

Kapitel 3: Abbildung auf das relationale Datenmodell

Skript zur Vorlesung
Geo-Informationssysteme

Wintersemester 2013/14

Ludwig-Maximilians-Universität München

(c) Peer Kröger 2011, basierend auf dem Skript von Christian Böhm aus dem
SoSe 2009

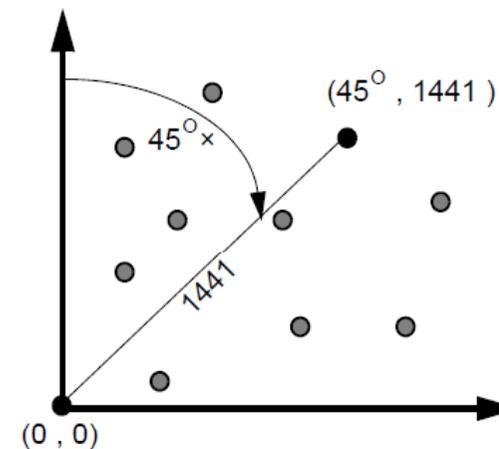
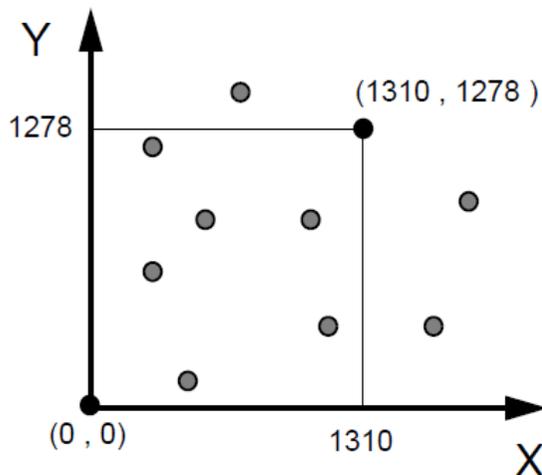


3. Abbildung auf das relationale Datenmodell

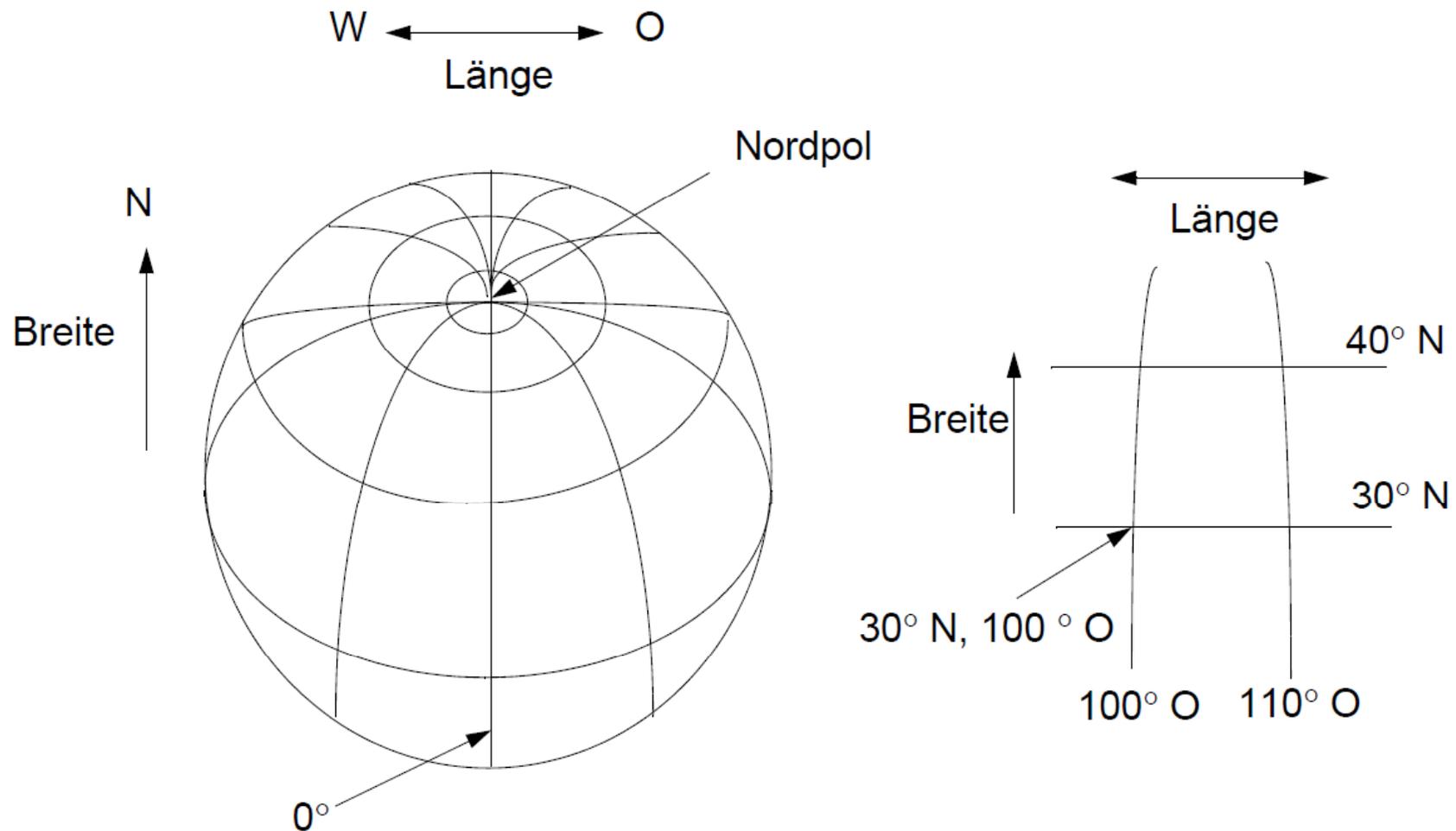
1. Einführung
2. Spaghetti-Modell
3. TIGER-Modell
4. Schwächen des relationalen Modells

