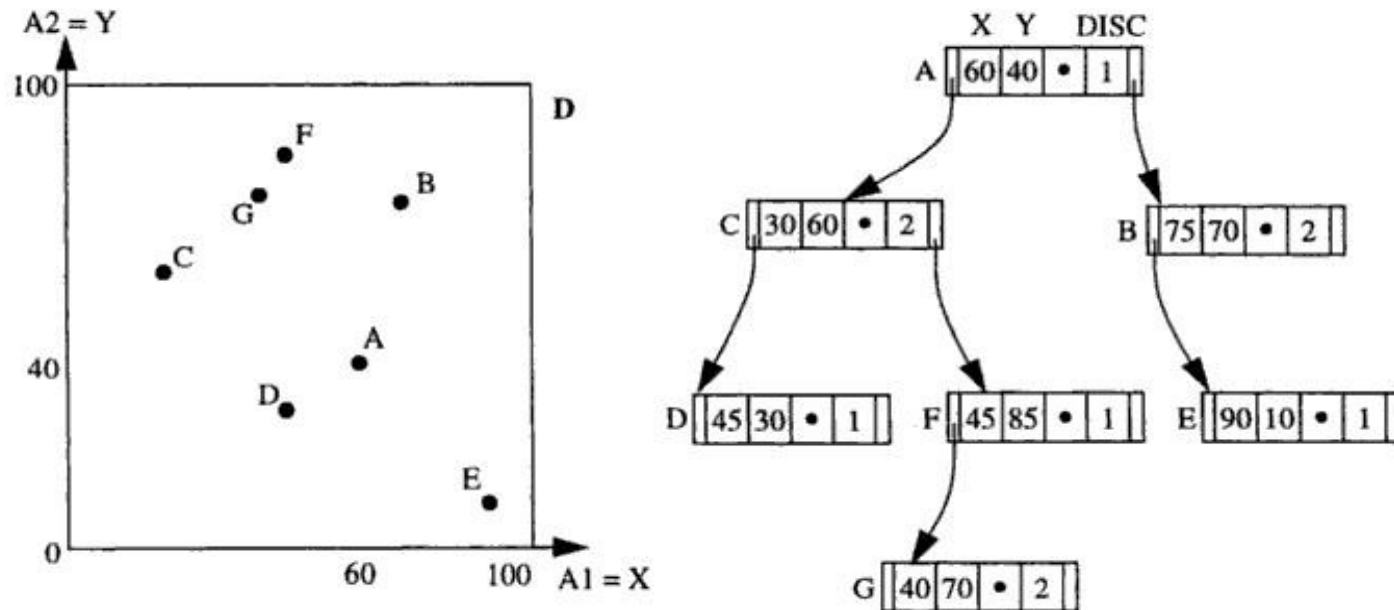
Anwendungsgebiete

- Geographische Informationssysteme
- Geometrische Daten (CAD, Medizin)
- MMDBMS (Video, Audio, Bilder)
- Effizientere Verarbeitung von Anfragen in herkömmlichen DB-Anwendungen

Homogener k-d-Baum (1)

- k-dimensionaler binärer Suchbaum
 - Gesamtschlüssel des Index setzt sich aus k Teilschlüsseln zusammen
 - Verfahren wie im normalen bin. Suchbaum (jeder Knoten 2 Pointer, auf linkes und rechtes Child)
 - jeder Knoten enthält einen Datensatz
 - zusätzlich ein „Diskriminator“ pro Knoten
 - bezeichnet Teilschlüssel nachdem die Kindknoten geordnet werden sollen und durchläuft alle Dimensionen
 - Berechnung z.B. für die Baumebene i: $d=(i\%k)+1$

Homogener k-d-Baum (2)



- Abhängigkeit von der Reihenfolge des Aufbaus (obiger Baum wurde mit „A,B,...,G“ aufgebaut)
- z.B. Entartung für Einfügereihenfolge „E,F,B,G,A,C,D“

