Räumliche Datenbanken:

Oracle Spatial

Autor: Thomas Friedrich
97 I

Oberseminar Datenbanken
Prof. Kudrass

Im vorliegenden Dokument werden räumliche Datenbanken behandelt, die die Abspeicherung und Verarbeitung von geometrischen Daten erleichtern. Dabei wird speziell auf "Oracle Spatial" eingegangen. Beide Arten der Repräsentation von räumlichen Daten, die in diesem Produkt enthalten sind, werden aufgezeigt und die Anwendung anhand von Beispielen erläutert. Außerdem wird auf Indizierungsmethoden eingegangen, die sich für räumliche Daten, wie z.B. Punkt- oder Bereichsdaten, eignen. Zuletzt werden einige Merkmale von SQL/MM Spatial erläutert.

Thomas Friedrich

1. Einführung

Räumliche Daten repräsentieren die wesentlichen geometrischen Charakteristiken von Objekten, und damit ihre Beziehung zu dem Raum, in dem sie existieren.
Beispiel Straßenkarte: die Karte ist ein zweidimensionales Objekt, welches Punkte, Linien und Polygone enthält und die z.B. Staatsgrenzen, Städte und Straßen repräsentieren. Eine Straßenkarte ist also die Visualisierung von geographischen Informationen. Die Lage von Städten, Straßen usw. auf der Erde wird auf eine zweidimensionale Fläche unter Beibehaltung ihrer relativen Positionen und Entfernungen projiziert.
Die Daten, die die Länge und Breite oder Höhe und Tiefe bestimmen, sind die räumlichen Daten.

Anwendungen von räumlichen Daten sind:

- Geographic Information Systems (GIS) bei geographischen Daten
- CAD/CAM z.B. bei Herstellung von Motoren

Der Unterschied in der Anwendung liegt nur im Maßstab und nicht in der Komplexität: die Lage einer Brücke kann um Zentimeter abweichen ohne Auswirkung auf den Straßenbau zu haben, aber im Motorenbau muß es viel genauer sein

Alle Beispiele haben gemeinsam, daß räumliche und nichträumliche Daten, wie z.B. Namen, Landschaftsnutzung und Teilnummer, gemeinsam in der Datenbank gespeichert werden und Operationen darauf ausgeführt werden. Das räumliche Attribut eines Objekts, bezeichnet als seine Geometrie, ist eine geordnete Folge von Scheitelpunkten, die durch gerade Linien oder Bögen miteinander verbunden sind. Die Semantik der Geometrie wird durch den Objekttyp bestimmt, z.B. Punkt, Linie oder Polygon.

2.Oracle Spatial

2.1 Beschreibung des Spatial-Produkts

Oracle Spatial, kurz Spatial genannt, soll die Verarbeitung und Handhabung räumlicher Daten für den Nutzer und Anwendungen wie Geographische Informationssysteme (GIS) erleichtern. Es ist eine Sammlung an Funktionen und Prozeduren, die es ermöglichen, räumliche Daten zu speichern, zurückzugewinnen, zu updaten und Abfragen darauf auszuführen.
Es besteht aus 4 Komponenten:

1. Schema der Beschreibung der Speicherung, Syntax und Semantik der unterstützten Datentypen
2. Räumliche Indizierungsmechanismen
3. Sammlung von Operatoren und Funktionen um Spatial-Join- und *area-of-interest*-Queries auszuführen
4. Administrative Tools

Das Release 8.1.5 unterstützt zwei Varianten der Darstellung der Geometrie von Objekten:

1. Objekt-Relationales Schema: in der Tabelle wird eine Spalte vom Typ MDSYS.SDO_Geometry benutzt und ein Objekt in einer Zeile gespeichert
2. Relationales Schema: benutzt vordefinierte Anzahl von Spalten des Typs NUMBER und eine oder mehr Zeilen pro Objekt

Die beiden Methoden korrespondieren ungefähr zu den 2 Alternativen, die in der "OpenGIS ODBC/SQL" –Spezifikation beschrieben sind. Das Objekt-Relationale Schema bezieht sich

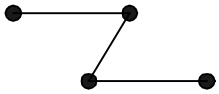
auf die Implementierung von räumlichen Möglichkeiten mittels "SQL with Geometry Types", während sich das Relationale Modell auf die Implementierung mittels numerischen Typen bezieht.

2.2 Geometrische Typen

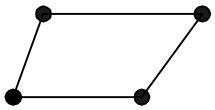
Folgende 3 primitive Datentypen und Zusammensetzungen daraus, werden in beiden Implementierungsmethoden von Spatial unterstützt.

1. Point: x,y – Koordinaten

2. Line String: ein oder mehrere zusammengesetzte Paare von Punkten, die ein Liniensegment definieren



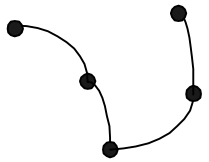
3. Polygon: zusammengesetzt aus verbundenen LineStrings, die einen geschlossenen Ring bilden und die Innenfläche einschließen



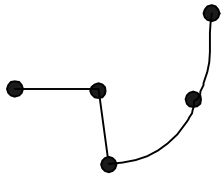
Schneidende Line Strings werden unterstützt, wobei dabei kein Polygon entsteht, es also keine Innenfläche hat.

Das Objekt-Relationale Modell unterstützt zusätzlich folgende Typen:

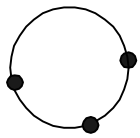
1. 2D-Arc Line-String



2. 2D-Arc Polygon
3. 2D-Compound Polygon
4. 2D- Compound Line String



5. 2D-Circles



6. 2D-Optimized Rectangles

2.3 Datenmodelle

Das Datenmodell von Spatial ist eine hierarchische Struktur, die aus Elementen, Geometrie und Layer besteht. Layer ist zusammengesetzt aus Geometrien, und diese wiederum aus Elementen.

- **Element**: Elemente sind die Basisblöcke einer Geometrie. Die unterstützten räumlichen Elementtypen sind Point, Line String und Polygon. Elemente können z.B. Sternenkonstellationen (Punktmenge), Straßen (Line String) oder Grenzen (Polygon) modellieren. Jede Koordinate eines Elements ist mit seinen x,y-Werten gespeichert. Punkt-Daten bestehen nur aus einer Koordinate, Line Strings aus zwei und Polygone aus Koordinaten-Paar-Werten mit einem Scheitelpunkt für jedes Liniensegment. Die Koordinaten werden entweder im oder entgegen dem Uhrzeigersinn beschrieben.

- **Geometrie**: Geometrie ist eine geordnete Anzahl an primitiven Elementen, die ein räumliches Objekt bilden.
  Beispielsweise könnte Bauland in einer Stadt durch Polygon mit Löchern beschrieben werden, bei denen Löcher z.B. Wasser oder andere, den Bau verhindernde Gebiete, darstellen.
  Eine Geometrie kann aus einem einzelnen Element bestehen, welches eine Instanz eines der unterstützten Primitivtypen ist. Es kann aber auch eine Menge von Elementen sein: homogene Menge (nur Elemente eines Typs), heterogene Menge (Elemente unterschiedlichen Typs). Im relationalen Modell muß jede Geometrie eine eindeutige GID (*geometry identifier*) haben, über die die einzelnen Elemente des Objektes in der Datenbank verbunden werden. Dies ist im Objekt-Relationalen nicht nötig.

- **Layer**: Layer ist eine heterogene Menge von Geometrien mit gleichen Attributen. Beispielsweise könnte es einen Layer in einem GIS geben, der nur topographische Daten enthält, ein anderer die Bevölkerungsdichten. Die Geometrie eines jeden Layers und ihr *Spatial index* sind in Standardtabellen der Datenbank gespeichert.

2.4 Anfragemodelle

Spatial nutzt ein zweistufiges Anfragemodell um Queries und Joins zu bearbeiten. Es werden 2 verschiedene Operationen ausgeführt und die Ausgabe am Ende enthält das exakte Ergebnis. Die beiden Operationen werden als *primary* und *secondary filter operations* bezeichnet.

- **primary filter:** Erlaubt schnelle Selektion aller Kandidaten-Records, die an den secondary filter weitergegeben werden. Er vergleicht nur geometrische Näherungswerte um den Rechenaufwand zu reduzieren. Das Ergebnis liefert eine Obermenge der Ergebnismenge.
- **secondary filter:** Wendet exakte Berechnung auf das Resultset vom primary filter an und liefert das genaue Ergebnis. Der Filter benötigt mehr Rechenaufwand, aber die genaue Berechnung wird nicht mehr auf alle Records angewandt, sondern nur auf eine eingeschränkte Menge.

Spatial nutzt einen linearen Quadtree-basierten Spatial-Index, um den primary filter zu implementieren, d.h beim Primärfilter wird nur auf der Indextabelle gearbeitet.
Secondary filter ist für die Bestimmung der räumlichen Beziehung zwischen Objekten, basierend auf der geometrischen Lage, zuständig. Die meisten Beziehungen basieren auf Topologie und Entfernung.