3.1 Punkt (I)

- Träger der geometrischen Information, auf dem die anderen Grundelemente aufbauen
- Die Lage eines Punktes im Datenraum wird über *Koordinaten* bezüglich eines *Koordinationsystems* bestimmt
 - Kartesisches Koordinatensystem (Distanzen in Richtung der Koordinatenachsen)
z.B. $x = 1310, y = 1278$
- Polarkoordinatensystem (Winkel und Distanz vom Ursprung)
z.B. $\phi = 45^\circ, r = 1441$

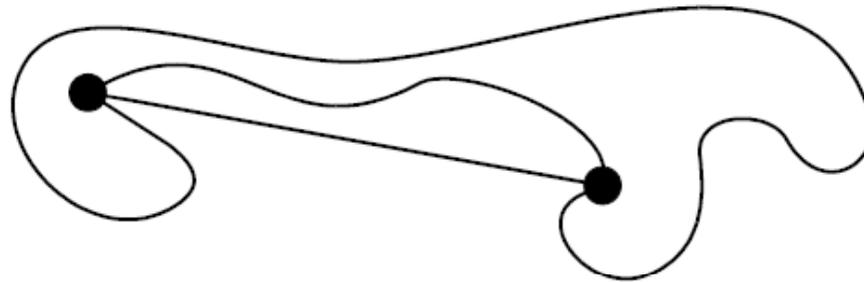


- Geographisches Koordinatensystem (Einteilung in Längen- und Breitengrade)



- Der Datenraum ist ein *metrischer Raum*:
 - Zwischen zwei Punkten P1 und P2 ist eine (nicht-negative) *Distanz d* (Metrik) definiert, so daß
 - $d(P1,P2) = 0 \Leftrightarrow P1 = P2$ (Identität)
 - $d(P1,P2) = d(P2,P1)$ (Symmetrie)
 - $d(P1,P2) \leq d(P1,P3) + d(P3,P2)$ (Dreiecks-Ungleichung)
 - z.B. Euklidische Distanz: $d((x_1,y_1), (x_2,y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
 - z.B. Manhattan Distanz: $d((x_1,y_1), (x_2,y_2)) = |x_1 - x_2| + |y_1 - y_2|$
- Es können weitere vermessungstechnische Informationen zugeordnet sein:
Höhe, Nummer, Art der Erfassung, Genauigkeit, Datum der Erfassung, ...

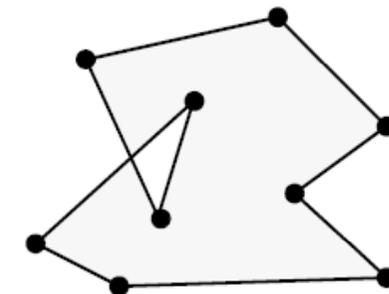
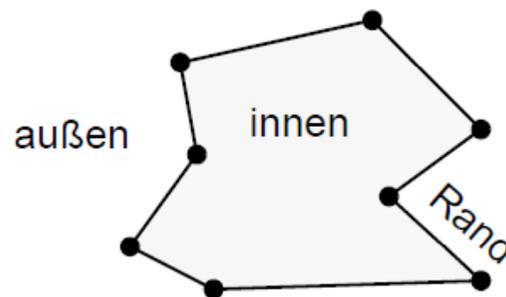
- als topologische Eigenschaft: *Kante*
 - direkte Verbindung zwischen zwei Punkten
 - alle direkten Verbindungen zwischen zwei Punkten sind topologisch äquivalent



- topologisch äquivalent: die relative Lage zweier Objekte zueinander ist gleich
- als geometrische Eigenschaft: *Strecke* oder *Streckenzug*
 - *Strecke*: geradlinige Verbindung zwischen zwei Punkten,
Streckenzug: Folge von Strecken
 - alternativ: Kurven (z.B. Kreisbögen, Bézier-Kurven)