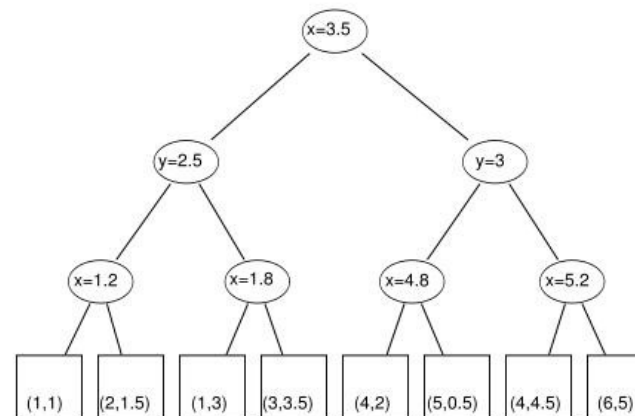
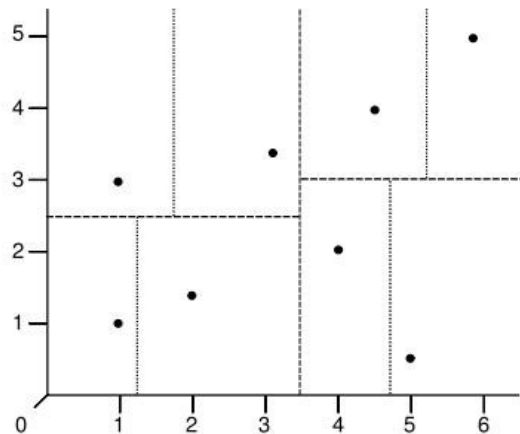
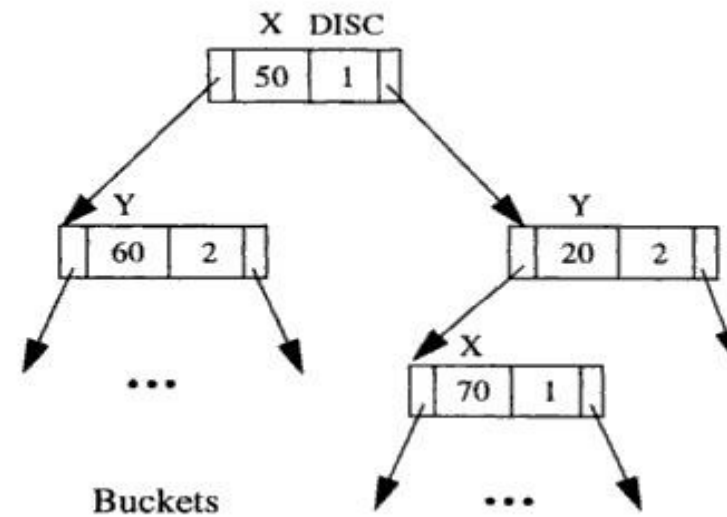
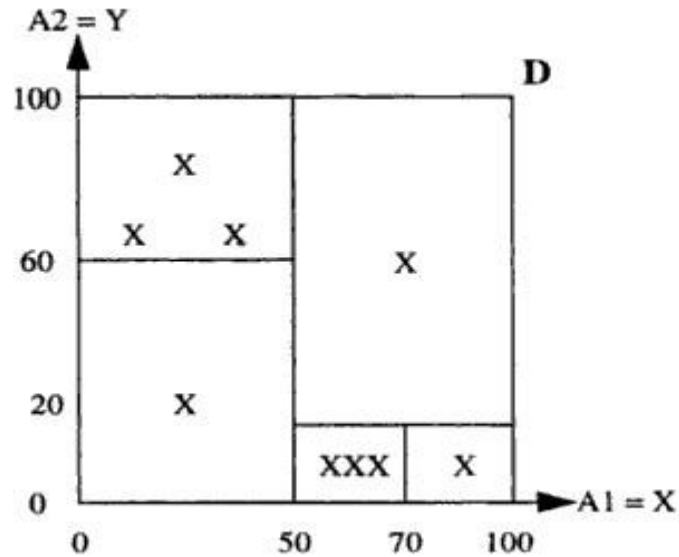
Homogener k-d-Baum (3)

- Nachteile:
 - Kein Balancierungskriterium (Entartung mögl.)
 - Keine Regeln zur Abbildung auf den Speicher (keine Clusterung bei Zuordnung der Knoten zu Buckets)
 - Bereichsanfragen oder partial match queries müssen über Rekursion abgewickelt werden
 - daher nicht für DBS geeignet
- Prinzip „Organisation der Datensätze“