Beispiel: Die Grenze einer Fläche besteht aus einer Menge von Kurven, die die Fläche vom Rest abgrenzt. Die Innenfläche besteht aus allen Punkten der Fläche, die nicht auf der Grenze liegen. 2 Flächen sind angrenzend, wenn sie Teile der Grenze teilen, aber keinen Punkt im Inneren. Die Entfernung zwischen 2 Objekten ist das Minimum der Entfernung zwischen zwei beliebigen Punkten beider Objekte. 2 Objekte befinden sich innerhalb einer Distanz, wenn ihre Entfernung kleiner als die gegebene Distanz ist.

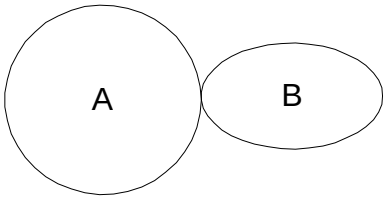
Spatial enthält zwei Funktionen, die den *secondary filter* implementieren.

1. **SDO_GEOM.RELATE()**: Diese Funktion wertet topologische Kriterien aus. RELATE implementiert ein *9-intersection-model*, um die topologischen Beziehungen zwischen Punkten, Linien und Polygonen zu kategorisieren. Jedes räumliche Objekt hat ein Inneres, Äußeres und eine Grenze.
   3 Komponenten:
   - interior: Ai
   - exterior: Ae
   - boundary: Ab

   Jedes Objekt hat 9 verschieden Möglichkeiten, mit einem anderen Objekt zu interagieren.

		B		
		b	i	e
A	b	1	0	1
	i	0	0	1
	e	1	1	1

9-intersection matrix



A TOUCH B

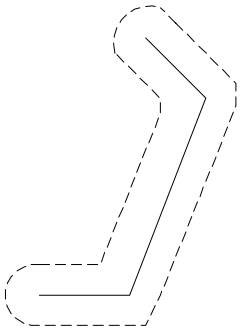
Einige mögliche Beziehungen sind:

- Disjoint: Grenze und Inneres interagieren nicht
- Touch: nur Grenzen interagieren miteinander
- Contains: Inneres und Grenze eines Objektes sind komplett im Inneren eines anderen Objektes

- Inside: Gegenteil von Contains (A inside B bedeutet B contains A)
- Equal: Inneres und Grenzen zweier Objekte sind gleich

2. **SDO_GEOM.WITHIN_DISTANCE**: bestimmt, ob Objekte innerhalb einer gegebenen Distanz liegen. Zuerst wird ein Entfernungspuffer um Objekt B erstellt und dann geschaut, ob Objekt A und der Puffer nicht disjunkt sind, also Punkte miteinander teilen.

   Bsp-Puffer:



Nicht immer sind beide Filter nötig. Wenn z.B. der Zoom auf einer Karte benötigt wird, liefert der Primärfilter schnell eine Obermenge. Die Anwendung kann dann durch Clipping-Operationen die Zielfläche darstellen. Zweck des Primärfilters ist die schnelle Erstellung einer Untermenge aller Daten, um die Rechenbelastung im Sekundärfilter zu verringern. Deswegen muß er so schnell wie möglich sein, was durch die Charakteristik des spatial index gewährleistet ist.

2.5 Indizierungsvarianten bei räumlichen Datentypen

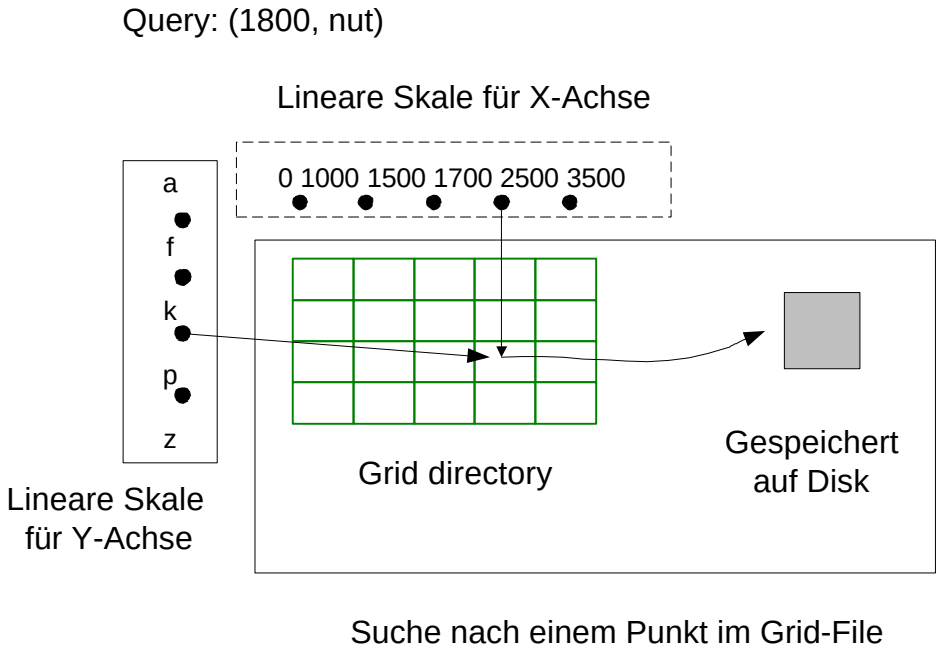
Um den Zugriff auf *spatial data* in der Datenbank zu beschleunigen, werden Indizes benötigt. Ein spatial Index verwendet Arten der räumlichen Beziehung um Dateneinträge zu verwalten, wobei jeder Schlüsselwert als Punkt (oder Region bei Bereichsdaten) in einem k-dimensionalen Raum gesehen wird. k ist die Anzahl der Felder im Suchschlüssel des Index.
Es gibt viele Arten an Strukturen für einen spatial index, wobei einige für die Benutzung von Punkten entwickelt worden, aber auch auf die Benutzung mit Region-Daten angepasst werden können, und andere, die für Bereichsdaten vorgesehen sind.

Ein Beispiel für eine Indexstruktur für Punktdaten ist das Gridfile. Für Bereichsdaten eignen sich z.B. R-Tree und 'Region Quad Tree'.
Es gibt aber keinen allgemeingültigen besten Index, wobei der R-Tree am weitesten verbreitet ist.

2.5.1 Grid-File

Das Grid-File teilt den Raum auf eine Weise, die die Datenverteilung im Datenbereich reflektiert. Es ist entwickelt worden, damit jede beliebige Anfrage, die sich auf eine Punkt bezieht, bei beliebig großen Datenbeständen in 2 Diskoperationen ausgeführt werden kann.
Die Struktur des Grid-File soll im zweidimensionalen Raum gezeigt werden. Das Grid-File teilt den Raum in rechteckige Flächen mit Kanten, die parallel zu den Achsen liegen. Die Einteilung der Achsen in Bereiche geschieht in einem Feld *linear scale*, wobei für jede Achse ein eindimensionales Feld enthalten ist. Außerdem hat das Grid-File ein Grid-Directory, in

dem für jeden Bereich ein Eintrag zur Identifikation der *data page*, in der der Punkt enthalten ist, gespeichert ist.



