

Performancetests für SQL Server 2008 bei der Verarbeitung räumlich- thematischer Daten

Bachelorarbeit

Technische Universität Hamburg Harburg

Igor Zlatkevich, 2011

Erstprüfer: _____

Zweitprüferin: _____

Bearbeitungszeitraum: _____

Hamburg, _____

Inhaltsverzeichnis

Inhaltsverzeichnis	2
Abbildungsverzeichnis.....	4
Einleitung.....	5
1 Datenbanksysteme allgemein	6
1.1 Architektur von Datenbanksystemen	6
1.2 Datenspeicherung: Datenmodell und Datentypen	6
2 GIS allgemein.....	8
2.1 Operationen mit GIS Objekten	9
2.2 OGC Funktionen (Simple Feature Modell)	10
2.2.1 Allgemeine Methoden	10
2.2.2 Methoden zum Testen der Beziehungen	11
2.2.3 Methoden für Analyse	11
2.2.4 OGS-konforme Funktionen von MS SQL Server	11
2.3 Dimensionsmatrix im 9-Intersection Model.....	12
3 GIS-Eigenschaften einer Datenbank	15
3.1 GIS Datentypen	15
3.2 Räumliche Indizes	16
3.3 Grafische Analyse GIS-Daten	16
4 MS SQL Server als Datenbanksystem für GIS	17
4.1 Installation von MS SQL Server	18
4.2 Werkzeuge.....	18
4.2.1 SQL Server Management Studio und Squirrel als Frontend	18
4.2.2 FME Desktop und shape2sql als Software für Datenimport.....	18
4.3 Erstellung der räumlichen Datenbank	19
4.4 OGC-methoden und räumliche Indizes in MS SQL Server	19
4.5 Import räumlicher Daten aus Shape-Dateien	20
4.6 Datenschema.....	21
4.6.1 Tiger/Line Daten: die Idee.....	21
4.6.2 Tiger Shape-Dateien: Struktur allgemein.....	22
4.7 Aufbau der Datenbank nach dem Import von Tigerdaten	23
5 Maßnahmen zur Erhöhung der Datenbankleistung	24

5.1.1	Tuning von Indizes	24
5.1.2	RCC-Relationen und benutzerdefinierte Funktion für deren Ermittlung.....	25
5.1.3	Erstellung von Tabellen mit vorberechneten RCC-Relationen	27
6	Abfragen an GIS Datenbank und Performancetests.	29
6.1.1	Test der verschiedenen Implementierungen der RCC-Relationen	29
6.2	Tests mit LUBM-basierte Abfragen	29
7	Fazit	35
8	Literaturverzeichnis	36

Abbildungsverzeichnis

Abbildung 1: Datenbankarchitektur [6]	6
Abbildung 2: Eine Reihe der Punkt-Objekte in Australien [1].....	8
Abbildung 3: Polygondarstellung eines Kreises mit 4 bzw. 8 Punkten.....	9
Abbildung 4: OGC-Methoden der <i>Geometry</i> Klasse in UML Notation.....	10
Abbildung 5: Dimensionen in DE-9IM am Beispiel von zwei Polygonen [1].....	13
Abbildung 6: Eine Linie auf der Erdoberfläche in <i>Geometry</i> -Darstellung [1].....	15
Abbildung 7: Eine Linie auf der Erdoberfläche in <i>Geography</i> -Darstellung [1]	15
Abbildung 8: Aufbau des räumlichen Indexes	20

Einleitung

Ziel dieser Arbeit ist es, die Performance eines relationalen Datenbanksystems (Microsoft SQL Server 2008 R2) bezüglich GIS-Datenverarbeitung ausführlich zu testen. Als GIS-Datenquelle wurden die Tigerline-Daten der Stadt New-York (www.census.gov) ausgewählt. Nach der Installation von MS SQL Server 2008 R2 wurden die Shape-Dateien mit Hilfe von zusätzlicher Software (FME Desktop und SarpGIS) in Datenbank importiert. Diese Datenbank dient als Basis für Performancetests mit LUBM Abfragen, die im Rahmen dieser Arbeit in SQL Abfragen übersetzt wurden. Die ersten Tests haben gezeigt, dass sogar die Abfragen mit geringerer Selektivität zu lange ausgeführt werden. Als Optimierungsschritt wurden folgende Maßnahmen erforscht: Erstellung der vorkalkulierten Tabellen und eine überwiegende Verwendung der Methoden, die räumlichen Index unterstützen. Es konnte eine relativ hohe Effizienz erreicht werden, die Benutzung von MS SQL Server für LUBM Operationen mit GIS Daten sinnvoll macht.

1 Datenbanksysteme allgemein

Die meisten Datenbanksysteme dienen zur Verwaltung des Datenbestandes und bieten folgende Funktionalität:

- Verwaltung des Mehrbenutzer-Datenzugriffes mit Rechtenunterscheidung
- Verwaltung des gleichzeitigen Datenzugriffes auf diesselben Daten
- Sicherung/Wiederherstellung der Daten
- Datenmanipulation: Dateneingabe, Datenaktualisierung und Datenausgabe mit Sicherstellung der Persistenz, Konsistenz und Redundanz

1.1 Architektur von Datenbanksystemen

Datenbanksysteme bestehen allgemein aus drei Schichten. Am tiefsten liegen die Daten selbst und für Datenbankverwaltung relevante Informationen. Sie werden in einem speziellen, von Datenbanksystem vordefinierten Format gespeichert.

Etwas höher liegt eine Verwaltungsschicht, wo die Zugriffsverwaltung und Transaktionsverwaltung erfolgt. Auch alle Datenmanipulationen finden auf dieser für Endbenutzer transparenter Ebene statt.

Auf der dritten und höchsten Ebene laufen eigenständige Prozesse, die als *Front-end* Applikationen für Benutzer zuständig sind und die man als Clients bezeichnen kann. Sie kommunizieren mit einer Verwaltungsschicht des Datenbanksystems (Abbildung 1).

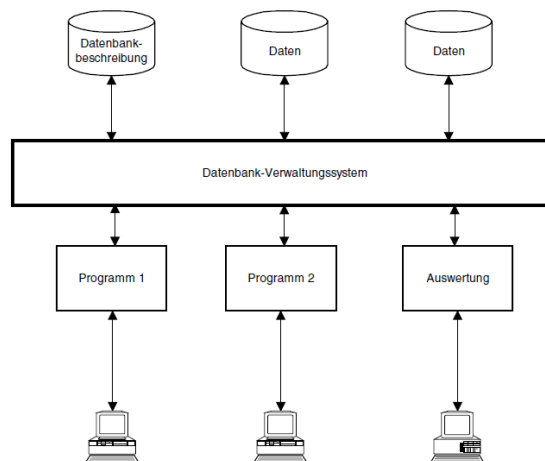


Abbildung 1: Datenbankarchitektur [6]

1.2 Datenspeicherung: Datenmodell und Datentypen

In Laufe dieser Arbeit wird mit dem Datenbanksystem gearbeitet, das auf dem relationalen Datenbankmodell beruht. Dieses Modell wurde 1970 von Edgar F. Codd erstmals

vorgeschlagen und bleibt bis heute trotz einiger Kritikpunkte ein etablierter Standard für Datenbanken.

Grundlage des Konzeptes relationaler Datenbanken bildet eine Relation, die eine mathematische Beschreibung einer Tabelle darstellt. Operationen auf Relationen werden durch die relationale Algebra bestimmt. Die relationale Algebra ist somit die theoretische Grundlage von SQL.