Heterogener k-d-Baum (1)

- wie homogener k-d-Baum, aber Datensätze nur in Blättern
 - Buckets 1:1 auf Blätter abgebildet
 - -> bietet Topologieerhaltung
 - Bei Bucketüberlauf -> Split -> Ordnung nach nächstem Teilschlüssel (wechselt wie bei homogenem k-d-Baum)

Heterogener k-d-Baum (2)



Heterogener k-d-Baum (3)

- Prinzip „Organisation des Datenraums“
- Verfeinerungsprinzip „Divide and Conquer“
- Nachteile:
 - Keine Balancierungsgewährleistung
 - -> Struktur immer noch abhängig von Einfügereihenfolge und Objektverteilung im Datenraum
 - Bereichsanfragen/partial match (s. Hom. K-d-Baum)
- Weiterentw.: LSD-Baum (local-split-decision)
 - Verbesserung der Bucketzuordnung
 - auch für räumlich ausgedehnte Objekte nutzbar

k-d-B-Baum

- Mischung aus heterogenem k-d-Baum und B*Baum
- Balancierung bewirkt gleichlange Wege zu allen Blättern -> gleichschneller Zugriff auf alle Datensätze
- sehr komplexe dynamische Reorganisation
 - Bucketüberlauf zieht Verfeinerung der inneren Knoten nach sich -> erzwingt evtl. Fortsetzung der Reorganisation bis zur Wurzel
 - dabei muss k-d-Baum Prinzip gewahrt bleiben (Ordnung nach den Dimensionen)

K-d-B-Baum (2)

- experimentell ermittelte Speicherplatzausnutzung von 60% +/-10%
- keine garantierte Speicherausnutzung >50%
- Weiterentwicklung: hB-Baum („holey-brick tree“)
 - bietet bessere Aufteilung von orthogonalen Linien von Punktmengen
 - hohe Effizienz, unabhängig von der Verteilung der Punkte im Datenraum
 - gut für DBS geeignet

Grid-File

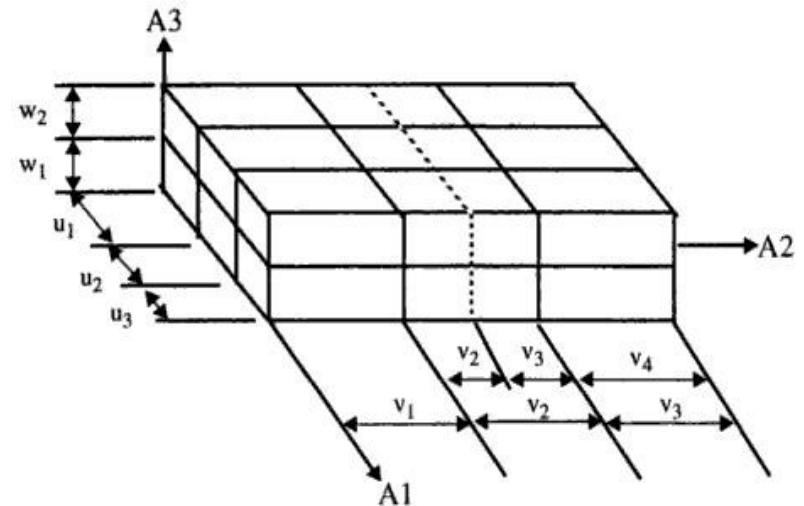
- ebenfalls basierend auf dem Prinzip der „Organisation des Datenraums“
 - Datenraum wird in k-dimensionale Quader (Zellen) zerlegt
 - bessere Aufteilung des Raumes als bei k-d-B-Baum
 - Dynamische Reorganisation der Gridstruktur bei Insert/Delete
- konzipiert für exact match, range queries und nearest neighbour Anfragen

Grid-File (2)

- für Look-Up garantiert nur 2 Plattenzugriffe benötigt
- funktioniert nicht mehr nach „Divide and Conquer“ sondern nach „Dimensionsverfeinerung“
 - beim Überlauf einer Zelle wird diese durch die vorher bestimmte Dimension geteilt