Suche nach einem Punkt im Grid-File

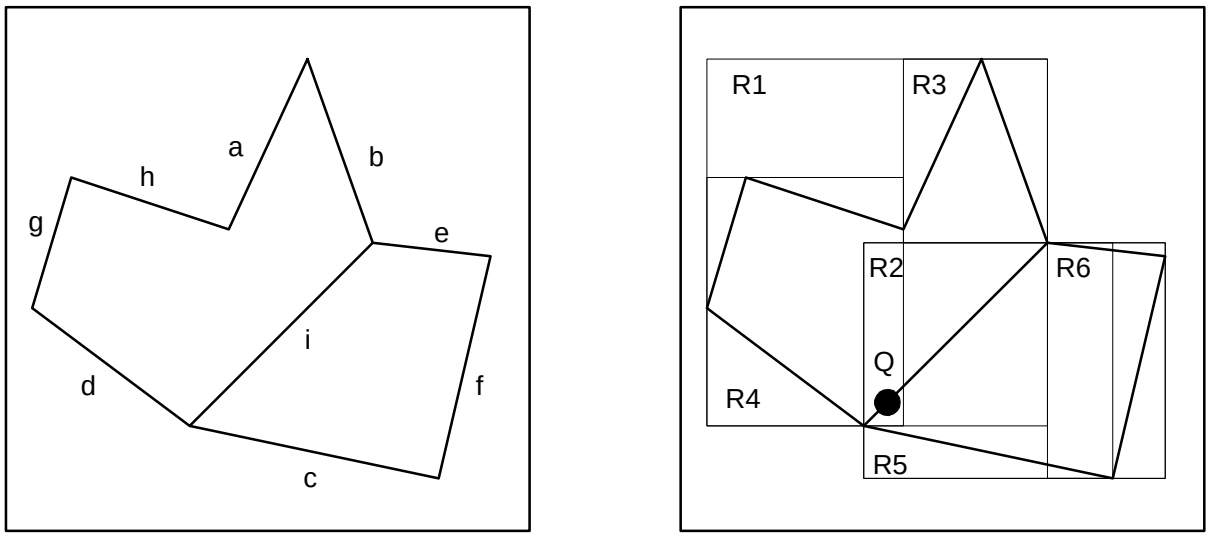
Wenn man nach einem Punkt sucht, wird zuerst auf der X-Achse der Bereich gesucht, in dem der X-Wert des Punktes enthalten ist. Das selbe wird dem Y-Wert auf der Y-Achse gemacht. Beide Werte identifizieren den Eintrag im grid-directory, und der Punkt kann geladen werden. Wenn man annehmen kann, daß die *linear scales* im Hauptspeicher gehalten werden, wird hierfür keine I/O-Operation benötigt. Da das grid-directory in der Regel nicht in den Speicher paßt, benötigt man eine I/O-Operation, um den relevanten Bereich in den Speicher zu laden. Mit der aus dem directory entnommenen ID der Seite der *data page*, die den Punkt enthält, kann der Punkt mit einer weiteren I/O-Operation geladen werden.
Range-Queries und Nachbarschafts-Anfragen können schnell beantwortet werden. Für Range-Queries werden die linearen Achsen genutzt, um die relevanten Bereiche zu finden. Für Nachbarschaftsanfragen zu einem bestimmten Punkt wird einfach die Datenseite des Punktes nach weiteren Punkten durchsucht. Ist diese leer, werden die angrenzenden Bereiche auf den Achsen genommen, um die umliegenden Bereiche zu untersuchen.

2.5.2 R-Tree

Der R-Tree ist eine Abwandlung des B+Baumes und ist eine Indexstruktur für Punkte und Bereiche. Der Baum ist höhenbalanciert. Der Suchschlüssel ist eine Menge von Intervallen, wobei für jede Dimension ein Intervall angegeben wird. Der Suchschlüssel ist also eine Box, mit Kanten parallel zu den Achsen. Diese wird als *bounding box* bezeichnet.
Jeder Dateneintrag besteht aus einem Tupel <n-dimensionale Box, rID>, wobei die Box das kleinste, das Objekt umschließende Rechteck ist, und rID das Objekt identifiziert. Im speziellen Fall, daß das Objekt ein Punkt ist, ist die Box auch ein Punkt statt eine Region. Die Dateneinträge sind in den Blättern gespeichert. Nicht-Blattknoten enthalten Indexeinträge der Form <n-dimensionale Box, Pointer zu Sohnknoten>. Die Box in solch einem Knoten ist die kleinste Box, die alle Boxen der Söhne enthält, d.h. es begrenzt die Fläche, in der alle Objekte im Unterbaum des Knotens enthalten sind.

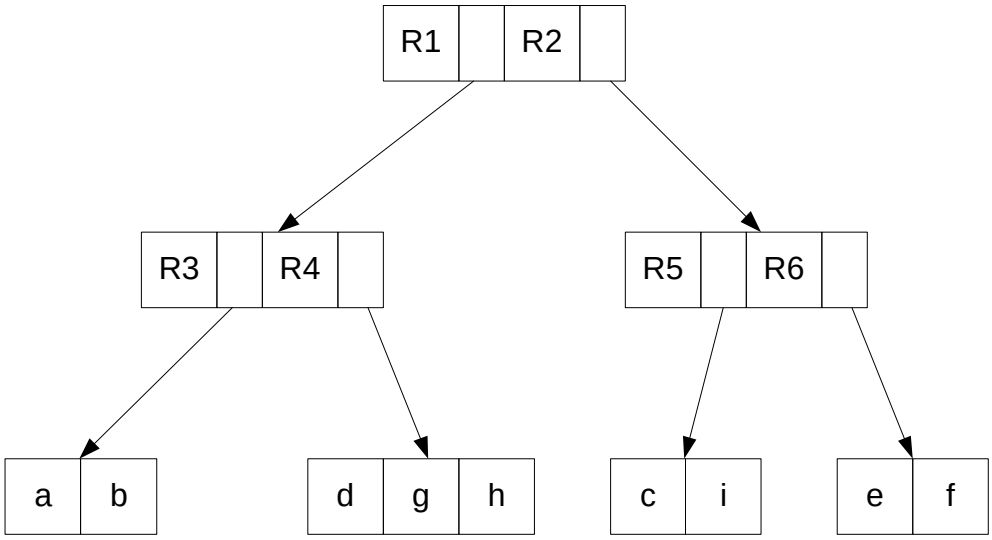
Die umschließenden Rechtecke, die zu unterschiedlichen Söhnen gehören, können sich überlappen. Jedoch kann ein Objekt nur in einem Blattknoten gespeichert werden, obwohl es in mehrere Bereiche fällt, die hierarchisch über der kleinsten Box stehen.

Beispiel:



Liniensegmente mit ihren umschließenden Rechtecken

ein möglicher R-Tree ist:



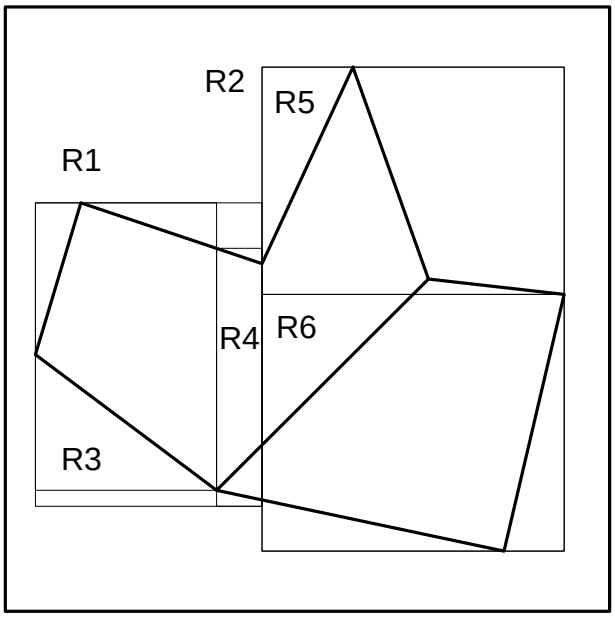
Wird in einer Anfrage nach einem Punkt gesucht, wird zuerst die *bounding box* zu diesem Objekt berechnet (hier: ein Punkt) und in der Wurzel des Baumes mit Suchen angefangen. Jede Box eines Sohnes wird untersucht, ob sie die Query-Box enthält. Ist dies der Fall, werden alle Söhne des Unterbaumes untersucht. Wenn mehrere Söhne die Box enthalten, müssen alle untersucht werden. Dies ist Nachteil des R-Trees, denn man muß bei der Suche nach einem Objekt eventuell in mehrere Unterbäume vordringen. Auf Blattebene wird der Knoten untersucht, ob der Punkt enthalten ist.

Wenn im Beispiel die Linie bestimmt werden soll, die durch Punkt Q geht, ist nicht klar, ob Q in R1 oder R2 liegt. Somit müssen beide Unterbäume durchgegangen werden.
Zuerst wird in R1 gesucht, dann weiter in R4. Dort wird festgestellt, daß die Linie nicht enthalten ist, obwohl Q im Rechteck R4 enthalten ist. Deswegen muß in R2 gesucht werden, und dort weiter in R5, wo sich die Linie i auch befindet.

Suche mit Bereichsobjekten oder Bereichsanfragen werden ähnlich gehandhabt. Zuerst wird die *bounding box* für die Region berechnet und mit der Suche angefangen. Bei einer Bereichsanfrage müssen aber auf Blattebene alle Bereichsobjekte, die dazugehören, untersucht werden, ob sie den gegebenen Bereich schneiden (oder enthalten, je nach Anfrage). Der Grund ist, daß die *bounding box* für ein Objekt die Anfrageregion überschneiden kann, das Objekt aber selbst nicht.