Da allgemeiner DL-Lite Ansatz von [7], der Ontologie und Modell-Sprachen vereint, sich auf relationale Datenbanken bezieht, wird deswegen in dieser Arbeit Performance einer relationalen Datenbank gemessen.

Die Daten in einem relationalen Datenbanksystem sind typisiert. Datentypen teilen, ähnlich wie bei typisierten Programmiersprachen, mit, welche Werte in konkreten Spalten gespeichert werden dürfen. Neben den Basisdatentypen wie *int*, *bigint*, *varchar* existieren auch systemeigene Datentypen, die sich vom System zu System unterscheiden. Auch GIS- Datentypen gehören zu einer Reihe zusätzlicher Datentypen, die die Funktionalität eines Datenbanksystems erweitern. GIS Datentypen erlauben in einer Datenbank auch solche Information über Objekte zu speichern, wie Koordinaten und Art des Objektes. Im nächsten Kapitel werden GIS (*Geographic Information Systems*) näher betrachtet.

2 GIS allgemein

Unter dem Begriff GIS (*Geographic Information System*) versteht man ein Informationssystem, das Werkzeuge für die Erfassung, Speicherung, Reorganisierung, Modellierung, Analyse und grafische Darstellung raumbezogener Daten vereint. Für die obengenannten Aufgaben werden oft eine Datenbanklösung (der z.B. GIS Datentypen unterstützt oder bei der eine GIS-Erweiterung nachinstallierbar ist) und eine Sammlung von zu Middleware gehörter spezialisierter GIS-Software verwendet. Unter Daten sind digitale Informationen gemeint, die eine Position, Form und Orientierung eines Objekts im Raum (zum Beispiel auf der Erdoberfläche) bestimmen. Im allgemeinen Fall sind es die Koordinaten von Rändern eines Objektes, sie beschreiben das Objekt mit einer bestimmten Genauigkeit, die von der Anzahl der vorhandenen Punkte abhängig ist (wenn das beschriebene Objekt selbst kein idealer Punkt ist). Solche Daten nennt man Vektordaten. Rasterdaten sind dagegen die Abbildungen von Objekten, die zum Beispiel durch Scannen von Fotografien entstehen.

Zu den GIS-Objekten zählen Objekte beliebiger Art, die mit ihren Koordinaten bestimmt und in einem GIS gespeichert werden können. Das können sowohl die Naturobjekte (Berge, Wälder etc.) als auch künstliche vom Menschen erbaute Objekte (Gebäude, Denkmäler etc.) sein. Es gibt auch weitere, engere Klassifikationen, bei denen Objekte mathematisch beschrieben werden: geometrische oder geografische (die Erde wird als eine Fläche oder Kugel dargestellt) und zwei-, drei- oder vierdimensionale Objekte. Weiter werden einfachheitshalber nur zweidimensionale geometrische Objekte betrachtet.

Vektorobjekte sind leichter mathematisch zu beschreiben und deswegen arbeiten wir mit Vektordaten und nicht mit Rasterdaten. Alle komplexe Objekte werden durch drei Primitive dargestellt: *Punkt*, *Linie* und *Polygon*. Mit einem Punkt, wie auf Abbildung 2 zu sehen, kann man genau die Lage eines Objektes definieren.



Abbildung 2: Eine Reihe der Punkt-Objekte in Australien [1]

In diesem Fall wird die Form des Objekts vernachlässigt. Wird ein Objekt als Linie dargestellt, so hat es keine Fläche, nur eine Länge. Als Beispiel kann man das Verkehrsnetz nennen.

Der dritte Typ von Objekten sind Polygone: sie haben Flächen und dürfen Löcher enthalten. Je mehr Randpunkte das Polygonobjekt definieren, desto genauer kann seine reale Form durch Approximation wiedergegeben werden (Abbildung 3).

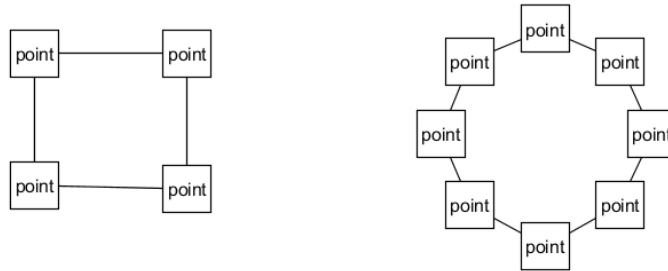


Abbildung 3: Polygondarstellung eines Kreises mit 4 bzw. 8 Punkten

Jedes Polygon besitzt eine innere Fläche (*interior*), die Kanten (*boundaries*) und eine externe Fläche (*exterior*). Die externe Fläche besteht aus allen Punkten, die außerhalb des Polygons liegen.

Die Lage des Objektes bestimmt man mit einem Koordinatensystem. Es existieren mehrere Typen von Koordinatensystemen, deren Zweck ist, die Lage des Objekts genau zu definieren. Zwei davon sind weitverbreitet: Das geografische Koordinatensystem und das projizierte Koordinatensystem.

Das *Geografische Koordinatensystem* beschreibt Objekte mit geographischer Breite und geographischer Länge, die den Winkeln zwischen der Ebene durch den Nullmeridian und der Meridianebene im Messungspunkt darstellen und in Grad gemessen werden. Die Geographische Breite ist die nördliche oder südliche Entfernung eines Punktes der Erdoberfläche vom Äquator. Die Geographische Länge ist ein Winkel, der ausgehend vom Nullmeridian (0°) bis 180° in östlicher und 180° in westlicher Richtung gemessen wird.

Das *Projizierte Koordinatensystem* hat im Vergleich mit dem geographischen Koordinatensystem nur zwei Dimensionen. Um die Objektlage zu bestimmen werden hier die linearen Maßeinheiten verwendet, daher sind wegen landschaftlicher Höhenunterschiede die Entfernungen nur annähernd berechenbar.

2.1 Operationen mit GIS Objekten

Nicht nur die Speicherung von Daten ist eine wichtige Eigenschaft von geographischen Informationssystem; auch die Operationen, die auf diese Objekte angewendet werden können, sind von großer Bedeutung. Die Funktionen, die in einem GIS-System enthal-

ten sind, erlauben bestimmte Objekte zu vergleichen, die Entfernungen zwischen Objekten zu messen und andere Beziehungen zwischen Objekten zu testen. OGC (*Open Geospatial Consortium*, <http://www.opengeospatial.org>) hat eine Standardisierung eingeführt, die eine Sammlung von Funktionen anbietet. Diese Funktionen werden von den meisten GIS Systemen implementiert und durch zusätzliche, systemeigene Funktionen erweitert.

2.2 OGC Funktionen (Simple Feature Modell)

OGC (*Open Geospatial Consortium*, <http://www.opengeospatial.org>) ist eine internationale Gesellschaft, das 424 Unternehmen vereint. Hauptziel ist eine Entwicklung von Standards für die Verarbeitung und Speicherung geografischer Daten.

OGC hat drei topologische Frameworks entwickelt, die Zlatanova in ihrem Buch [2] beschreibt. Das sind boolesche Mengen von Operationen, Egenhofer-Operatoren und Clementini-Operatoren. Mit diesen Frameworks entwickelte OGC eine Klasse, die drei Arten von Funktionen (Methoden) anbietet: allgemeine Funktionen, Funktionen zum Testen der Beziehungen zwischen Objekten und Funktionen, die analytischen Zwecken dienen, siehe Abbildung 4.