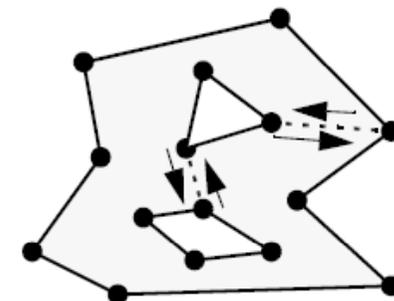
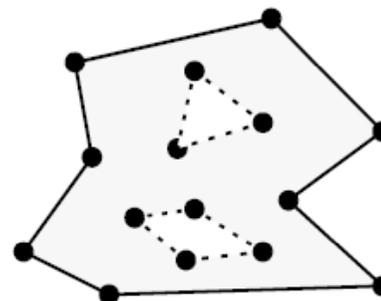
Fläche ohne Löcher

- über geschlossenen Kanten- / Streckenzug definiert
 - *Polygon*: geschlossener Streckenzug
 - *einfaches Polygon*: Strecken sind bis auf die Eckpunkte überlappungsfrei



Fläche mit Löchern

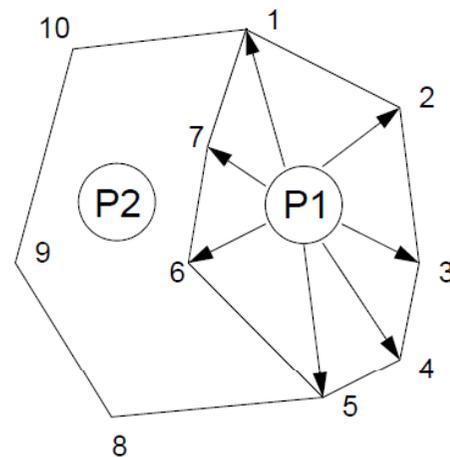
- Modellierungsalternativen:
 - über ein *äußeres Polygon* und 0, 1 oder mehrere innere Polygone
 - mit Hilfe von *Pseudokanten*



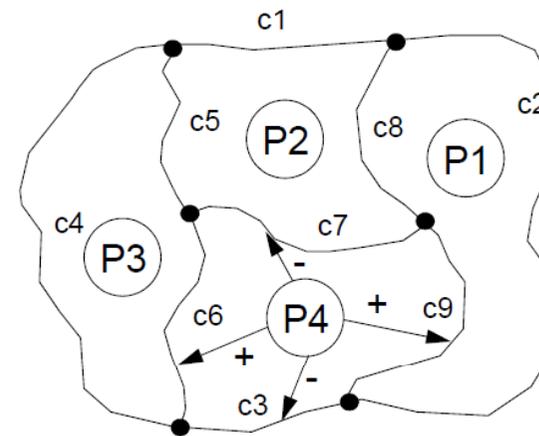
- ist ein einstufiges Vektormodell
- Jedes Objekt wird als Folge ausgezeichneter (x, y) -Koordinaten (z.B. der Eckpunkte) definiert. Die Repräsentation zweier Objekte ist unabhängig voneinander (d.h. gleiche Punkte werden mehrfach gespeichert).
- Objektklassen
 - Punkt
 - Streckenzug
 - einfaches Polygon
- Eigenschaften:
 - **einfach**
 - keine explizite topologische Information (insbesondere keine Löcher in Polygonen explizit modellierbar)
 - starke Redundanz

3.3 Zweistufige Vektormodelle

- mit *Koordinatenreferenzierung*
 - Polygon: geordnetes Verzeichnis von Punktreferenzen
- mit *Streckenzugreferenzierung*
 - Polygon: geordnetes Verzeichnis von Streckenzugreferenzen (Orientierungsinformation notwendig)



Koordinatenreferenzierung



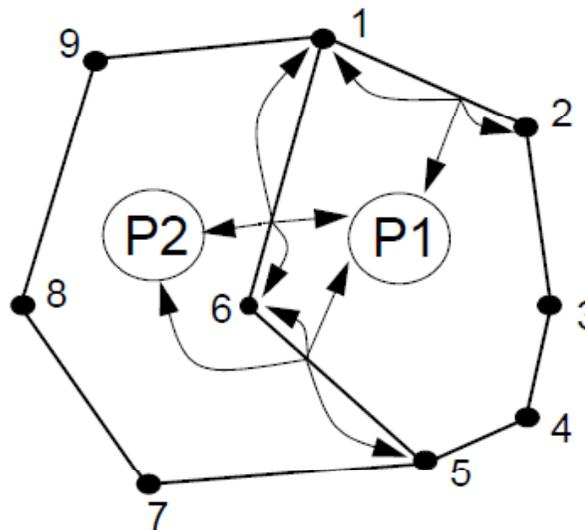
Streckenzugreferenzierung

- Polygon über Verzeichnis repräsentiert (Datenstruktur variabler Größe)
- keine Löcher, keine Nachbarschaftsinformation modelliert

- explizite Speicherung der topologischen Struktur
 - ⇒ einfache Bestimmung von benachbarten Objekten
 - ⇒ höherer Aufwand bei Änderungen
 - ⇒ höherer Speicheraufwand

DIME

- *Dual Independent Map Encoding System (U.S. Bureau of Census, 1970)*
- Grundelement: Kante hat vier Zeiger (Anfangs- und Endpunkt, links und rechts liegende Fläche)