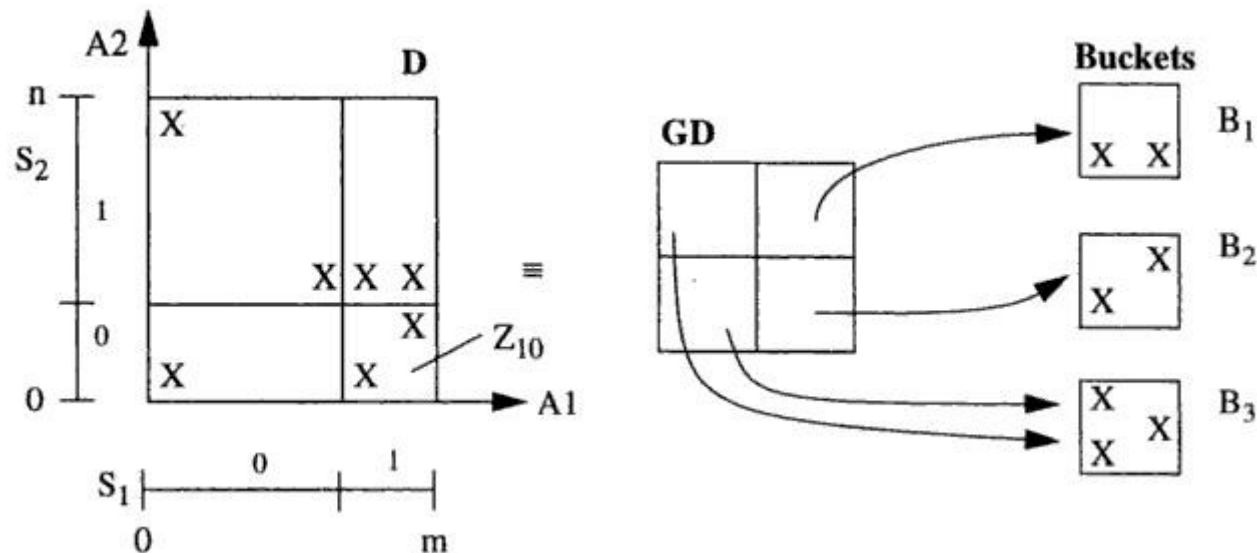
Grid-File (3)

- alle Zellen auf der gleichen Dimensionsachse auch geteilt, wobei nicht zwangsläufig für jede Zelle ein extra Bucket angelegt wird (Zellen mit zu wenig Punkten können mit Nachbarzellen zusammen auf gemeinsame Speicherstellen gelegt werden)
- ->weniger Buckets

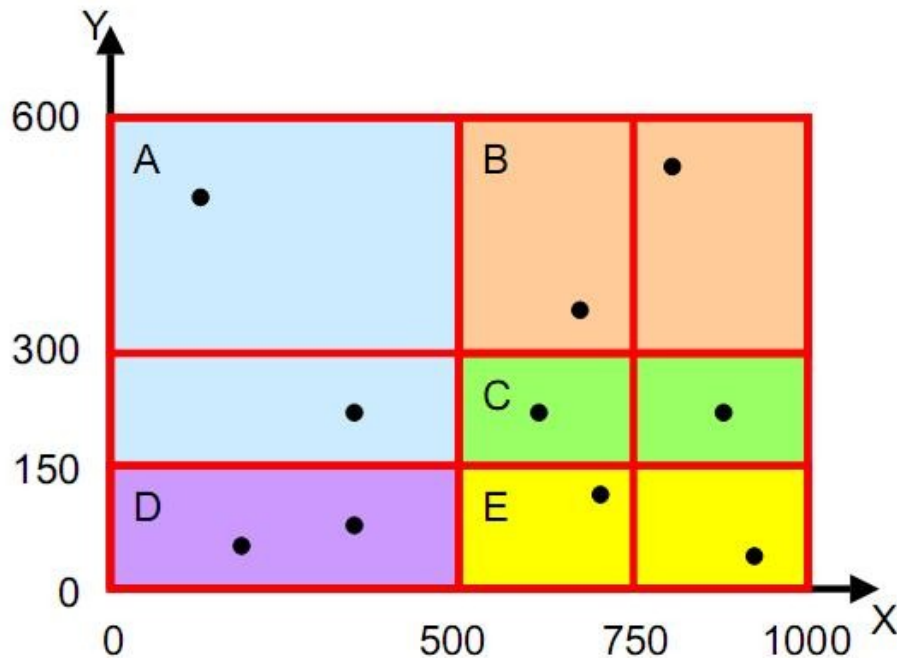


Grid-File (4)