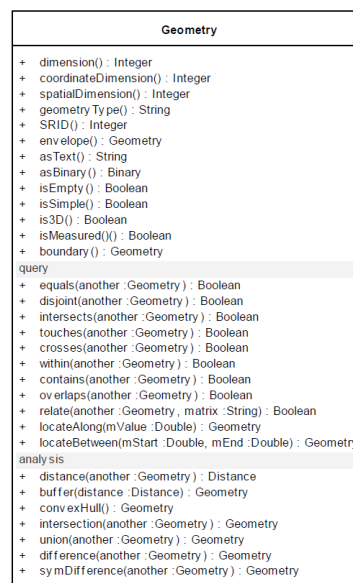


Abbildung 4: OGC-Methoden der *Geometry* Klasse in UML Notation

2.2.1 Allgemeine Methoden

Zu den allgemeinen Methoden zählen diejenige Methoden, die nicht dem Vergleich zweier Objekte (*Geometrien*) und nicht Analyse Zwecken dienen. Sie spiegeln einige Eigenschaften des Objekts oder GIS-Systems wieder, wie zum Beispiel das verwendete Koordinatensystem oder die SRID des Objekts. Als Beispiel sind hier zwei allgemeine Methoden vorgestellt:

GeometryType (): String – gibt den Namen der Geometrie des Objektes (*polygon, line, point*) zurück.

SRID (): Integer – gibt *Spatial Reference System ID* für einen Objekt zurück

Betrachten wir SRID Begriff etwas genauer: im vorherigen Kapitel wurden unterschiedliche Koordinatensysteme beschrieben. Um jedes System eindeutig identifizieren zu können wird in GIS Programmen Identifikationsnummer SRID benutzt. Für diese Bachelor Arbeit ist SRID ein wichtiger Begriff, weil SQL Server ihn braucht um die gespeicherten Koordinaten einheitlich zu betrachten.

2.2.2 Methoden zum Testen der Beziehungen

Diese Methoden können auf die wichtigsten Fragen bezüglich Beziehungen zwischen Objekten antworten. In vielen Themengebieten wird analysiert, wie die einzelnen Objekte oder Gruppen von Objekten zueinander stehen. Hier sind zwei Beispiele:

Equals (): - Gibt 1 zurück, falls zwei Objekte äquivalent sind (das Innere und der Rand eines Objekts liegen auf dem Rand des zweiten Objekts und das zweite bedeckt das erste Objekt)

Touches (): - Gibt 1 zurück, falls zwei Objekte sich berühren (die Ränder schneiden sich, nicht aber das Innere der beiden Geometrien)

2.2.3 Methoden für Analyse

Diese Methoden gehen im mathematischem Sinne etwas weiter als ein Vergleich zweier Objekte. Hier wird eine komplexere Analyse durchgeführt, wie z.B. eine Ermittlung der Entfernung zwischen zwei Objekte. Zwei Beispiele sind unten eingeführt:

Distance (): Double – gibt die Kürzeste Entfernung zwischen zwei beliebige Punkte zweier Objekte zurück.

Difference (): Geometry – gibt eine Geometrie zurück, die eine Differenz zweier Objekte darstellt.

2.2.4 OGS-konforme Funktionen von MS SQL Server

In dieser Bachelorarbeit wurde für Performance Tests die SQL Server des Unternehmens Microsoft benutzt, der nativ OGC-konforme Methoden unterstützt (seit Version 2008). SQL Server hat 4 Klassen, die Methoden für geografische und geometrische Daten anbieten. Das sind sowie eigene, als auch OGC-konforme Methoden. Die OGC-konfome Methoden fangen mit Präfix ST an, zum Beispiel STContains (liefert „true“ zurück, falls eine Geometrie eine andere vollständig enthält).

2.3 Dimensionsmatrix im 9-Intersection Model

Um die Analyse so zu gestalten, dass jeder Teil eines Objektes mit jedem Teil des anderen Objektes auf mögliche topologische Beziehungen geprüft werden kann, wurden drei prinzipiell neue Modellen eingeführt. Bei der Berechnung der Beziehungen werden paarweise disjunkte Mengen aus Objektteilen (innere Fläche, Kante und äußere Fläche) gebildet und auf Überschneidungen untersucht.

Der Hauptansatz des ersten Modells ist die Abbildung der beiden Objekte auf eine 2D-Fläche und der darauffolgende paarweise Vergleich von Rändern und inneren Flächen beider Objekte miteinander. Für jeden Schnitt hat man zwei mögliche Ergebnisse, \emptyset und $-\emptyset$, wobei \emptyset für ein leeres und $-\emptyset$ für ein nicht leeres Ergebnis steht. Diese Abbildung ist als eine 2x2 Matrix (Tabelle 1) darstellbar und heißt **4-intersection Modell (4IM)**.

Tabelle 1: 2x2 Matrix für 4-intersection Modell [3]

X / Y	Innere	Rand
Innere	$X(\text{innere}) \cap Y(\text{innere})$	$X(\text{innere}) \cap Y(\text{Rand})$
Rand	$X(\text{Rand}) \cap Y(\text{innere})$	$X(\text{Rand}) \cap Y(\text{Rand})$

Als Erweiterung für 4-intersection Modell wurde ein **9-intersection Modell (9IM)** (Egenhofer Operatoren) vorgeschlagen. Das neue Modell bildet sich auf der Basis von 4-intersection Modell mit zusätzlicher Beachtung von äußerer Fläche, die sich außerhalb von Rändern des Objekts befindet und stellt ein Komplement zu der inneren Fläche des Objekts dar (Tabelle 2). Anders als in dem 4-intersection Modell kann man in diesem Modell auch die Punkte betrachten, weil diese im Vergleich mit anderen Arten von Objekten keine innere Fläche besitzen.

Tabelle 2: 3x3 Matrix für 9-intersection Modell [3]

X / Y	Innere	Rand	Äußere
Innere	$X(\text{innere}) \cap Y(\text{innere})$	$X(\text{innere}) \cap Y(\text{Rand})$	$X(\text{innere}) \cap Y(\text{äußere})$
Rand	$X(\text{Rand}) \cap Y(\text{innere})$	$X(\text{Rand}) \cap Y(\text{Rand})$	$X(\text{Rand}) \cap Y(\text{äußere})$
Äußere	$X(\text{äußere}) \cap Y(\text{innere})$	$X(\text{äußere}) \cap Y(\text{Rand})$	$X(\text{äußere}) \cap Y(\text{äußere})$

Das dritte Modell stellt eine weitere Entwicklung des 9IM Modells dar und wird von OGC für die Berechnung topologischer Relationen akzeptiert. Dieses Modell heißt **Dimensionally Extended 9-intersection Modell (DE-9IM)**. Hier werden auch Dimensionen des Schnittes berücksichtigt. Das heißt, dass jeder Schnitt in diesem Modell 4

Werte annehmen kann (-1, 0, 1, oder 2); -1 bedeutet ein leeres Ergebnis und 2 eine maximale Anzahl (max = 3) von geometrischen Einheiten (Punkt, Linie, Fläche), die in Schnitt hereingezogen sind. Siehe Abbildung 5.