TIGER-Modell

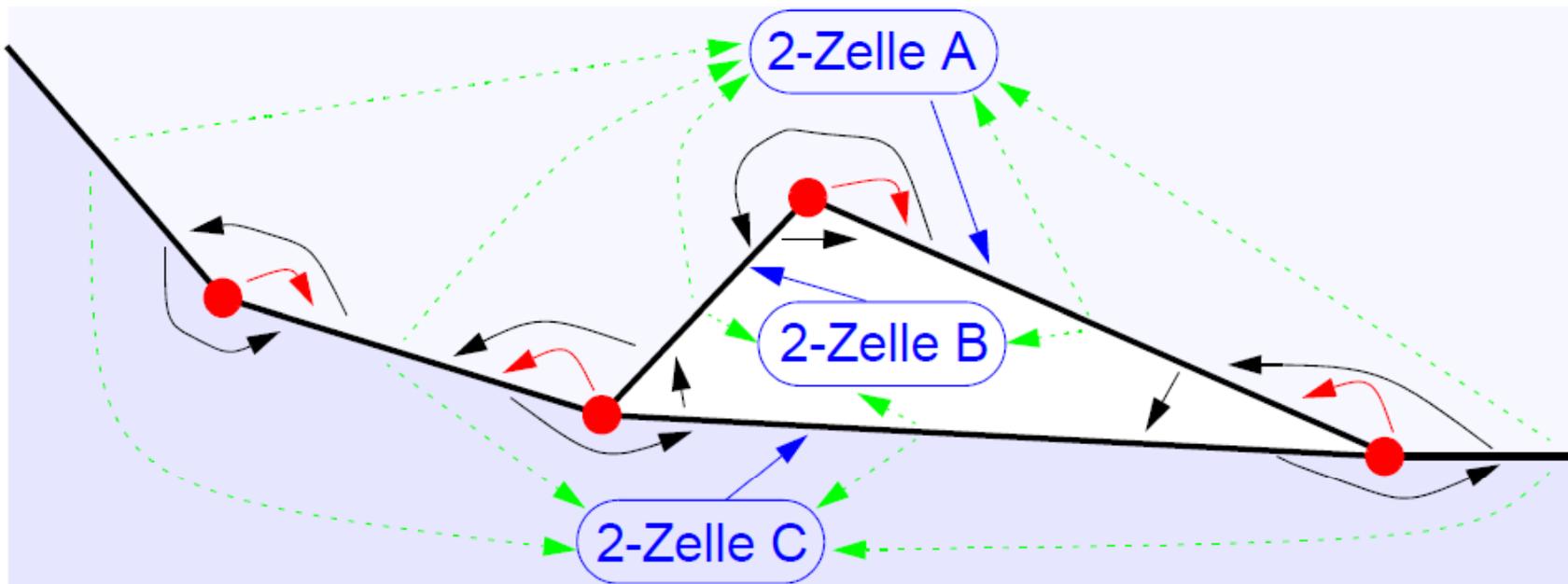
- *Topologically Integrated Geographic Encoding and Referencing Model (Marx, 1986)*
- Datentypen:
 - 0-Zellen: Knoten: Endpunkte von Steckenzügen, Schnittpunkte
 - 1-Zellen: Kante: Verbindung zweier 0-Zellen
 - 2-Zellen: Fläche, die durch Verknüpfung von 1-Zellen beschrieben werden kann (geschlossener Kantenzug)
 - Cover: Vereinigung von 2-Zellen
- Organisation: Listen für 0-, 1- und 2-Zellen, Directories für 0-Zellen und Cover (Datenstrukturen fester Größe)

TIGER-Modell (Fortsetzung)

- 0-Zelle
 - besteht aus ihren Koordinaten, verschiedenen Attributen und *einem* Verweis zu einer 1-Zelle
- 2-Zelle
 - besteht insbesondere aus verschiedenen Attributen und *einem Verweis* zu einer 1-Zelle
- 1-Zelle (Kante)
 - zentrale Struktur: verbindet 0-Zellen und umschließt 2-Zellen
 - besteht insbesondere aus
 - zwei Verweisen auf 0-Zellen (Anfangs- und Endpunkt)
 - zwei Verweisen auf 2-Zellen (linke und rechte Fläche)
 - je *einen* Verweis auf die nächste 1-Zelle, die zu der entsprechenden 2-Zelle gehört
 - einem Verweis auf eine Beschreibungsliste (für interne Punkte)

TIGER-Modell (Fortsetzung)