- Aufbau:
 - k Arrays („scales“) jeweils zum Speichern der Unterteilung der k-ten Dimensions-Achse
 - k-dimensionales Grid-Directory zum Speichern der Bucket-Adressen, Bsp. für 3-stufige Buckets:



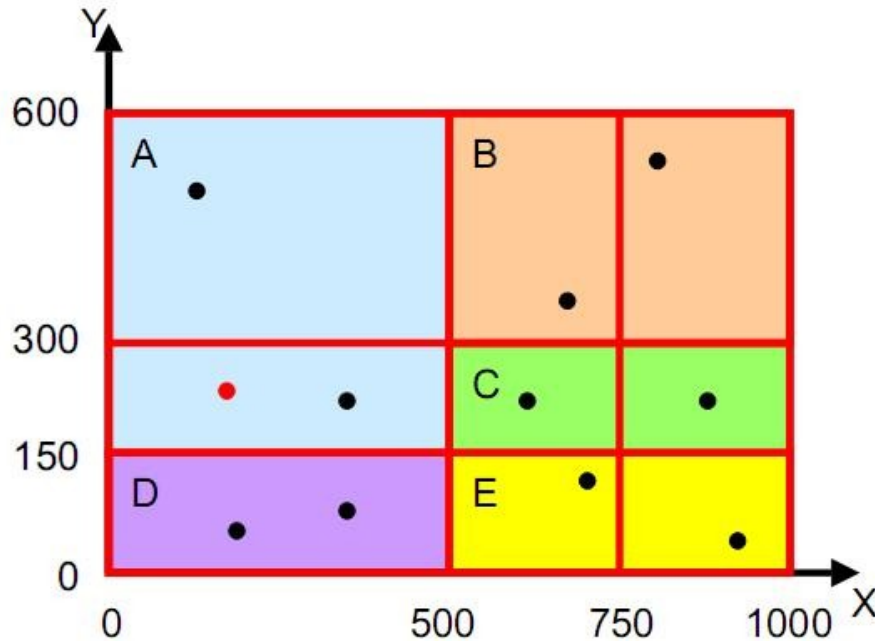
Gridfile (5)



- Grid-File und Directory vor dem Einfügen eines Wertes (Bucketgröße=2)