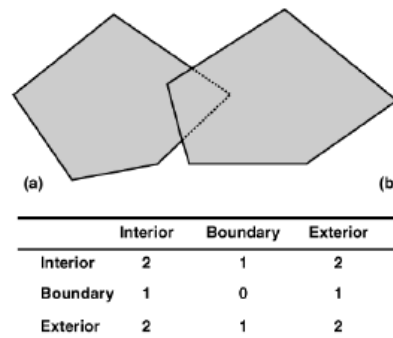


Abbildung 5: Dimensionen in DE-9IM am Beispiel von zwei Polygonen [1]

Aus dem in Abbildung gezeigten Beispiel kann man folgendes entnehmen: zwei Polygonen überschneiden sich in allen Bereichen (Ränder, Innere, Äußere) was bedeutet, dass kein Schnitt die Dimension -1 besitzt, dabei überschneiden sich die Ränder nur punktwise und besitzen somit Dimension 0 aus der Menge $\{-1, 0, 1, 2\}$. Die inneren und äußeren Flächen kann man sich als eine Menge von Punkten, Linien oder als ein Polygon vorstellen, deswegen ergibt sich bei Überschneidung die Dimension 2, die höchste aus der Menge. Genauso analysiert man auch weitere Schnitte, was in der Abbildung 5 (unten in Matrix) zu sehen ist.

Die Beziehung zwischen zwei zu vergleichenden Objekte kann man in einer einfacheren Form beschreiben als Matrix. Man nimmt ein 9-stelliges Pattern, in dem man alle Dimensionen von links nach rechts und von oben nach unten einfügt. Es ist auch möglich, dieses Pattern weiter zu vereinfachen: falls man nicht mit Dimensionen arbeiten will, ersetzt man alle Werte ab 0 bis 2 mit „T“ und den Wert -1 mit „F“. Falls auch das irrelevant ist, dann trägt man ein Joker-Zeichen „*“ für beliebige Werte ein.

Alle mögliche Dimensionen der Matrix und Werte des Patterns sind in der Tabelle 3 gelistet.

Tabelle 3: Dimensionen der Matrix und Werte des Patterns

Pattern	Dimension
$\mathbf{p = T}$	$\dim(x) \in \{0, 1, 2\}$, d.h. $x \neq \emptyset$
$\mathbf{p = F}$	$\dim(x) = -1$, d.h. $x = \emptyset$
$\mathbf{p = *}$	$\dim(x) \in \{-1, 0, 1, 2\}$, d.h. Don't Care
$\mathbf{p = 0}$	$\dim(x) = 0$
$\mathbf{p = 1}$	$\dim(x) = 1$
$\mathbf{p = 2}$	$\dim(x) = 2$

Für bessere Integration von GIS Operationen mit Programmiersprachen, wurden 5 Prädikaten eingeführt, mit denen man bestimmte Patterns ersetzen kann, wie zum Beispiel *Disjoint*, *Touches*, *Crosses*, *Within* und *Overlaps*. Diese Prädikate können auf Datenbankebene als skalare Funktionen implementiert und durch datenbankplattformeigene Funktionen erweitert werden. Ein Beispiel für einen Test, ob zwei Objekte „a“ und „b“ äquivalent sind, sähe wie folgt aus: „a.Equals(b)“ ist gleich „a.Relate(b, “TFFFTFFFT”““, wobei TFFFTFFFT ein Pattern nach dem DE-9I Modell ist. Dieses Pattern spiegelt eine Matrix wider, die von links nach rechts und von unten nach oben gelesen wird!

GIS-Operatoren, die auf der Datenbankebene realisiert sind, zeigen bessere Performance als die, die für die Frontend-Anwendungen entwickelt sind. In dieser Bachelorarbeit wird der Wert auf Leistungsaspekte gelegt und eine entsprechende Datenbanklösung gewählt, die eine Sammlung von solchen Operatoren bereits enthält.

Durch GIS-Operatoren werden RCC8-Relationen modelliert und in SQL Abfragen involviert. Dadurch wird die Möglichkeit erreicht, aus einer Ontologie mit thematischen und räumlichen Konzepten, SQL-Abfragen zu generieren. [4] [5]

3 GIS-Eigenschaften einer Datenbank

Eine GIS-Datenbank unterscheidet sich von einer gewöhnlichen relationalen Datenbank durch eine zusätzliche Funktionalität: die Verarbeitung der speziellen geometrischen und geographischen Datentypen. Sie verfügt über eine Reihe von Funktionen, die die GIS-Operatoren (s. Kapitel 2.1) implementieren. Außerdem kann eine GIS-Datenbank auch besondere Werkzeuge zur Visualisierung von GIS-Daten anbieten.

3.1 GIS Datentypen

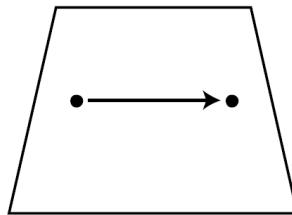


Abbildung 6: Eine Linie auf der Erdoberfläche in *Geometry*-Darstellung [1]

Viele GIS Datenbanksysteme unterstützen zwei grundlegende GIS Datentypen: *Geometry* und *Geography*. Mit *Geometry* wird oft ein Datentyp bezeichnet, der eine oder mehrere Instanzen der Klasse *Geometry* mit Namen und Koordinaten beschreibt, zum Beispiel: `line(34.023302239, 23.332020224, 32.321222985, 32.143343201)`. In diesem Beispiel bedeuten die zwei ersten Zahlen Koordinaten X und Y des Anfangspunktes der Linie und die übrigen zwei Zahlen beschreiben den Endpunkt. Ein Wert von diesem Typ spiegelt die Lage (Koordinaten) und Art (Linie) des Objektes. Dabei werden Koordinaten in einem euklidischen (flachen) Koordinatensystem dargestellt. Der *Geometry*-Datentyp eignet sich für flache Projektionen der Erdoberfläche (Abbildung 6).

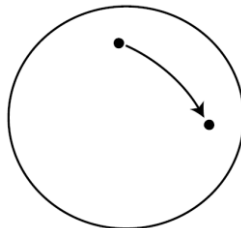


Abbildung 7: Eine Linie auf der Erdoberfläche in *Geography*-Darstellung [1]

Im Gegenteil zu *Geometry* operiert *Geography* nicht mit Projektionen von der Erdoberfläche, sondern betrachtet die Erde als Kugel, und als Koordinaten werden Längengrad und Breitengrad benutzt (Abbildung 7).

3.2 Räumliche Indizes

Ein Index in der Datenbank ist Teil einer Strategie, die dazu dient, die Leistung einer Datenbank bei Datenabfragen zu erhöhen. Die Suche nach Daten durch die gesamte Tabelle kann bei der großen Anzahl von Daten sehr lange dauern; um das zu verhindern, werden die Indizes erstellt, die aus einer oder mehreren Spalten bestehen und auf entsprechenden Zeilen in der Tabelle zeigen. Unter allen Strategien gibt es auch spezielle für räumliche Daten, wie zum Beispiel „*Filter and refine*“, B-Bäume und R-Bäume.

Bei „*Filter and refine*“ Strategie wird zunächst Filter durchgegangen, bei dem eine „grobe“ Suche stattfindet. Das heißt, es werden angenäherte geometrische Figuren benutzt, die die gesuchten Objekte umhüllen. Als Beispiel kann man MBR (*Minimum Bounding Rectangle*) nennen, der ein minimales, originale Geometrie umspannendes, Rechteck darstellt. Solche Methode ist rechnerisch günstig, weil es eine kleine Menge von relativ großen Objekten getestet wird. Nur die Kandidaten (MBRs), die Objekte enthalten, passen zum zweiten Schritt.