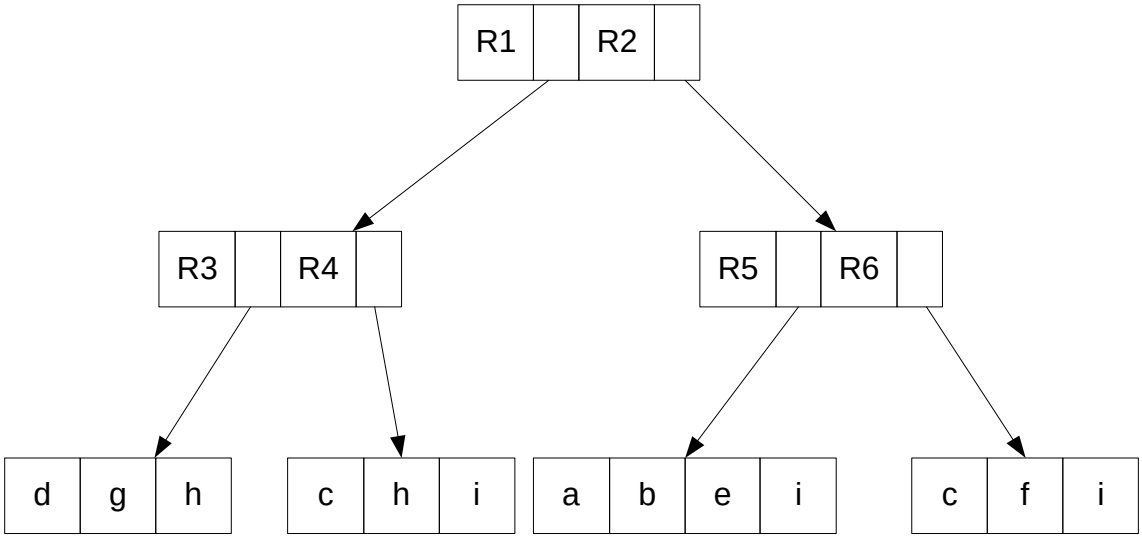
2.5.3 R^+-Tree

Der R^+-Tree verhindert das Überlappen innerhalb der umschließenden Rechtecke. Jedes Objekt wird mit allen Rechtecken, durch die es durchgeht, in Verbindung gebracht.Das Ergebnis ist, daß mehrere Wege von der Wurzel zum selben Objekt führen können. Dies vergrößert die Höhe des Baumes, aber verbessert die Geschwindigkeit des Auslesens der Daten.
Beispiel:



Liniensegmente mit ihren umschließenden Rechtecken

R^+-Tree:



Die Linien c und h erscheinen in 2 unterschiedlichen Knoten, Linie i sogar in 3.

Der Nachteil von getrennten Mengen tritt z.B. bei der Berechnung der Fläche eines Objektes auf. Alle Zellen, in denen das Objekt vorkommt, müssen besucht werden. Dies kommt auch beim Löschen eines Objekts vor. Ein weiteres Problem ist z.B. die Bestimmung, welche Objekte in einer bestimmten Region vorkommen. Dabei kann ein Objekt mehrmals gefunden werden, was aber problematisch ist, wenn das Ergebnis der Anfrage den Input für eine andere Anfrage liefert. Wenn man beispielsweise den Umfang aller Objekte in einer gegebenen Region berechnen will, soll jedes Objekt nur einmal berechnet werden. Die Entfernung der Duplikate ist dabei ein schwieriges Problem.

2.6 Indexierungsmethoden in Spatial

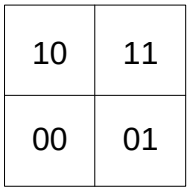
Ein *spatial index* ist nötig, um z.B. bei Anfragen die Suche innerhalb einer Tabelle zu minimieren, z.B. um

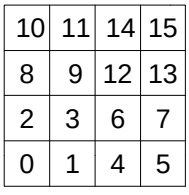
- Objekte innerhalb eines indizierten Raumes zu finden, die einen speziellen Punkt oder eine *area-of-interest* überdecken -> window query
- aus 2 indizierten Datenräumen die Objekte zu finden, die räumlich interagieren -> spatial join

Spatial benutzt ein Index-Schema, welches auf einem linearem Quadtree-Verfahren basiert. Der Koordinatenraum (für den Layer, in dem alle Objekte enthalten sind), wird der Prozeß der *Tesselation* ('Zerteilung') unterzogen. Dieser definiert durch Zerlegung des Koordinatensystems für jedes Objekt Teile, die es überdecken. Die Begrenzung des Koordinatensystems wird als Rechteck betrachtet. Im ersten Level der Zerlegung werden die Länge und Breite dieses Rechteckes in der Hälfte geteilt, so daß 4 neue Rechtecke entstehen. Jedes Rechteck, welches noch Teile des Objektes enthält, wird wieder auf die gleiche Art zerlegt. Dies geschieht solange, bis entweder die Teile eine bestimmte Größe erreicht haben, oder die maximale Anzahl an Teilen erreicht wurde, die ein Objekt überdecken soll.

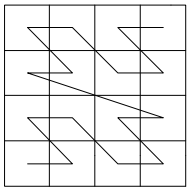
Die Teile in jedem Level können durch systematisches Entlanggehen entlang einer *space-filling*-Kurve durchsucht werden. Solche Kurven basieren auf der Annahme, daß jeder Wert durch eine feste Anzahl an k Bits ausgedrückt werden kann. Das Maximum der Werte auf jeder Achse ist 2^k.

Bsp für 2-dimensionalen Raum:



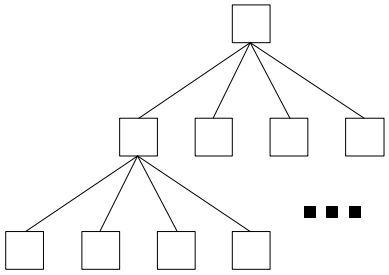


Quadtree-Zerlegung



space-filling curve

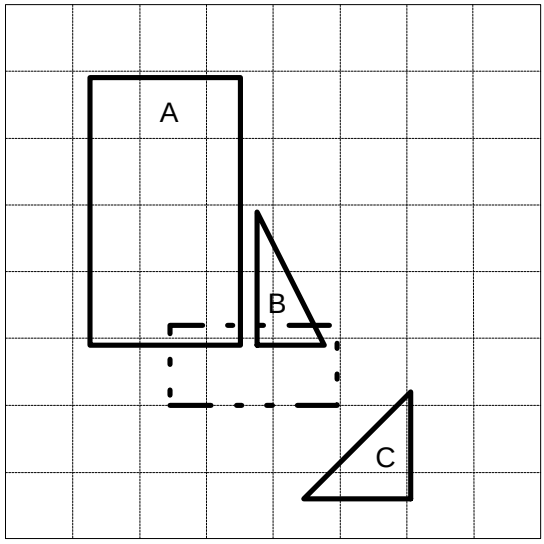
Die *space-filling*-Kurve führt zu einer linearen Sortierung im Bereich. Der Z-Wert, d.h. die Verkettung von X- und Y-Wert, und eine Referenz auf das Objekt werden in der Indextabelle gespeichert.
Der Region-Quadtree veranschaulicht direkt die Zerlegung des Bereiches in je 4 gleich große Teile. Jeder Knoten repräsentiert einen solchen Bereich, wobei die Wurzel den gesamten Bereich markiert. Der linke untere Quadrant wird durch 00, der rechte untere durch 01, der linke obere durch 10 und der rechte obere Quadrant durch 11 repräsentiert. D.h. alle Söhne von 00 beginne auch wieder mit 00, die Söhne von 01 mit 01usw..



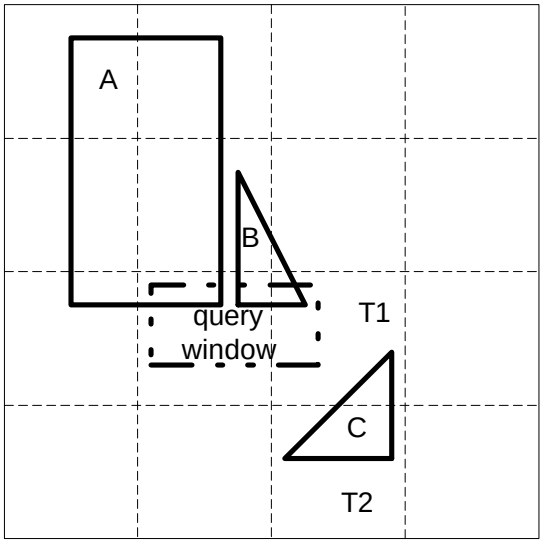
Region Quadtree

Spatial kann entweder immer gleichgroße (*fixed-sized*) oder variabel große (*variable-sized*) Teile benutzen. Das Ergebnis des Tesselation-Prozesses wird in der SDOINDEX-Tabelle gespeichert.

1. ***Fixed-Indexing*:** benutzt gleich große Teile, um ein Objekt zu überdecken. Es ist die bevorzugte Methode im Relationalen Modell. Da alle Teile gleich groß sind, haben sie alle einen Code gleicher Länge und der Standard-SQL Gleichheitsoperator '=' kann während einer Join-Operation als Vergleichsoperator genutzt werden. Dies führt zu einer sehr guten Performance. Die Größe der Teile kann durch den Benutzer mit dem Parameter SDO_LEVEL verändert werden. Er bestimmt die Anzahl der Zerlegungen des Koordinatensystems. Je kleiner die Teile werden, um so genauer kann das Objekt abgebildet werden. Aber bei kleinen Teilen und großem Objekt, würden sehr viele Teile benötigt. Andererseits könnte ein großes Objekt von wenig Teilen nur sehr schlecht angenähert werden. Deswegen sollte bei Festlegung des SDO_LEVEL auf gute Effizienz und Effektivität geachtet werden. Den Zusammenhang zwischen Teilgröße und Objektgröße veranschaulichen folgende Bilder.