Y \ X			
	< 500	< 750	< 1000
< 600	A	B	B
< 300	A	C	C
< 150	D	E	E

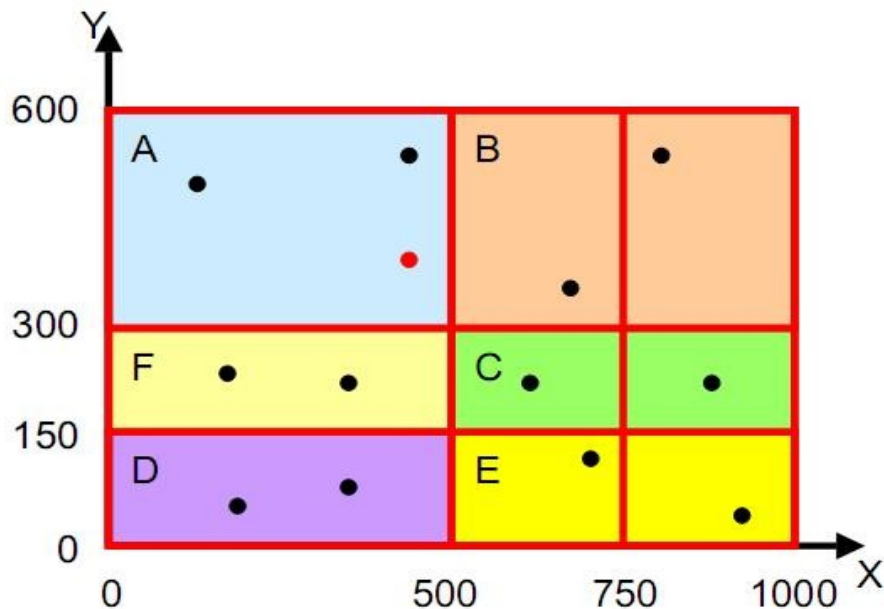
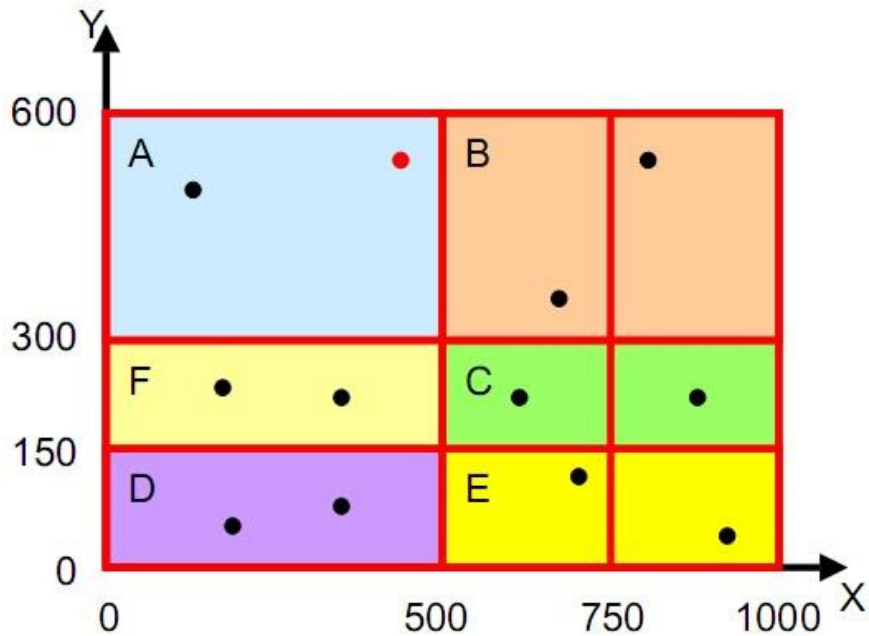
Grid-File (6)



X \ Y	< 500	< 750	< 1000
< 600	A	B	B
< 300	F	C	C
< 150	D	E	E

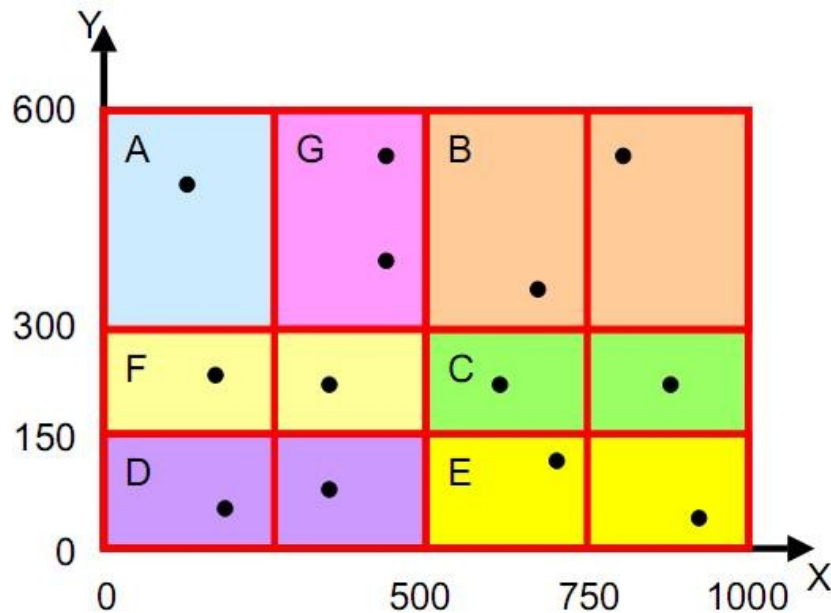
- Insert eines 3. Punktes in Bucket A
-> Überlauf
- -> splitte A nach bereits vorhandener Dimensions-Line in und F