Auch B-Bäume und R-Bäume zählen zu Indexstrukturen; in dieser Arbeit werden aber nur R-Bäume betrachtet, da diese als räumliche Indizes in MS SQL Server implementiert sind. Dabei werden achsparallele Rechtecke zur Beschreibung der Objekte für den räumlichen Zugriff verwendet (statt der räumlichen Objekte selbst). Der R-Baum ist für mehrdimensionale Räume und Phänomene räumlicher Ausdehnung geeignet. Es ist ein höhen-bilanzierter Baum, der sich einem Binärbaum ähnelt und dessen Indexeinträge in den Blättern die Zeiger zu den realen Objekten beinhalten. Die Knoten korrespondieren zu den physikalischen Plattenpages. Er erfordert keine periodische Reorganisation. Zur Suche nach räumlichen Objekten ist nur eine kleine Anzahl von Knoten zu analysieren - im Gegensatz zu hierarchischen Baumstrukturen kann allerdings die parallele Suche in verschiedenem Asten notwendig sein [8]. Räumliche Indizes (R-Bäume), die MS SQL Server zur Verfügung stellt, werden im Kapitel 5 genauer betrachtet.

3.3 Grafische Analyse GIS-Daten

Einige GIS-unterstützende Datenbanksysteme besitzen Werkzeuge zum Visualisieren von grafischen Daten. MS SQL Server hat auch eine Visualisierungseinheit. In der Version 2008 R2 von SQL Server, kann man jede einzelne Spalte, wo sich Koordinaten von Objekten befinden auswählen und mit SQL Server Management Studio eine Karte mit allen Objekte automatisch generieren lassen. Das hilft beim Testen von Ergebnissen der SQL Abfragen: wenn zum Beispiel zwei Objekte auf Überschneidung überprüft werden, kann man dann auch auf der Karte visuell erkennen ob diese Objekte sich wirklich überschneiden. Falls Funktionalität von MS SQL Server nicht ausreichend ist, können die Daten parallel in ein anderes GIS-Werkzeug geladen und da grafisch analysiert werden.

4 MS SQL Server als Datenbanksystem für GIS

MS SQL Server hat als Datenbanksystem für GIS einige Vorteile, die hier erwähnt werden müssen:

- Die geografischen und geometrischen Datentypen sind schon in System integriert und funktionieren ohne jegliche Nachinstallation.
- Große Auswahl an eigenen und OGC-konformen Methoden, die Entwickler, zum Beispiel in SQL Server Management Studio in Abfragen anwenden kann.
- Bereits existierende Datenbanken können durch GIS Eigenschaften erweitert werden. Es ist nicht notwendig den gesamten Datenvolumen ins neue Platform zu übertragen.
- Auch kostenlose Express Version von MS SQL Server enthält die GIS Funktionalität. [1]

Tatsache, dass MS SQL Server GIS Eigenschaften erst ab Version 2008 besitzt kann als Nachteil angesehen werden. Einige Fehler wurden bis Produktionsphase leider nicht behoben. Ein Fehler ist auch im Rahmen dieser Arbeit festgestellt worden: nicht alle mögliche Mustern von Matrix aus 9-Intersection Modell werden berücksichtigt. Wenn man nur die von Microsoft vorgeschlagenen Muster benutzt, dann funktioniert alles mangelfrei. Teil dieser Arbeit war aber auch Benchmarking unterschiedlicher Modellen (naiv, seminaiv), die unterschiedlichen Muster beinhalten, durchzuführen. Dieses Fehlverhalten wurde bei Microsoft gemeldet und wird in den nächsten Versionen von MS SQL Server behoben.

Fehlerbeschreibung (Englisch):

„The area1 is a Central Park in New York and the area2 is Jefferson Park, which is far away from Central Park. They dont share any points. But if I write the query with STRelate(geom2, 'T*****') SQL Server answers TRUE (it means both Parks should share Interiors despite the fact that they can't do this).

Query:

```
SELECT area1.geom.STRelate(area2.geom,'T*****') AS Result from realm area1, realm area2
```

```
WHERE area1.fullname like '%central park%' and area2.fullname like '%Jefferson Park%'“
```

4.1 Installation von MS SQL Server

Um MS SQL Server als GIS-Datenbanksystem zu benutzen, muss die richtige Version gekauft oder heruntergeladen werden: GIS Funktionalität besitzt den Server erst seit Version 2008.

Bei der Wahl der Edition muss entschieden werden, wie man den Server benutzen wird. Wenn nur GIS Eigenschaften des Servers in Anspruch genommen werden, dann reicht auch eine kostenlose Express-Edition, wobei die maximale Größe einer Datenbank hier unter 4 Gb bleiben muss (bei Version 2008). Falls man die volle Funktionalität des Servers ausnutzen will, muss man sich die ganze Palette von Editionen anschauen: Workgroup Edition (für kleine Unternehmen), Standard Edition (4 CPU, 32/64 Bit) und Enterprise Edition¹ (vollste Funktionalität).

Wenn man für wissenschaftliche Zwecke MS SQL Server nutzen will und die Universität Zugang zu MSDNAA (MSDN Academic Alliance) hat, dann kann man auch die Developer Edition kostenlos beziehen. Beim Installieren von MS SQL Server folgt man dem SETUP-Assistenten.

Die offizielle Seite von MS SQL Server:

<http://www.microsoft.com/sqlserver/2008/en/us/default.aspx>

4.2 Werkzeuge

4.2.1 SQL Server Management Studio und Squirrel als Frontend

Zum Verwalten der Daten in MS SQL Server wurden zwei DBMS (*Database Management Systems*) eingesetzt. Die erste heißt SQL Server Management System und kommt zusammen mit MS SQL Server 2008. SQL Server Management Studio ist ein grafisches Tool mit einem SQL Editor, um alle Komponenten von MS SQL Server zu verwalten. Die Zweite ist ein Open-Source Projekt, heißt Squirrel und wurde speziell für Macbook entwickelt. Squirrel wird mit der Datenbank durch JDBC verbunden und unterscheidet sich vom SQL Server Management System durch Plattformunabhängigkeit, weil er in Java geschrieben ist.

4.2.2 FME Desktop und shape2sql als Software für Datenimport

Als Software für Import von Shape Dateien wurde zwei unterschiedliche Software-Werkzeuge benutzt. Zuerst wurde Shape2sql installiert, das Tool hat aber fehlerhaftes Verhalten gezeigt. Beim Importieren von großen Dateien wurde eine „Exception“ aus-

¹ Developer Edition und Compact Edition werden hier nicht vorgestellt. Developer Edition unterscheidet sich nur Lizenzbedingt vom Enterprise Edition und Compact Edition ist eine eingebettete Datenbank (Embedded DB).

gelöst. Ein weiterer Nachteil ist es, dass Shape2sql nicht weiterentwickelt wird. Danach wurde FME Desktop installiert und akademische Lizenz beantragt. FME Desktop ist ein sehr flexibles Programm, bei dem man nicht nur Shape, sondern auch .dbf Dateien und fast 100 anderen importieren kann. FME Desktop erlaubt den gesamten Umwandlungsprozess mit vielen Parametern zu steuern. Man kann für neue Tabellen andere Datentypen definieren. Auch wie Koordinaten in die Geometrie Datentyp kann man in diesem Programm einstellen.