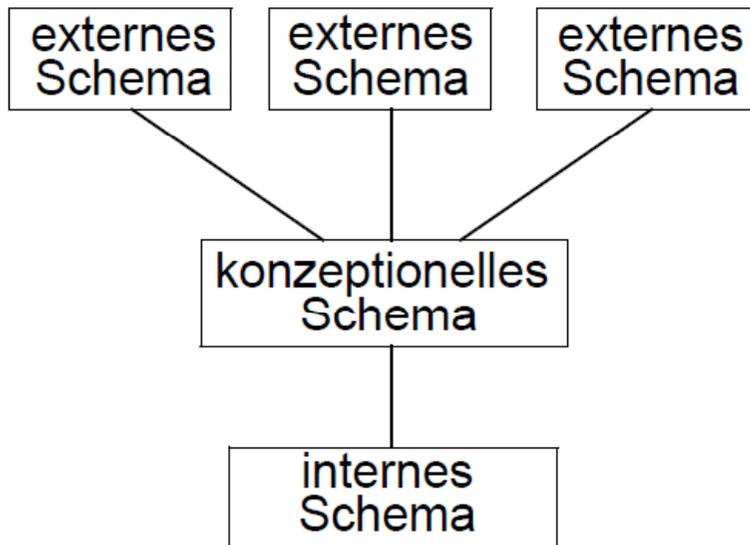
- Beispiel



Äquivalentes Modell (Datenstrukturen variabler Größe)

- Knoten
 - Verweise zu *allen Kanten (umlauforientiert)*
 - Koordinaten (Geometrie)
- Kante (auch geschlossen)
 - zwei Verweise auf Eckknoten
 - zwei Verweise auf benachbarte Flächen
 - Verweis auf eine Beschreibungsliste (für interne Punkte, Geometrie)
- Fläche (mit Löchern)
 - Verweise zu *allen umgebenden Kanten (ablauforientiert)*
 - Verweise zu *allen eingeschlossenen Kanten (Menge, ablauforientiert)*
 - Verweis auf Cover
- Cover
 - Verweise auf Flächen

Ebenen eines DBS



unterschiedliche Sichten verschiedener Benutzer / Anwenderprogramme auf die Daten

logische Gesamtsicht aller Daten

physische Datenorganisation

- externe und konzeptionelle Ebene
 - logisches Datenmodell
 - Anfragesprache
- interne Ebene
 - physische Datenrepräsentation
 - Anfragebearbeitung und -optimierung

Schema für mehrstufiges Vektormodell

Parzellen

<u>OID</u>	PolID	Q-Meter
Par1	Pol1	1200
Par2	Pol2	1435

Polygone

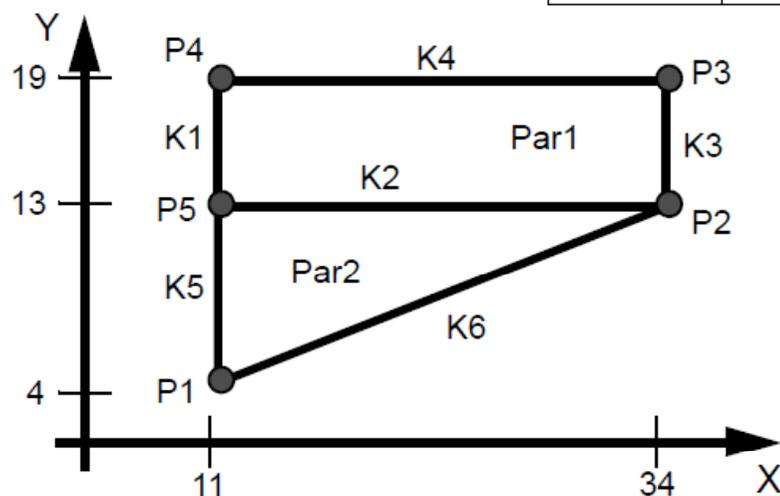
<u>PolID</u>	<u>KID</u>
Pol1	K1
Pol1	K2
Pol1	K3
Pol1	K4
Pol2	K2
Pol2	K5
Pol2	K6

Kanten

<u>KID</u>	Von	Nach
K1	P4	P5
K2	P5	P2
K3	P2	P3
K4	P3	P4
K5	P5	P1
K6	P1	P2

Punkte

<u>PID</u>	x	y
P1	11	4
P2	34	13
P3	34	19
P4	11	19
P5	11	13



Anfragen an dieses Schema

Bestimme alle Koordinaten der Parzelle Par1:

```
SELECT Punkte.x, Punkte.y
FROM Parzellen, Polygone, Kanten, Punkte
WHERE (Parzellen.OID=Par1) AND
      (Parzellen.PolID = Polygone.PolID) AND
      (Polygone.KID = Kanten.KID) AND
      (Kanten.Von = Punkte.PID);
```

Probleme

- hoher Aufwand für Zugriff auf alle Komponenten eines Polygons (Joins)
- Antwort auf Anfrage ist unsortiert (kein "Objekt" Polygon)

⇒ Objektorientiertes Datenmodell

Objekte

- Objekte sind vom Benutzer definierte Entities, die aus Werten oder anderen Objekten aufgebaut sind (Attribute).
- Attribute können listen- oder mengenwertig sein.

Objekttypen

- Ein Objekttyp ("Klasse") beschreibt die Charakteristika einer Menge ähnlicher Objekte.
- Ein Objekt ist eine konkrete Instanz seines Objekttyps, hat also dessen Charakteristika.