Grid-File (7)



- Insert eines 2.Punktes in A
- -> nichts passiert, da A noch nicht voll gewesen
- Insert 3.Punkt in A
- -> Zellteilung notwendig

Grid-File (8)



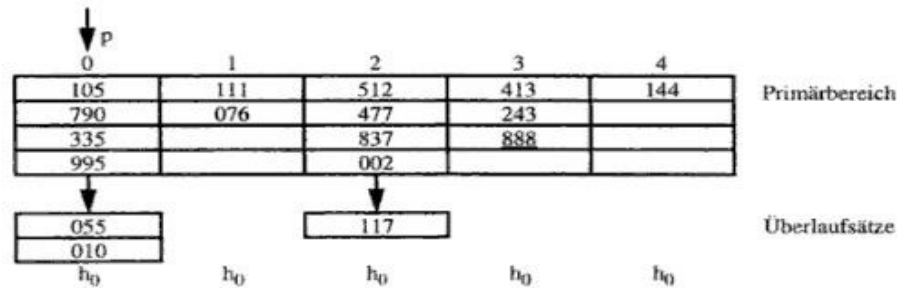
X \ Y	< 250	< 500	< 750	< 1000
< 600	A	G	B	B
< 300	F	F	C	C
< 150	D	D	E	E

- Dimensionsteilung orthogonal zu längerer Kante (um gleichmäßige Bucket zu erhalten)
- Buckets F und D ebenfalls geteilt
- Grid Directory vergrößert (X-scale)

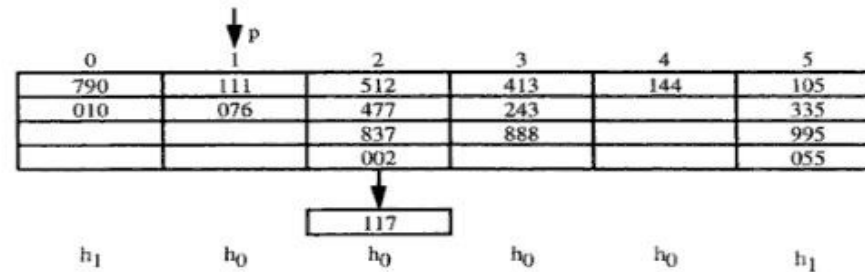
Grid-File (9)

- populärste mehrdimensionale Organisationsform für DBS
- Speicherplatzausnutzung liegt bei ~70%
- 2 Plattenzugriffe bei einfachen Anfragen (1 für Zugriff auf Grid Directory + 1 für Zugriff auf Bucket)
- Problem:
 - Nichtlineares Wachstum des Grid Directorys, teilweise viele leere Zellen
 - Lösung: Interpolationsbasierte Grid-Files, haben im worst case aber schlechter Speicherauslastung in den Buckets