Kleine, viele Teile

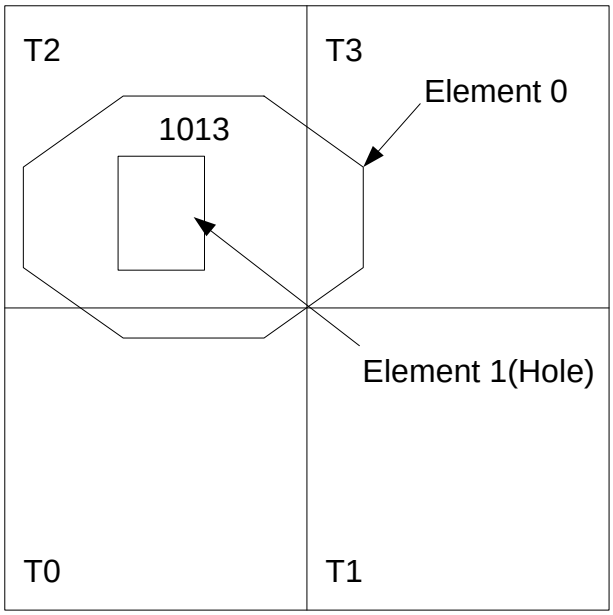


Große, wenige Teile

Im ersten Bild werden kleine Teile benutzt. Die Annäherung ist ganz gut, aber es sind viele Teile nötig, um z.B. Objekt A zu überdecken. Eine Anfrage über Objekte in der *area-of-interest* würde A und B identifizieren, aber nicht C.

Im zweiten Bild werden große Teile benutzt, so daß weniger Teile notwendig sind. Bei dieser schlechten Annäherung würde die Anfrage in diesem Gebiet auch Objekt C identifizieren, obwohl es weit weg von B ist (aber in T1 enthalten ist).

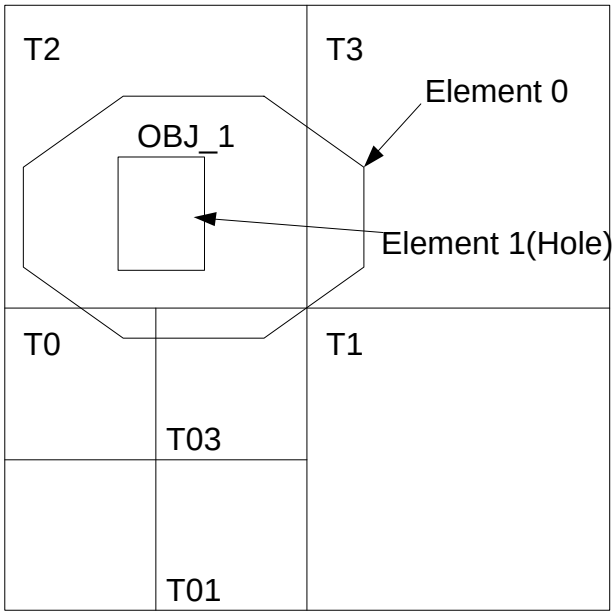
Weiteres Beispiel: SDO_LEVEL=1



Nur 3 der 4 Teile bedecken das Objekt und werden in die SDOINDEX-Tabelle übernommen.

SDO_GID	SDO_CODE
1013	T0
1013	T2
1013	T3

2. ***Hybrid-Indexing*:** ist die bevorzugte Indizierungsmethode für das Objekt-Relationale Modell. Es benutzt eine Kombination aus fixed-sized und variable-sized Teilen. Für jedes Objekt wird eine Menge an gleich großen Teilen erzeugt, die das Objekt voll überdecken, und danach eine Menge an variabel-großen Teilen, die Teile des Objekts überdecken. Der Parameter SDO_NUMTILES definiert die Anzahl an variable-sized Teilen, die für ein Objekt generiert werden sollen. Variabel-große Teile werden geteilt, wenn sie mit dem Objekt interagieren und die Zerteilung nicht in Teilen endet, die kleiner sind , als die vorgegebene Größe. Diese wird bestimmt durch den Parameter SDO_MAXLEVEL.
   Bsp.: SDO_LEVEL=1, SDO_NUMTILES=4 (solche kleinen Werte nur zur Vereinfachung). Die bedeckenden Teile werden in der SDOINDEX-Tabelle gespeichert. Die Nummerierung ist nur zur Veranschaulichung und entspricht nicht dem Format in Spatial.



Die *fixed-sized* Teile T0,T2,T3 interagieren mit dem Objekt. Nur diese werden in der Indextabelle gespeichert (Spalte SDO_GROUPCODE). T0 wurde weiter in 4 kleinere Teile (*variable-sized*) zerlegt, wobei nur T02 und T03 das Objekt bedecken. Auch diese beiden Teile werden in die Indextabelle übernommen (in die Spalte SDO_CODE).

SDO_ROWID	SDO_CODE	SDO_MAXCODE	SDO_GROUPCODE	SDO_META
GID_OBJ_1	T02	<binary data>	T0	<binary data>
GID_OBJ_1	T03	<binary data>	T0	<binary data>
GID_OBJ_1	T2	<binary data>	T2	<binary data>
GID_OBJ_1	T3	<binary data>	T3	<binary data>

Um die Art der Indizierung auszuwählen, müssen dieParameter SDO_LEVEL und SDO_NUMTILES gesetzt werden. Folgende Kombinationen sind zulässig:

SDO_LEVEL	SDO_NUMTILES	Auswahl
Nicht gesetzt	Nicht gesetzt	Fehler
>=1	Nicht gesetzt	Fixed indexing
>=1	>=1	Hybrid indexing
Nicht gesetzt	>=1	wird nicht unterstützt

2.7 Object-Relational-Model

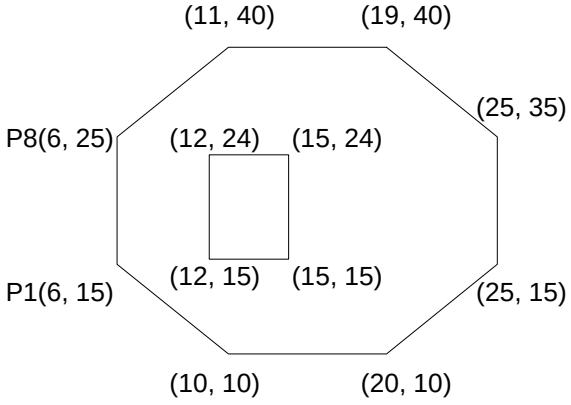
Die objekt-relationale Implementierung von Spatial besteht aus einer Menge von Datentypen, Indizierungsmethoden und Operatoren auf diesen Typen.

2.7.1 Datenstruktur

Ein geometrisches Objekt wird als Objekt in einer Zeile und Spalte des Typs SDO_GEOMETRY gespeichert.
Der Typ SDO_GEOMETRY besteht aus:

Bezeichnung	Datentyp	Beschreibung
SDO_GTYPE	Number	bestimmt den Typ des Objektes, z.B. Punkt, Linestring, Polygon
SDO_SRID	Number	reserviert für zukünftige Erweiterungen
SDO_POINT	SDO_POINT_TYPE	für Speicherung von Punktdaten; beinhaltet die Koordinaten der Punkte (nur wenn nächsten 2 Elemente Null sind, ansonsten ignoriert)
SDO_ELEM_INFO	MDSYS.SDO_ELEM_INFO_ARRAY	enthält Triplets an Attributen, wie die Werte in SDO_ORDINATES interpretiert werden sollen; besteht Objekt nur aus einem Element, sind hier nur 3 Zahlen gespeichert
SDO_ORDINATES	MDSYS.SDO_ORDINATE_ARRAY	enthält die Koordinaten des Objektes in X,Y bzw. X,Y,Z-Paaren

Die Indexerstellung, Operatoren und Funktionen ignorieren die z-Koordinate, da dieses Release nur zweidimensionale Objekte unterstützt. Die Dimension, mit der jeder Wert abgespeichert ist, wird als Metadaten in der Tabelle SDO_GEOM_METADATA gespeichert.
Beispiel:



Geometrie mit Loch

```
SDO_GEOMETRY Column = (
SDO_GTYPE = 3
SDO_SRID = NULL
SDO_POINT = NULL
SDO_ELEM_INFO = (1, 3, 1, 19, 3, 1)
SDO_ORDINATES = (6, 15, 10, 10, 20, 10,
                 25, 15, 25, 35, 19, 40,
                 11, 40, 6, 25, 6, 15,
                 12, 15, 15, 15, 15, 24,
                 12, 24, 12, 15) )
```

2.7.2 Metadaten

Die Metadaten, wie z.B. die Festlegung der Dimension, untere und obere Grenzen und Toleranz in jeder Dimension, müssen als Einträge in der Tabelle SDO_GEOM_METADATA gespeichert werden. Jeder Nutzer, der in einer Tabelle den Typ SDO_GEOMETRY hat, benötigt eine solche Tabelle. Für jede Tabelle, in der eine Spalte dieses Typs vorhanden ist, muß ein Eintrag in die Tabelle gemacht werden. Dafür ist der Nutzer selbst verantwortlich.