- Beispiel:

```
type Vertex is
```

```
    public . . .
```

```
    body [x, y, z: float;]
```

```
end type Vertex;
```

```
type Cuboid is
```

```
    public . . .
```

```
    body [mat: Material;
```

```
        value: float;
```

```
        vertices: <Vertex>;]// Typ: Liste von Vertices
```

```
end type Cuboid;
```

Operationen

- Jeder Objekttyp bietet an seiner Benutzerschnittstelle eine Menge von Operationen an, mit deren Hilfe Instanzen dieses Typs manipuliert werden können.
- Neben diesen Operationen gibt es keine andere Möglichkeit, von außen den internen Zustand eines Objekts abzufragen oder zu ändern.

- Beispiel:

type Vertex is

public translate, scale, rotate, distance, inOrigin, . . . //

Benutzerschnittstelle

body [x, y, z: float;]

operations

declare translate: Vertex → void; // Änderungsfunktion

declare scale: Vertex → void; // Änderungsfunktion

declare rotate: float, char → void; // Änderungsfunktion

declare distance: Vertex → float; // Beobachterfunktion

declare inOrigin: → bool; // Beobachterfunktion

. . .

implementation

. . .

end type Vertex;

Modellierung von Spatial Data Types

```
type Vertex is
  public distance, contained_in, ...
  body [x, y: float;]
  ...
end type Vertex;
```

```
type Polygon is
  public intersect, ...
  body [boundary: <Edge>;
  ...]
end type Polygon;
```

```
type Edge is
  public ...
  body [begin, end: Vertex;]
  ...
end type Edge;
```

```
type Region is // Parzelle
  public ...
  body [name: string;
  extension: Polygon; area: float;]
end type Region;
```

- Bestimme alle Koordinaten der Parzelle Par1:

```
select x, y  
from (select begin  
  from (select r.extension.boundary  
    from r in Region  
    where r.name = "Par1"))
```

Vorteile

- Spatial Data Types können definiert werden, die in Attributen referenziert werden können.
- Die Modellierung von Eigenschaften variabler Kardinalität ist einfach mit Hilfe listenbzw. mengenwertiger Attribute.
- Logisch zusammengehörige Objekte werden physisch benachbart gespeichert (Clustering), so daß das Einlesen / Ändern eines Geo-Objektes relativ effizient möglich ist.
- In den Anfragen können die Operationen der benutzerdefinierten Objekttypen verwendet werden.
 - ⇒ Geo-Objekte als Objekte modellierbar
 - ⇒ effizienter Zugriff auf ein Geo-Objekt

Schwächen

- Zugriff auf mehrere benachbarte Geo-Objekte ist noch nicht effizient unterstützt (Erweiterung analog TIGER-Modell möglich)
- Unterstützung durch ausgereiftes DBMS oft schwierig

=> Schwächen auf konzeptioneller Ebene nicht (so einfach) durch oo Datenmodell behebbar

Kompromiss

- Die meisten (kommerziellen) relationalen DBMS bieten *objektrelationale* Erweiterungen an (u.a. auch mit gewissen Einschränkungen benutzerdefinierte Datentypen)

Aufgaben der internen Ebene

- physische Datenrepräsentation
- Anfragebearbeitung und –optimierung
- ...

Ziele

- dauerhafte (persistente) Speicherung der Daten
- effiziente Bearbeitung von thematischen und geometrischen Anfragen, von Kombinationen, von topologischen Operationen usw.
- Kriterien für Effizienz:
 - Antwortzeit für einzelne Anfragen
 - Durchsatz über alle Anfragen
- dynamisches Einfügen, Löschen und Verändern von Daten
- hohe Speicherplatzausnutzung