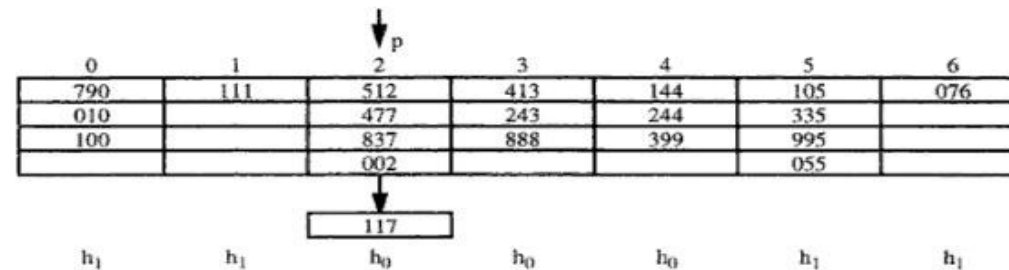
Lineares Hashing



a) Belegung der Hash-Bereichs vor der ersten Erweiterung



b) Belegung nach der Aufteilung von Bucket 0

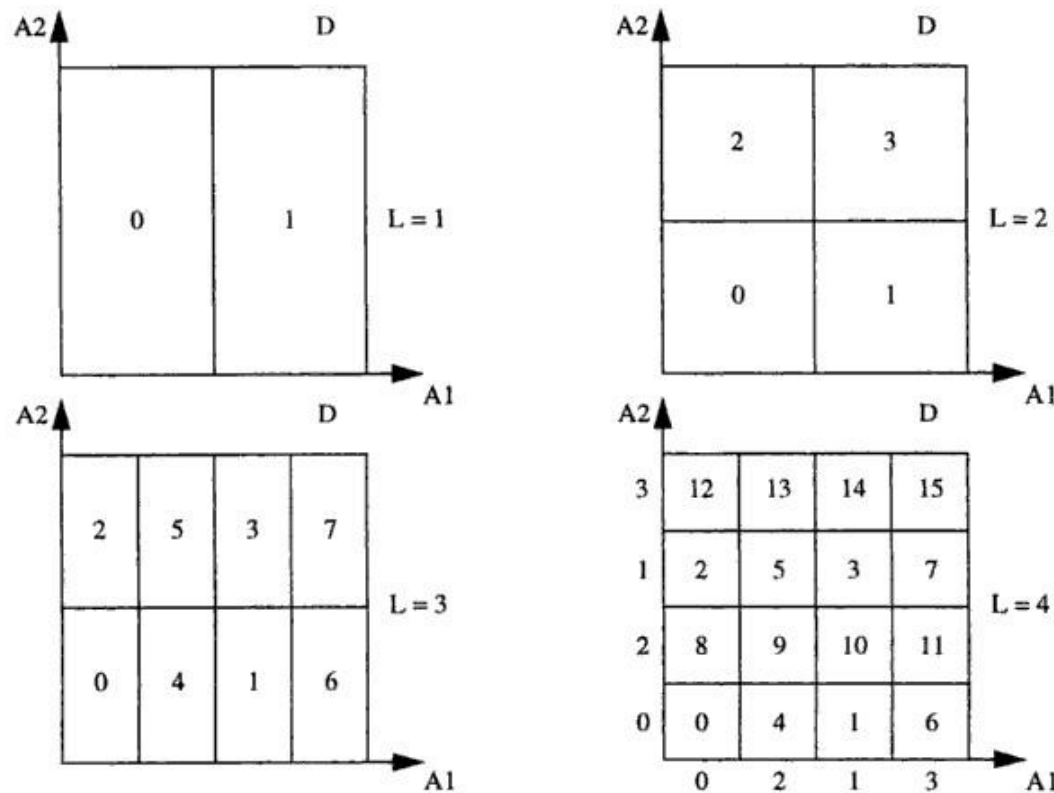


c) Belegung nach weiteren Einfügungen und der Aufteilung von Bucket 1

Mehrdimensionales Hashing

- Entwickelt wegen der Nachteile des Grid-Files:
 - Grid Directory Wachstum
 - 2 Plattenzugriffe falls entsprechende Seite des Grid Directorys nicht im Puffer
- baut auf linearem Hashing auf
- ->benötigt kein Directory

Mehrdimensionales Hashing (2)



- Zuordnung der Zellen zu Buckets durch Adressierungsfunktion

Mehrdimensionales Hashing (3)

- Bei Gleichverteilung dem Grid-File in Bereichsanfragen überlegen, da sehr gute Speicherplatzauslastung
- in Praxis aber nicht durchgesetzt
- Nachteile:
 - komplexe Adressierungsfunktion
 - Anzahl der Verdopplungen der Bucketzahl und Position des Pointers auf aktuelles Bucket
 - Split-History für Verfeinerungsreihenfolge (z.B. „A1, A2, A1, A2“)
 - bei starker Ungleichverteilung sehr schlechte Speicherplatzbelegung / leere Buckets

Quellen

- Bücher

- Datenbanksysteme: Konzepte und Techniken der Implementierung – Theo Härder, Erhard Rahm
- Taschenbuch Datenbanken

- Internet

- wikipedia
- www.ikg.uni-hannover.de/lehre/katalog/gis/gisII_uebung/Gridfile.pdf (Insert-Beispiele)
- <http://www.inf.uni-konstanz.de/dbis/teaching/ws0304/datatypes/download/paper-grid-file.pdf>