Erstellung der Tabelle:

```
Create Table SDO_GEOM_METADATA (
  TABLE_NAME      VARCHAR2(30),
  COLUMN_NAME VARCHAR2(30),
  DIMINFO             MDSYS.SDO_DIM_ARRAY);
```

Table_Name enthält den Namen der Tabelle, die die Spalte des Typs SDO_GEOMETRY enthält. Der Name der Spalte dieses Typs wird in der Spalte Column_Name abgelegt.
DIM_Info ist ein Feld variabler Länge, daß in geordneter Reihenfolge für jede Dimension einen Eintrag enthält. In diesem Feld sind Elemente des Typs SDO_DIM_ELEMENT enthalten.

Bezeichnung	Datentyp	Beschreibung
SDO_DIMNAME	VARCHAR2	Name der Dimension
SDO_LB	NUMBER NOT NULL	untere Grenze
SDO_UB	NUMBER NOT NULL	obere Grenze
SDO_TOLERANCE	NUMBER NOT NULL	Toleranz

Ein Beispiel für Metadaten folgt bei den Beispielen für das Einfügen eines Objektes.

2.7.3 Spatial Index Struktur

Spatial Index data und die Metadaten dazu werden in Tabellen gespeichert, die automatisch von den Indizierungsroutinen angelegt werden. Es gibt für jeden User genau eine SDO_INDEX_METADATA-Tabelle und eine Indextabelle pro indizierter Spalte in allen Tabellen des Users.
Die SDO_INDEX_METADATA-Tabelle enthält Daten wie SDO_LEVEL, SDO_NUMTILES, SDO_MAXLEVEL, SDO_INDEX_TABLE usw..
Die Indextabelle enthält Daten wie z.B. SDO_ROWID, SDO_CODE, SDO_GROUPCODE usw., wobei einige Spalten nur vorhanden sind, wenn der Indextyp *Hybrid indexing* gesetzt ist (empfohlen bei objekt-relationalem Modell).

Für nähere Informationen: siehe 'Oracle8i Spatial User's Guide and Reference'.

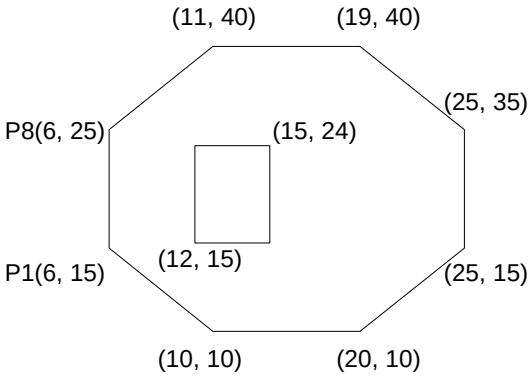
2.7.4 Laden und Indizierung

2.7.4.1 Laden

Es gibt 2 Möglichkeiten, räumliche Daten in die Datenbank abzuspeichern.

1.***bulk loading:*** wird benutzt, wenn eine große Menge an Daten in die Datenbank übertragen werden soll. Die Methode nutzt den SQL*Loader, um die Daten zu laden (siehe *Oracle8i Utilities User's Guide*).

2. ***transactional loading*** wird benutzt, um relativ kleine Datenmengen in die Datenbank zu übertragen. Diese Methode benutzt den INSERT-Befehl von SQL. Beispiel: Speicherung eines Polygons mit Loch



Geometrie mit Loch

Diese Beispiel setzt eine Tabelle Parks voraus, die auf folgende Weise erstellt wurde:

```
CREATE TABLE parks
(       Name VARCHAR2(32),
        Shape MDSYS.SDO_GEOMETRY  );
```

Die SQL-Anweisung zum Einfügen der Daten sieht so aus:

```
INSERT INTO parks
VALUES('OBJ_1', MDSYS.SDO_GEOMETRY( 3, NULL, NULL,
                    MDSYS.SDO_ELEM_INFO_ARRAY(1, 3, 1, 19, 3, 3),
                    MDSYS.SDO_ORDINATE_ARRAY(6, 15, 10, 10, 20, 10, 25, 15,
                    25, 35, 19, 40, 11, 40, 6, 25, 6, 15, 12, 15, 15, 24 ) ) );
```

Eintragen der Daten in die Tabelle SDO_GEOM_METADATA unter der Annahme, daß die 2 Dimensionen X und Y heißen und ihre Grenzen 0 und 100 sind:

```
INSERT INTO SDO_GEOM_METADATA
VALUES( 'Parks', 'Shape', MDSYS.SDO_DIM_ARRAY(
                            MDSYS.SDO_DIM_ELEMENT('X', 0, 100, 0.005),
                            MDSYS.SDO_DIM_ELEMENT('Y', 0, 100, 0.005) ) );
```

2.7.4.2 Indizierung

Nach dem Einfügen der Daten in die Datenbank, muß ein Index für den effizienten Zugriff auf die Daten erstellt werden. Die Varianten der Indizierung wurde oben schon beschrieben.

Folgendes Beispiel setzt eine Tabelle Parks voraus, und einen Eintrag für die Spalte Parks.Shape in der Tabelle SDO_GEOM_METADATA.

- Fixed-Index (immer gleich große Teile) erstellen:

```
CREATE INDEX Parks_Fixed ON Parks(Shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('SDO_LEVEL=8');
```

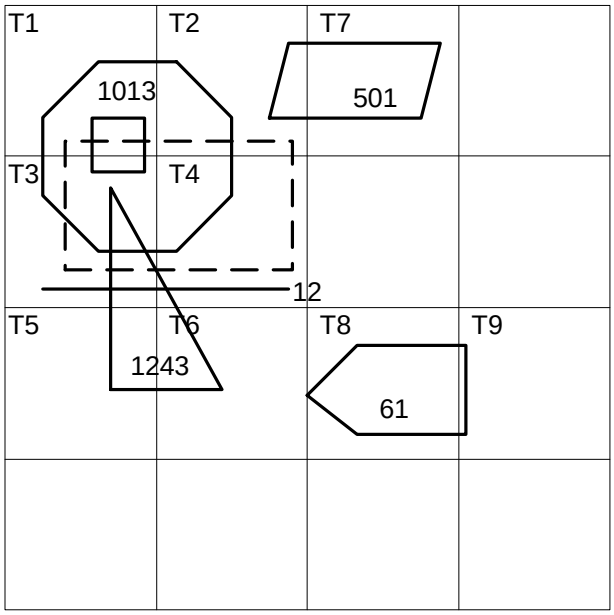
- Hybrid-Index (variabel-große Teile) erstellen:

  CREATE INDEX Parks_Fixed ON Parks(Shape)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  PARAMETERS('SDO_LEVEL=6, SDO_NUMTILES=12');

2.7.5 Anfragen

Wie schon erwähnt wurde, nutzt Spatial ein zweistufiges Anfragemodell mit Primär- und Sekundärfilter. Werden beide Filter hintereinander angewendet, ergibt dies das exakte Ergebnis. In den folgenden Beispielen werden zur besseren Veranschaulichung nur *fixed-sized* Teile verwendet, obwohl Hybrid-Indexing für das Objekt-Relationale Modell empfohlen wird.

Beispiel:



Eine typische Anfrage ist die Suche nach allen Objekten, die innerhalb des gestrichelten Bereiches liegen.

Primärfilter: Um nur den Primärfilter zu benutzten, muß die Funktion **SDO_FILTER()** verwendet werden. Diese Funktion nutzt nur die Indexdaten, um eine Menge an Kandidaten zu finden, die in der Fläche liegen könnten.

Syntax: SDO_FILTER(geometry1 MDSYS.SDO_GEOMETRY,
geometry2 MDSYS.SDO_GEOMETRY,
params VARCHAR2);

geometry1 : ist der Name einer Spalte des Typs MDSYS.SDO_GEOMETRY
geometry2 : ist ein Objekt des Typs MDSYS.SDO_GEOMETRY, welches entweder in einer Tabelle definiert sein kann oder nicht
params: Parameter, die das Verhalten von SDO_FILTER steuern

Bsp.:

```
SELECT A.Feature_ID FROM Target A
WHERE mdsys.sdo_filter(A.shape, mdsys.sdo_geometry(3,NULL,NULL,
                                    mdsys.sdo_elem_info(1,3,3),
                                    mdsys.sdo_ordinates(x1,y1,x2,y2)),
                        'querytype=window') = 'TRUE';
```

(x1,y1) und (x2,y2) sind die linke untere und obere rechte Ecke des Abfragefensters. Die Anfrage liefert alle Objekte, die ein gemeinsames Indexteil mit denen des Fensters haben. ImBeispiel sind dies die Teile T1,T2,T3 und T4. Somit liefert die Anfrage die IDs der Objekte 1013, 1243, 12 und 501.