4.3 Erstellung der räumlichen Datenbank

Nach der Installation des Servers kann sofort eine Datenbank angelegt werden. Einfachster Weg dazu ist SQL Server Management Studio zu öffnen und entweder grafisch durch einen bereitgestellten „Wizard“ oder mit SQL Befehlen eine zusätzliche Datenbank zu erstellen.

Da in dieser Arbeit Objekte der Stadt New-York aus Tigerdata (<http://www.census.gov>) genommen werden, empfiehlt es sich der Datenbank auch sprechende Namen zu vergeben, wie zum Beispiel „New-York“.

4.4 OGC-methoden und räumliche Indizes in MS SQL Server

In SQL Server 2008 und höheren Versionen werden räumlichen Indizes unterstützt. Hierunter fällt die Unterstützung des planaren räumliche-Daten-Typs geometry, und geography. Ein räumlicher Index wird für eine Tabellenspalte definiert, die räumliche Daten enthält (eine räumliche Spalte). Jeder räumliche Index verweist auf einen endlichen Raum. Zum Beispiel verweist ein Index für eine geometry-Spalte auf einen benutzerdefinierten rechteckigen Bereich auf einer Ebene.

In SQL Server 2008 werden räumliche Indizes mithilfe von R-Strukturen erstellt, und das heißt, dass die Indizes die zweidimensionalen räumlichen Daten in der linearen Reihenfolge der R-Strukturen darstellen müssen. Bevor Daten in einen räumlichen Index eingelesen werden, implementiert SQL Server 2008 eine einheitliche hierarchische Zerlegung des Raums. Während der Indexerstellung wird der Raum in eine vier Ebenen umfassende Rasterhierarchie zerlegt. Diese Ebenen werden als Ebene 1 (die oberste Ebene), Ebene 2, Ebene 3 und Ebene 4 bezeichnet.

Auf jeder nachfolgenden Ebene wird die ihr übergeordnete Ebene weiter zerlegt, sodass jede Zelle der übergeordneten Ebene ein vollständiges Raster der nächsten Ebene enthält. Auf einer gegebenen Ebene verfügen alle Raster an beiden Achsen über die gleiche Anzahl von Zellen (beispielsweise 4 x 4 oder 8 x 8), und die Zellen sind alle gleich groß.

In der folgenden Abbildung wird dargestellt, wie die rechte obere Zelle auf jeder Ebene der Rasterhierarchie in ein Raster der Größe 4 x 4 zerlegt wird. In Wirklichkeit werden alle Zellen auf diese Art und Weise zerlegt. Wenn beispielsweise ein Raum in vier Ebenen von 4 x 4-Rastern zerlegt wird, resultieren daraus insgesamt 65.536 Zellen auf Ebene 4.

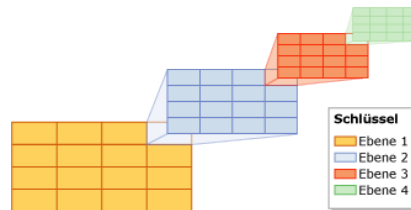


Abbildung 8: Aufbau des räumlichen Indexes

Leider nicht alle in SQL Server eingebaute Methoden unterstützen räumliche Indexes. Unter bestimmten Bedingungen unterstützen räumliche Indizes nur die folgenden mengenorientierten geometry-Methoden: STContains(), STDistance(), STEquals(), STIntersects(), STOverlaps(), STTouches() und STWithin(). Diese Methoden werden nur dann von einem räumlichen Index unterstützt, wenn sie in der WHERE-Klausel oder JOIN ON-Klausel einer Abfrage verwendet werden und in einem Prädikat allgemeinen Form stehen.

4.5 Import räumlicher Daten aus Shape-Dateien

Als Datenquelle für unsere Datenbank sind Shape-Dateien (Das Dateiformat Shapefile ist das von ESRI ursprünglich für „ArcGIS“ entwickelte Format für Geodaten) mit dem Stadt New-York gewählt, die die Internetseite <http://www.census.gov/cgi-bin/geo/shapefiles2010/main> bereitstellt.

Shape-Datei ist nur ein Teil des Konzeptes. Es sind insgesamt drei obligatorischen Dateien:

- .shp dient zur Speicherung der Geometriedaten
- .dbf Sachdaten im dBASE-Format
- .shx dient als Index der Geometrie zur Verknüpfung der Sachdaten (auch Attributdaten genannt)

Manchmal kommen dazu auch optionale, zusätzliche Dateien

- .atx Attributindex
- .sbx und .sbn Index für Tabellenverbindungen (Joins)
- .aih und .ain Index für Tabellenverknüpfungen (Links)
- .shp.xml Metadaten zur Shape-Datei
- .prj Projektion der Daten

- .cpg um den in der .dbf verwendeten Zeichensatz zu spezifizieren.

Alle Geometrischen Objekte sind in Shape-Dateien als Polygone gespeichert (SRID = 4296)

Beim Importieren von räumlichen Daten unterstützt uns Software von dritten Anbietern, weil MS SQL Server leider noch keiner Funktionalität zum Konvertieren von Shape (.shp) Dateien in den Tabellen hat. Man kann dafür sowohl kostenlose Opensource Utilliten, als auch kommerzielle Lösungen verwenden.

Kostenloses Tool:

<http://sharpgis.net/page/Shape2SQL.aspx>

Kommerzielles Tool:

<http://www.safe.com/products/fme-desktop/> (wurde bei der Vorbereitung dieser Arbeit benutzt)

4.6 Datenschema

Datenschema der Datenbank entspricht dem Datenschema von Tiger Shapefiles. In weiteren Unterkapiteln wird der Sinn, Struktur und Inhalt Tiger Shapefiles ausführlicher betrachtet.

4.6.1 Tiger/Line Daten: die Idee

TIGER/Line ((Topologically Integrated Geographic Encoding and Referencing) ist ein Geo-Datensatz von US Census Bureau, welches diverse Elemente für die gesamte USA umfasst. Die Shape-Dateien schließen Information über 50 amerikanischen Staaten ein, auch für Columbia, Puerto Rico und Inseln ein. Datenbank enthält folgende Objekte: Straßen, Eisenbahnen, Flüsse, Seen, und Landmarks, wie Schulen, Flughäfen usw. Geometrisch werden Objekte als Linien, Punkte und Polygonen dargestellt und in Dateien gespeichert.

4.6.2 Tiger Shape-Dateien: Struktur allgemein

Shape-Dateien enthalten Polygonen, die in Form von Namen der Objekten, Attributen, Koordinaten und Beziehungen zwischen Objekten gespeichert sind. Format, Attributen und Beziehungen zwischen Objekten werden in Metadaten Datei in XML Format gespeichert.

Als Beispiel führe ich „Areawater“-Datei vor. „Areawater“ ist eine Sammlung von allen Wasserobjekten, wie Seen oder Flüsse usw. , die sich in einem bestimmten Region befinden. Format der Dateinamen: **tl_2010_<Region_Bezeichnung>_areawater.shp**, wo als Regionbezeichnung dient ein FIPS Schlüssel, der über alle Daten eindeutig ist und kann, beim Import in Datenbank, als Primärschlüssel benutzt werden. Die Attribute werden als Spalten abgebildet und eine Datei wird in eine Tabelle komplett importiert.