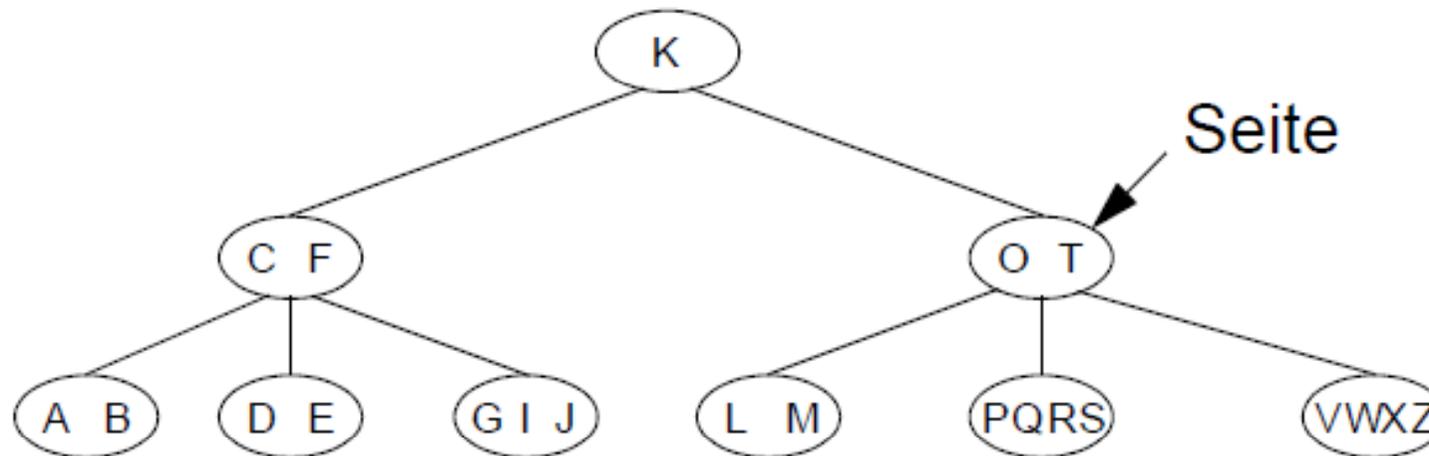
Physische Datenspeicherung

- seitenorientiert
 - Seite: Transfereinheit zwischen Haupt- und Sekundärspeicher
 - Wahlfreier (direkter) Zugriff
 - Feste Größe: zwischen 512 Byte und 8 Kbyte
 - Zugriff über einen Kamm mit Schreib-/Leseköpfen, der quer zu sich rotierenden Magnetplatten bewegt wird
 - Positionierung des Schreib-/Lesekopfes [6 msec]
 - Warten auf Seite [4 msec]
 - Kontrolle + Übertragung der Seite [1 msec + 0,2 msec / 2KByte Seite]
 - Entwicklung:
 - Kapazität von Plattenspeichern erhöht sich drastisch
 - Zugriffszeiten sinken relativ langsam
- ⇒ Zugriffe auf Sekundärspeicher sind verglichen mit Hauptspeicheroperationen sehr zeitaufwendig; Minimierung der Anzahl der Seitenzugriffe

B-Bäume

- Ein Baum heißt *Suchbaum*, wenn für jeden Eintrag in einem inneren Knoten alle Schlüssel im linken Teilbaum kleiner sind und alle Schlüssel im rechten Teilbaum größer sind als der Schlüssel im Knoten.
- Ein *B-Baum der Ordnung m* ist ein Suchbaum mit folgenden Eigenschaften:
 - (1) Jeder Knoten enthält höchstens $2m$ Schlüssel.
 - (2) Jeder Knoten außer der Wurzel enthält mindestens m Schlüssel.
 - (3) Die Wurzel enthält mindestens einen Schlüssel.
 - (4) Ein Knoten mit k Schlüssel hat genau $k+1$ Söhne.
 - (5) Alle Blätter befinden sich auf demselben Level.

- Beispiel: B-Baum der Ordnung 2



- Höhe eines B-Baums für n Schlüssel ist $O(\log n)$

Leistung

- Einfügen, Löschen und Punktanfrage im B-Baum sind auf einen Pfad beschränkt und damit in $O(\log n)$ Laufzeitkomplexität

Weitere Ziele

- *Sequentielles Auslesen aller Datensätze*, die von einem B-Baum organisiert werden.
- *Unterstützung von Bereichsanfragen* der Form:
“Nenne mir alle Studenten, deren Nachname im Bereich [Be ... Brz] liegt.”

Idee

- Trennung der Indexstruktur in *Directoryknoten* (innere Knoten) und Datenknoten (Blätter).
- *Sequentielle Verkettung* der Datenknoten.

B+-Baum

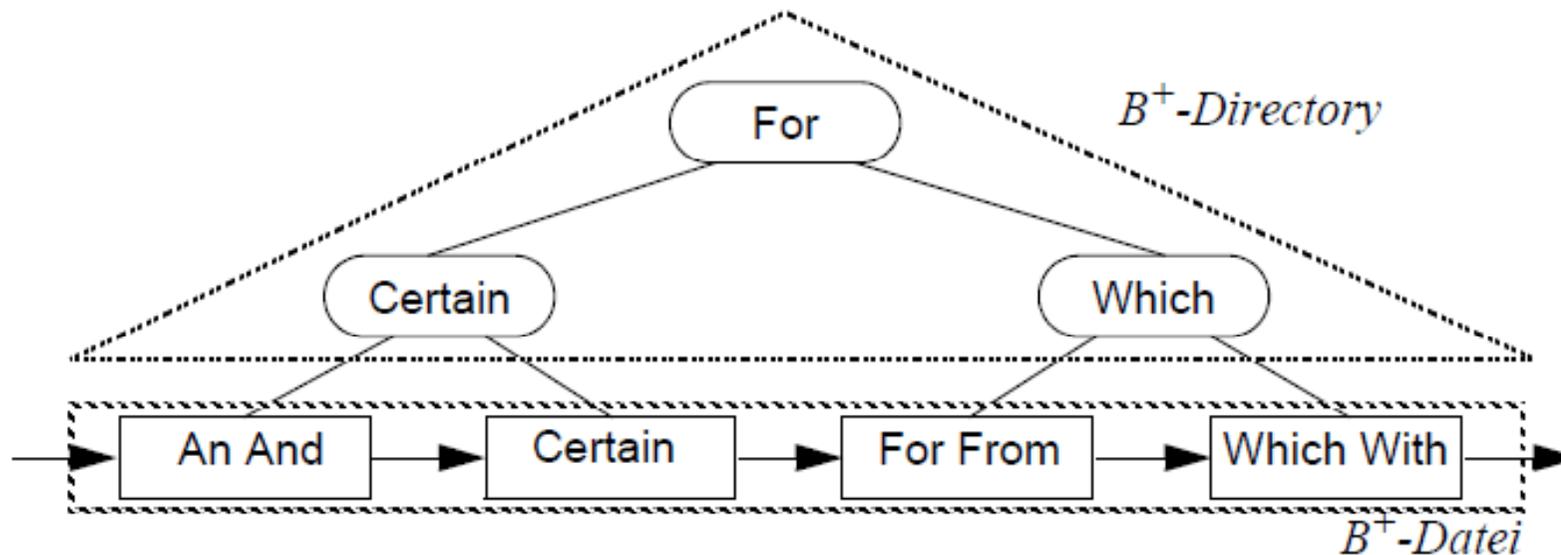
- Die Datenknoten enthalten alle Datensätze.
- Directoryknoten enthalten nur noch *Separatoren* s .