Um beide Filter anzuwenden, wird die Funktion **SDO_RELATE()** benutzt. Diese wendet Primär- und Sekundärfilter hintereinander an.

Syntax:
```
SDO_RELATE(geometry1 MDSYS.SDO_GEOMETRY,
           geometry2 MDSYS.SDO_GEOMETRY,
           params VARCHAR2);
```
geometry1 : ist der Name einer Spalte des Typs MDSYS.SDO_GEOMETRY
geometry2 : ist ein Objekt des Typs MDSYS.SDO_GEOMETRY, welches entweder in einer Tabelle definiert sein kann oder nicht
params: Parameter, die das Verhalten von SDO_RELATE steuern

Bsp.:

```
SELECT A.Feature_ID FROM Target A
WHERE mdsys.sdo_relate(A.shape, mdsys.sdo_geometry(3,NULL,NULL,
                                    mdsys.sdo_elem_info(1,3,3),
                                    mdsys.sdo_ordinates(x1,y1,x2,y2)),
                        'mask=anyinteract querytype=window') = 'TRUE';
```

(x1,y1) und (x2,y2) sind die linke untere und obere rechte Ecke des Abfragefensters. Die Anfrage liefert alle Objekte, die innerhalb des Fensters liegen oder es überlappen. Hier wäre das Ergebnis die IDs der Objekte 1243 und 1013.
In beiden Beispielen wird das Abfragefenster im Speicher indiziert, so daß die Performance sehr gut ist.

Um zu bestimmen, welche Objekte einer Tabelle innerhalb einer bestimmten Distanz von einem Referenzobjekt liegen, wird die Funktion **SDO_WITHIN_DISTANCE()** benutzt.

Syntax:
```
SDO_WITHIN_DISTANCE(geometry1 MDSYS.SDO_GEOMETRY,
                    aRefGeom MDSYS.SDO_GEOMETRY,
                    params VARCHAR2);
```

geometry1 : ist der Name einer Spalte des Typs MDSYS.SDO_GEOMETRY
aRefGeom : ist eine Instanz des Typs MDSYS.SDO_GEOMETRY
params: Parameter, die das Verhalten von SDO_WITHIN_DISTANCE steuern

Bsp:

```
SELECT A.Feature_ID FROM Target A
WHERE mdsys.sdo_within_distance(A.shape,
                                mdsys.sdo_geometry(3,NULL,NULL,
                                  mdsys.sdo_elem_info(1,3,3),
                                  mdsys.sdo_ordinates(x1,y1,x2,y2)),
                                'distance=1.35') = 'TRUE';
```

Spatial Join: Ein Spatial Join ist äquivalent zu regulären Joins, nur das jetzt spatiale Operatoren vorhanden sind. In Spatial findet ein Join statt, wenn alle Objekte eines Layers mit allen anderen Objekten eines anderen Layers verglichen werden sollen. Dies ist der Gegensatz zu einem Abfragefenster, in dem nur ein einfaches Objekt (das Fenster) mit allen Objekten eines Layers verglichen wird.
Spatial Joins können z.B. genutzt werden, um Fragen wie "Welche Straßen gehen durch Nationalparks?" zu beantworten.

Bsp: Folgende Tabellenstruktur liege vor:

```
Parks(      GID VARCHAR2(32), Shape MDSYS.SDO_GEOMETRY);
Highways(GID VARCHAR2(32), Shape MDSYS.SDO_GEOMETRY);
```

Abfrage mit Join:

```
SELECT A.GID, B.GID
FROM parks A, highways B
WHERE mdsys.sdo_relate(A.shape, B.shape,
          'mask=ANYINTERACT querytype=join');
```

2.7.6 Geometrie-Funktionen

In der folgenden Tabelle ist eine Auswahl einiger Funktionen dargestellt, die zur Berechnung von geometrischen Sachverhalten zur Verfügung stehen. Zur näheren Informationen: siehe 'Oracle8i Spatial User's Guide and Reference'.

Funktion	Beschreibung
SDO_GEOM.AREA	berechnet die Fläche eines zweidimensionalen Polygons
SDO_GEOM.LENGTH	berechnet die Länge oder den Umfang eines Objektes
SDO_GEOM.SDO_POLY_DIFFERENCE	liefert ein Objekt, das die Differenz zwischen 2 Polygonen repräsentiert
SDO_GEOM.SDO_POLY_UNION	liefert ein Objekt, das die Vereinigung zwischen 2 Polygonen repräsentiert
SDO_GEOM.VALIDATE_GEOMETRY	testet, ob ein geometrisches Objekt der Beschreibung des Typs entspricht, z.B. ob Polygon mind. 4 Punkte hat und der 1. und der letzte Punkt identisch sind

2.8 Relational-Model

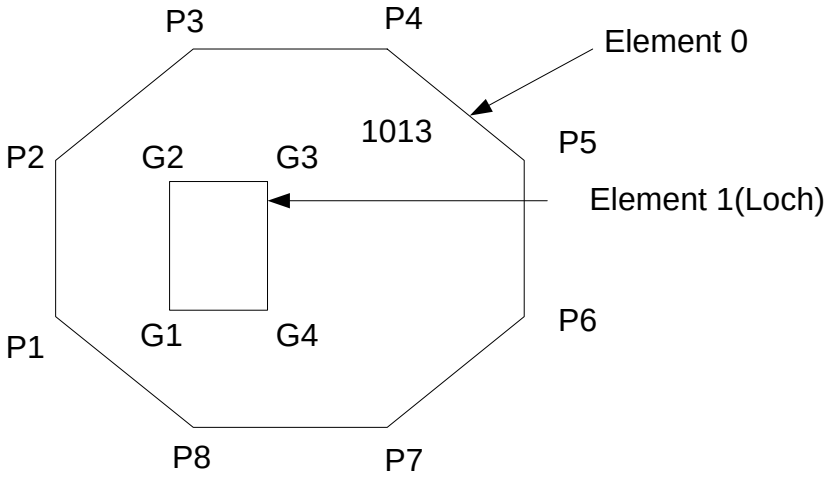
Diese Variante der Implementierung von Spatial nutzt numerische Datentypen um räumliche Daten abzuspeichern.

2.8.1 Datenstruktur

Statt in 2 Tabellen wie im Objekt-Relationalen Modell, werden im Relationalen Modell die Daten und die Indizes in 4 Tabellen abgelegt. Alle 4 Tabellen zusammen werden als ein Layer bezeichnet.

Tabellenname	Inhalt
<layername>_SDOLAYER	enthält u.a. die Anzahl der Werte in der Tabelle SDOGEOM, die Werte für SDO_LEVEL und SDO_NUMTILES
<layername>_SDODIM	enthält die Dimension, die obere und untere Grenze der Dimension, die Toleranz und den Name
<layername>_SDOGEOM	enthält Typ und die Werte für alle Elemente eines Objekts und die SDO_GID, die für jedes Objekt eindeutig ist
<layername>_SDOINDEX	enthält die Indizes

Beispiel:



Der Inhalt der Tabellen sieht dann folgendermaßen aus:

<layername>_SDOLAYER: In jeder Datenzeile in SDO_GEOM sollen 4 Koordinatenwerte enthalten sein.

SDO_ORDCNT (number)
4

<layername>_SDODIM: Grenzen der Dimensionen: 0, 100; Toleranz: 0.05

SDO_DIMNUM	SDO_LB	SDO_UB	SDO_TOLERANCE	SDO_DIMNAME
1	0	100	.05	X Achse
2	0	100	.05	Y Achse

<layername>_SDOGEOM:

SDO_GID	SDO_ESEQ	SDO_ETYPE	SDO_SEQ	SDO_X1	SDO_Y1	SDO_X2	SDO_Y2
1013	0	3	0	P1(X)	P1(Y)	P2(X)	P2(Y)
1013	0	3	1	P2(X)	P2(Y)	P3(X)	P3(Y)
1013	0	3	2	P3(X)	P3(Y)	P4(X)	P4(Y)
1013	0	3	3	P4(X)	P4(Y)	P5(X)	P5(Y)
1013	0	3	4	P5(X)	P5(Y)	P6(X)	P6(Y)
1013	0	3	5	P6(X)	P6(Y)	P7(X)	P7(Y)
1013	0	3	6	P7(X)	P7(Y)	P8(X)	P8(Y)
1013	0	3	7	P8(X)	P8(Y)	P1(X)	P1(Y)
1013	1	3	0	G1(X)	G1(Y)	G2(X)	G2(Y)
1013	1	3	1	G2(X)	G2(Y)	G3(X)	G3(Y)
1013	1	3	2	G3(X)	G3(Y)	G4(X)	G4(Y)
1013	1	3	3	G4(X)	G4(Y)	G1(X)	G1(Y)

2.8.2 Laden und Indizierung

2.8.2.1 Laden

Auch im Relationalen Modell gibt es 2 Möglichkeiten, die Daten in die Datenbank zu laden: *bulk loading* oder *transactional loading*, wobei hier nur auf letzteres eingegangen werden soll.