Tabelle 4: Beispielstruktur der Shape-Datei

Attribut	Länge	Datentyp	Beschreibung
STATEFP	2	String	Eindeutige Referenz, die auf einen Bestimmten amerikanischen Staat zeigt
COUNTYFP	3	String	Eindeutige Referenz, die auf eine Bestimmte County innerhalb von Staat zeigt.
ANSICODE	8	String	Ein optionales Feld, der weist eine offizielle Referenz von Bundesagentur vor.
HYDROID	22	String	Eindeutige Referenz (kann als Primärschlüssel benutzt werden) die auf aktuelles Objekt zeigt.
FULLNAME	100	String	Name des Objektes
MTFCC	5	String	MAF/TIGER Referenz, die eine Zugehörigkeit zu den Bestimmte Gruppe der Objekten zeigt.
ALAND	14	Number	Fremdschlüssel: Land area
AWATER	14	Number	Fremdschlüssel: Water area
INTPTLAT	11	String	Koordinate: Breitengrad von Objekt
INTPTLON	12	String	Koordinate: Längengrad von Objekt

4.7 Aufbau der Datenbank nach dem Import von Tigerdaten

GIS-Datenbank „New-York“ besteht aus 63 Tabellen, darunter sind 7 Tabellen wurden mit geografischen Objekten und Beziehungen zwischen denen für Tests ausgewählt:

Tabelle 5: Tabellen mit geografischen Daten

Tabellenname	Primärschlüssel	Was wird in Tabellen gespeichert
Addr	tlid, side, arid	Adressensammlung. <i>Tabelle ohne geometrisches Datenfeld.</i>
Addrfn	arid, linearid	Beziehungen zwischen Straßenobjekten (Edges) und Adressen (Addr). <i>Tabelle ohne geometrisches Datenfeld.</i>
Arealm	Areaid	Erdbereiche als Polygonen gespeichert. <i>Tabelle mit einem geometrischen Datenfeld.</i>
Areawater	Hydroid	Hier sind Wasserbereiche als Polygonen gespeichert. <i>Tabelle mit einem geometrischen Datenfeld.</i>
Edges	countyfp, tlid	Hier sind lineare Objekte, wie Straßen, Autobahnen usw. gespeichert. <i>Tabelle mit einem geometrischen Datenfeld.</i>
Faces	Tfid	Objekte die von „Edges“ eingegrenzt sind. <i>Tabelle mit einem geometrischen Datenfeld.</i>
Vtd10	geoid10	Wahlbezirke. <i>Tabelle mit einem geometrischen Datenfeld.</i>

Die weiteren Tabellen gehören nicht zu Tiger/Line Daten und sind mit gespeicherten Prozeduren aus obengenannten sieben Tabellen generiert. Solche Tabellen heißen im Rahmen dieser Bachelorarbeit „vorkalkulierte Tabellen“ und stellen vorkalkulierte RCC-Relationen dar, die im nächsten Kapitel genauer diskutiert werden.

5 Maßnahmen zur Erhöhung der Datenbankleistung

Prozess der Datenbankoptimierung ist ein komplexes Thema, das eine große analytische Arbeit voraussetzt. Jede Optimierungsaufgabe ist individuell und muss auf eine bestimmte Datenbank angepasst werden. Außerdem gibt es auch allgemeine Maßnahmen, die bei Optimierung der Datenbank jeglicher Art wichtig sind.

Als erster Maßnahme muss das Datenbankschema analysiert werden. Falls nötig, wird dieses normiert (es werden 1/2/3/4-normale Formen verwendet). Datentypen werden anhand ihrer Größen angepasst (zum Beispiel, wenn kein Wert größer als 255 geplant ist, wird bigint durch smallint ersetzt). Falls Primärschlüssel den Datentyp String hat, wird dieser durch Integer ersetzt.

Auch Abfragen einer Datenbank sollen optimiert sein (z.B. durch einen effektiven Ausführungsplaner). Falls die Indizes noch nicht definiert sind, muss man die erstellen. Und wenn Hardware nicht genügend Arbeitsspeicher anbietet, müssen für große Datenmengen temporäre Tabellen benutzt werden.

Es muss auch nicht sofort die gesamte Ergebnismenge übermittelt werden. Die Ergebnisse werden nach Relevanz sortiert und die am besten passende in erster Reihe geliefert (Ranking, Paging, Daten-Segregation und Partitionieren)

Ausnutzung des Potentials von gespeicherten Prozeduren, die in Datenbanksystem ausgelagert sind und Datenbank schon zu der Zeit der Ausführung ihr Ausführungsplan kennt.

In dieser Arbeit wurden folgenden Optimierungsmaßnahmen untersucht und durchgeführt:

- Datenebene: In Tabellen sind, von Tiger/Line Datenspezifikation vordefinierte Datentypen für Schlüsselfelder von Strings auf Integers geändert. Weitere Änderung: es wurden räumliche Indizes angelegt und Tabellen mit vorkalkulierten RCC-Relationen.
- Abfragenebene: die Abfragen wurden so überarbeitet, damit sie so viel wie möglich Operatoren verwenden, die räumliche Indizes unterstützen.
- Logikebene: es wurden naive, seminaive Strategien untersucht, um festzustellen welche zu den optimiertesten Abfragen führt.

5.1.1 Tuning von Indizes

Tuning von räumlichen Indizes besteht darin, die richtigen Entscheidungen beim Anpassen zu treffen. Als erstes definiert man für einen Index eine Rasterdichte. Dichte wird durch Anzahl der Zellen entlang der Achsen eines Rasters bestimmt: je größer ist

die Anzahl, desto dichter wird das Raster. Beispielsweise ist ein 8 x 8-Raster (64 Zellen), dichter als ein 4 x 4-Raster (16 Zellen). Die Rasterdichte wird pro Ebene definiert. Die Experimente haben gezeigt, dass auf der untersten Ebene am wenigsten Zellen sein muss und je höher die Ebene sich befindet desto mehr Zellen muss sie enthalten.

Noch eine Einstellung ist die Anzahl von Zellen pro Objekt. Diese Regel erzwingt den Zellen-pro-Objekt-Grenzwert, der die maximale Anzahl von Zellen festlegt, die für jedes Objekt gezählt werden können. Dieser Grenzwert gilt nicht für Ebene 1. Auf tieferen Ebenen steuert die Zellen-pro-Objekt-Regel die Informationsmenge, die über das Objekt aufgezeichnet werden kann. Dieser Wert wurde im Rahmen dieser Arbeit experimentell ermittelt. Unterschiedliche Werte wurden ausprobiert und die Laufzeit der Testabfrage wurde gemessen. Das beste Ergebnis wurde bei 16 Zellen pro Objekt erreicht.

Auch ein umgebendes Feld muss man durch vier Koordinaten definieren:

- x-min ist die X-Koordinate der linken unteren Ecke des umgebenden Felds.
- y-min ist die Y-Koordinate der unteren linken Ecke.
- x-max ist die X-Koordinate der oberen rechten Ecke.
- y-max ist die Y-Koordinate der oberen rechten Ecke.

Dieses Feld muss einerseits möglichst alle Objekte enthalten und andererseits so klein wie möglich sein.