Separatoreigenschaft

- Für jeden Separator $s(u)$ eines Knotens u gelten folgende *Separatoreneigenschaften*:
 - $s(u) > s(v)$ für alle Directoryknoten v im linken Teilbaum von $s(u)$.
 - $s(u) < s(w)$ für alle Directoryknoten w im rechten Teilbaum von $s(u)$.
 - $s(u) > k(v')$ für alle Primärschlüssel $k(v')$ und alle Datenknoten v' im linken Teilbaum von $s(u)$.
 - $s(u) \leq k(w')$ für alle Primärschlüssel $k(w')$ und alle Datenknoten w' im rechten Teilbaum von $s(u)$.

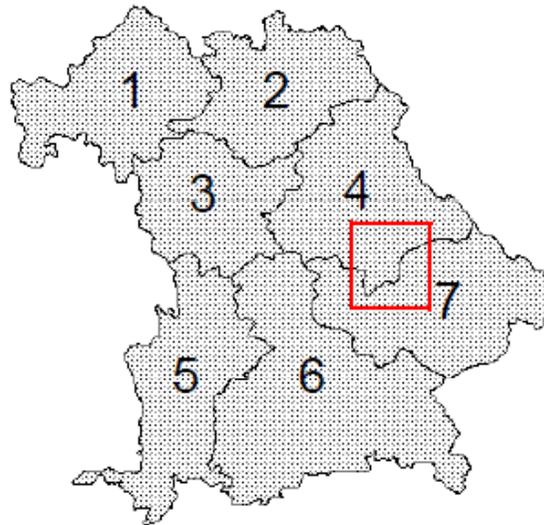
Separatoreigenschaft

- Beispiel: **B+**-Baum für Zeichenketten



B-Bäume für Geo-Objekte

- Lineare Ordnung des 2D Raums nötig
- Beispiel



- im 2D benachbarte Objekte werden häufig miteinander angefragt
- es gibt keine lineare Ordnung, die alle Nachbarschaften des 2D-Raums erhält
⇒ die Antworten auf eine Fensteranfrage sind über den ganzen Index verteilt

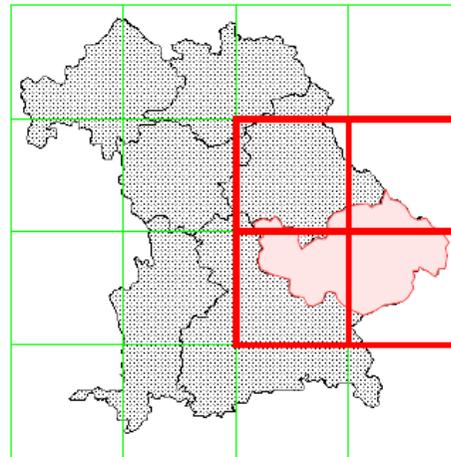
Hashverfahren

- Zuordnung der Objekte zu Einträgen einer Hashtabelle

0	1	2	3	4	5	6	7
	9 25		3	28	5	14	→ Seite

Hashfunktion: Schlüssel MOD 8

- Voraussetzung: Definition einer Hashfunktion mit möglichst wenigen Kollisionen



⇒ im 2D benachbarte Objekte auf dieselbe bzw. benachbarte Adresse abbilden

⇒ ähnliche Probleme wie bei B-Bäumen

Zusammenfassung

- Schwächen auf konzeptioneller Ebene
 - Mächtigkeit des relationalen Modells begrenzt
 - Logisches Schema oft sehr komplex
 - Kein Objektbegriff, Information über einzelne Objekte auf viele Tabellen verteilt
 - Komplexe Anfragen
- Schwächen auf interner Ebene
 - Datenstrukturen zur physischen Anfragebearbeitung (Indexstrukturen) sind für 1D Daten entworfen (Primärschlüsselsuche)
 - Räumliche Anfragen auf 2D/3D Geo-Objekte
 - Abbildung auf 1D (Linearisierung) führt zu schlechtem Antwortverhalten
 - Nachbarschaftsbeziehungen können nicht vollständig erhalten werden
 - Antworten können über den ganzen Index verteilt sein