Beispiel: Um ein Polygon einzufügen, müssen die Daten mittels INSERT in die Tabelle <layer>_SDOGEOM eingetragen werden.

```
INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                            SDO_X1, SDO_Y1, SDO_X2, SDO_Y2,SDO_X3,
                            SDO_Y3, SDO_X4, SDO_Y4, SDO_X5,SDO_Y5)
VALUES (18, 0, 3, 0, 1, 15, 1, 16, 2, 17, 3, 17, 4, 18);

INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                            SDO_X1, SDO_Y1, SDO_X2, SDO_Y2,SDO_X3,
                            SDO_Y3, SDO_X4, SDO_Y4, SDO_X5,SDO_Y5)
VALUES (18, 0, 3, 1, 4, 18, 5, 18, 6, 19, 7, 18, 6, 17);

INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                            SDO_X1, SDO_Y1, SDO_X2, SDO_Y2,SDO_X3,
                            SDO_Y3, SDO_X4, SDO_Y4, SDO_X5,SDO_Y5)
VALUES (18, 0, 3, 2, 6, 17, 7, 16, 7, 15, 6, 14, 7, 13);

INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                            SDO_X1, SDO_Y1, SDO_X2, SDO_Y2,SDO_X3,
                            SDO_Y3, SDO_X4, SDO_Y4, SDO_X5,SDO_Y5)
VALUES (18, 0, 3, 3, 7, 13, 6, 12, 5, 13, 4, 13, 3, 14);

INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                            SDO_X1, SDO_Y1, SDO_X2, SDO_Y2,SDO_X3,
                            SDO_Y3)
VALUES (18, 0, 3, 4, 3, 14, 2, 14, 1, 15);
```

In diesem Beispiel ist die Zeile der Tabelle nicht lang genug, um alle Koordinaten des Polygons hintereinander einzutragen. Deswegen muß man SDO_GID, SDO_ESEQ, und SDO_ETYPE wiederholen und nur SDO_SEQ in jeder Zeile inkrementieren.

2.8.2.2 Indizierung

Nach dem Einfügen der Daten muß ein Index erstellt werden. Dazu muß erst eine Tabelle <layername>_SDOINDEX erstellt werden.

```
CREATE TABLE <layername>_SDOINDEX
(       SDO_GID number,
        SDO_CODE raw(255)
);
```

Um einen Index zu erstellen, gibt es 2 Funktionen:

1. **SDO_ADMIN.POPULATE_INDEX(layername)**: diese Funktion führt den Prozeß der *tesselation* auf alle geometrischen Objekte in der <layername>_SDOGEOM Tabelle aus, die noch keinen Eintrag in der Indextabelle haben.

2. **SDO_ADMIN.UPDATE_INDEX(layername, GID)**: diese Funktion erstellt den Index nur für das angegebene Objekt. Ist für dieses schon ein Eintrag in der Indextabelle vorhanden, wird dieser überschrieben.

Welche Indizierungsmethode genutzt wird (*fixed* oder *hybrid indexing*) , hängt von den Werten SDO_LEVEL und SDO_NUMTILES in der Tabelle <layername>_SDOLAYER ab.

2.8.3 Anfragen

Wie schon erwähnt, benutzt Spatial ein zweistufiges Anfragemodell mit Primär- und Sekundärfilter. Im Relationalen Modell sind die Anfragen aber schwieriger zu handhaben. Dies soll am selben Beispiel wie im Objekt-Realtionalen Modell gezeigt werden. Es sollen alle Objekte gefunden werden, die im markierten Fenster liegen.

Primärfilter: Zuerst muß ein Layer für die Anfragefläche erstellt werden und die Koordinaten des Bereiches angegeben werden. Für den Layer bekommt man eine GID zurückgeliefert. Außerdem wird ein Index für den Layer erstellt. Danach kann man eine Anfrage erstellen, die einen Join auf dem Index des Anfragefensters und dem Index des Layer der Objekte ausführt. Damit werden alle Objekte gefunden, die gemeinsame Teile im Index haben.

```
SELECT DISTINCT A.SDO_GID
FROM <layer1>_SDOINDEX A, <windowlayer>_SDOINDEX B
WHERE A.SDO_CODE=B.SDO_CODE
AND B.SDO_GID = {GID des Anfragefensters};
```

Das Resultset dieser Query ist die Menge {1013, 501, 1243, 12}.

Sekundärfilter: Dieser Filter führt eine exakte Berechnung auf der Datenmenge aus.
Das folgende Beispiel zeigt die Hintereinanderausführung beider Filter:

```
SELECT SDO_GID
FROM <layer1>_SDOGEOM,
(
        SELECT SDO_GID GID1
        FROM (
                SELECT DISTINCT A.SDO_GID
                FROM <layer1>_SDOINDEX A, <windowlayer>_SDOINDEX B
                WHERE A.SDO_CODE=B.SDO_CODE
                AND B.SDO_GID = {GID des Anfragefensters};
                )
        WHERE SDO_GEOM.RELATE('<layer1>, SDO_GID, 'ANYINTERACT',
        '<window>, 1) = 'TRUE'
)
WHERE SDO_GID= GID1;
```

Die Anfrage liefert alle Objekte, die innerhalb des Fensters liegen oder es überlappen. Hier wäre das Ergebnis die IDs der Objekte 1243 und 1013.

Spatial Join: Auch hier soll der Join am Beispiel der Frage "Welche Straßen führen durch Nationalparks?" gezeigt werden.
Die Tabellen des Users könnten folgende Struktur haben:

Parks:

Name	GID	Campsite

Highways:

Name	GID	Width

Die Tabelle SDOINDEX sieht für beide Tabellen gleich aus:
Parks_SDOINDEX, Highways_SDOINDEX:

GID	CODE	MAX

Das Ergebnis des Primärfilters muß an den Sekundärfilter übergeben werden.

```
SELECT DISTINCT GID_B
FROM
(
        SELECT DISTINCT A.SDO_GID GID_A, B.SDO_GID GID_B
        FROM parks_SDOINDEX A, highways_SDOINDEX B
        WHERE A.SDO_CODE=B.SDO_CODE
)
WHERE SDO_GEOM.RELATE('Parks', GID_A, 'ANYINTERACT',
                                'HIGHWAYS',GID_B) <> 'FALSE';
```

2.8.4 Geometrie-Funktionen

Folgende Funktionen werden im Relationalen Modell unterstützt:

Funktion	Beschreibung
SDO_GEOM.RELATE	bestimmt wie 2 Objekte interagieren
SDO_GEOM.VALIDATE_GEOM	testet, ob ein geometrisches Objekt der Beschreibung des Types entspricht
SDO_GEOM.VALIDATE_LAYER	untersucht einen Layer auf korrekte Objektbeschreibung

2.9 Selektierung eines der beiden Modelle

Grundsätzlich ist das Objekt-Relationale Modell vorzuziehen, wenn keine Reproduktion oder verteilte Datenbank notwendig ist.

Vorteile des Objekt-Relationalen Modells sind:

- zusätzliche geometrische Typen
- Index und Anfrage sind leicht handhabbar
- Objekte sind in einer Spalte und einer Zeile gespeichert
- gute Performance

Vorteile des Relationalen Modells sind:

- verteilte Datenbanken werden unterstützt
- Tabellenpartitionierung und *parallel index loading* werden unterstützt

3. Vergleich Oracle Spatial – SQL/MM Spatial

SQL/MM unterstützt ebenso wie Oracle Spatial nur zweidimensionale geometrische Objekte und Operationen.
Geometrische Objekte sind:

- Punkt
- Bögen und Untertypen, die durch unterschiedliche Interpolation zwischen den Punkten entstehen: ST_LineString (lineare Interpolation),
  ST_CircularString (circular arcs),
  ST_CompoundString (mixed)
- zweidimensionale Oberflächen: ST_Polygon, ST_CurvePolygon

In der Hierarchie über allen Typen steht der Typ ST_Geometry, in dem alle geometrischen Objekte zusammengefasst sind.

Enthaltene Operationen sind u.a.:

- Entfernung
- Tests (enthalten, überschneiden, berühren ...)
- Schnittpunkte, Differenzmenge, Vereinigungsmenge
- Länge, Fläche, Umfang

4. Anhang

Literaturangaben:

1. Online Dokumentation "Oracle 8i Spatial Reference and User's Guide"
2. SQL3 Tutorial
3. "Modern Database Systems" Won Kim, Addison Wesley
4. "Database Management Systems" R.Ramakrishnan/ J.Gehrke, McGraw Hill