5.1.2 RCC-Relationen und benutzerdefinierte Funktion für deren Ermittlung.

Der RCC-Kalkül (*Region Connected Calculus*) basiert auf räumlichen Gebieten, die Teilmengen eines universellen topologischen Raums U sind, genannt das Universum. Gebiete werden als abgeschlossen betrachtet, d.h. sie sind äquivalent zum Abschluss ihres Inneren, anders ausgedrückt: jedes Gebiet hat einen Rand und dieser wird als Bestandteil des Gebiets betrachtet. Ein Gebiet kann aus mehreren nicht zusammenhängenden Teilen bestehen. Räumliche Variable, bezeichnet mit X , Y , Z usw. beziehen sich auf den ganzen topologischen Raum. Beziehungen zwischen Gebieten werden durch Relationen $C(X, Y)$ definiert. $C(X, Y)$ ist wahr, wenn die Gebiete X und Y mindestens einen Punkt gemeinsam haben. [9]

Hier ist eine Tabelle, die Beziehungen zwischen Gebieten darstellt:

Tabelle 6: RCC-Relationen

Bezeichnung	Notation
DisConnected	DC
Externally Connected	EC
Partial Overlap	PO
Equal	EQ
Tangential Proper Part	TPP
Non- Tangential Proper Part	NTPP
Inverse of Tangential Proper Part	TPP ⁻¹
Inverse of Non-Tangential Proper Part	NTPP ⁻¹

Im Rahmen dieser Arbeit, wurde folgende benutzerdefinierte Funktion erstellt, mit deren Hilfe eine Bestimmung der RCC-Relation zwischen zwei Geometrien möglich war:

```

1. FUNCTION [dbo].[rccrelation] (@r1 geometry, @r2 geometry)
2. returns smallint
3.
4. AS
5. BEGIN
6.
7. DECLARE @res smallint = 0;
8.
9. if (@r1.Filter(@r2) = 1)
10. BEGIN
11.
12.     if @r1.STTouches(@r2) = 1
13.     SET @res = 2 --ec
14.     ELSE if @r1.STEquals(@r2)= 1
15.     SET @res = 1 --eq
16.     ELSE if @r2.STContains(@r1)=1 AND @r1.STRelate(@r2,
17.     '2FF1FF212')=1
18.     SET @res = 3 --ntpp
19.     ELSE if @r1.STContains(@r2)=1 AND @r1.STRelate(@r2,
20.     '212FF1FF2')=1
21.     SET @res = 4 --ntppi
22.     ELSE if @r1.STOverlaps(@r2)= 1
23.     SET @res = 5 --po
24.     ELSE if @r1.STWithin(@r2)=1 AND NOT @r1.STEquals(@r2)=1 AND
25.     NOT @r2.STRelate(@r1, 'T**FF*FF*')=1
26.     SET @res = 6 --tpp
27.     ELSE if @r1.STContains(@r2)=1 AND NOT @r1.STEquals(@r2)=1 AND
28.     NOT @r1.STRelate(@r2, 'T**FF*FF*')=1
29.     SET @res = 7 --tppi
30. END
31. RETURN(@res)
32. END

```

Hier ist eine Erklärung zur Implementierung dieser Funktion. Als eingehende Parametern sind zwei zu vergleichende Geometries definiert. Im ersten Schritt beim Ausführung dieser Funktion, wird eine SQL Server eigene Methode Filter angewendet (Zeile 9), die dazu dient, alle Geometries, die eine „Disjoint“-Beziehung (oder laut Tabelle 6 - DC, *DisConnected*) haben, auszufiltern. Da „Filter“-Methode den räumlichen Index unterstützt, wird diese Methode der anderen (STDisjoint) bevorzugt.

Als weiterer Schritt werden die Beziehungen, nach dem ersten Schritt gebliebenen Geometries anhand RCC Regeln ermittelt und das Ergebnis wird dem Ausgangswert zugewiesen.

In Zeile 13 wird geprüft, ob EC (*Externally Connected*) Relation für eingegebenen Geometries stimmt. Das kann mit STTouches() Methode durchgeführt werden. STTouches() unterstützt räumlichen Index. Ähnliches Vorgehen wird auch mit EQ (STEquals(), Zeile 14) und PO (STOverlaps(), Zeile 20) verwendet. Aber für NTPP, NTPPI, TPP und TPPI gibt es leider keine Methoden in SQL Server die nur als einzige Befehl den Logik des Relation abdecken könnte, es muss eine Reihe von Methoden und logischen Verknüpfungen definiert werden, was auch zum Leistungsverlust führt, weil nicht alle verknüpfte Methoden den räumlichen Index unterstützen.

Diese benutzerdefinierte Funktion wird mit eine gespeicherte Prozedur benutzt, die für die Vorbereitung der vorkalkulierten Tabellen verantwortlich ist. Diese Prozedur wird im nächsten Kapitel eingeführt.

5.1.3 Erstellung von Tabellen mit vorberechneten RCC-Relationen

Tabellen wie m_rcc_* werden als von uns definierte zusätzliche Indizes oder Cache benutzt. In diesen Tabellen werden alle Relationen nach der Vorbereitung mit einer gespeicherten Prozedur gespeichert. Es kann sein, dass die Abfrage eine Beziehung in vorkalkulierten Tabellen nicht finden kann, dann kann die Suche nach dieser bestimmten Beziehung direkt in ursprünglichen Tabellen vorgelegt werden.

Tabellen wie m_rcc_* werden folgendermaßen aufgebaut: sie enthalten Spalten id1, id2, rcc, wo id's sind id's sind Referenzen von Objekten, die in einem in Spalte rcc beschrieben Verhältnis zueinander stehen, und zwar in der Richtung id1 -> id2. Falls man die Richtung umkehren will, braucht man keine neue Tabellen zu erstellen, man muss lediglich die entsprechende inverse rcc in einer separaten Anfrage definieren. Als Beispiel nehmen wir „ntpp“: aus der Tabelle m_rcc_arealm nimmt man id1 und id2 und sieht das rcc-spalte den Wert „ntpp“ hat, dann weißt man genau, dass id2 -> id1 Verhältnis genau „ntppi“ ist. Dadurch vermeiden wir enormes Wachstum von unserer Datenbank.

Die DC Relationen werden in die Tabellen nicht aufgenommen, da es die am meisten vorkommende Relation, diese Maßnahme dient der Verkleinerung von Datenbank und Erhöhung der Performance.

Erstellt werden die Tabellen mit einer gespeicherten Prozedur, deren Implementierung hier kurz beschrieben wird. Zunächst werden Variablen deklariert. Darunter auch Cursor, mit dem die Ergebnisse zeilenweise verarbeitet werden.

```
@table1 nvarchar(2000),  
@table2 nvarchar(2000),  
@cursor bit
```

Als nächstes werden die vorkalkulierte Tabellen mit SQL Anweisung „CREATE“ erstellt:

```
CREATE TABLE m_eq_' + @table1 + '_' + @table2 + 'c (ID1 varchar(22) NOT  
NULL, ID2 varchar(22) NOT NULL)
```

Dann wird Cursor-Schleife implementiert:

```
SELECT @iNextRowId = MIN(cast('+@id1+' as bigint))  
FROM '+@table1+'
```

```
SELECT @iCurrentRowId = cast('+@id1+' as bigint),  
@GEOM_c = GEOM  
FROM '+@table1+'  
WHERE cast('+@id1+' as bigint) = @iNextRowId
```

```
WHILE @iLoopControl = 1
```

Danach werden mit Hilfe der Funktion `dbo.rccrelation()`, abhängig von RCC-Relation, die Tabellen mit Daten gefüllt:

```
INSERT INTO m_ec_' + @table1 + '_' + @table2 + 'c (ID1, ID2)  
SELECT @iCurrentRowId as id1, area2.' + @id2 + ' as id2  
FROM ' + @table2 + ' AS area2  
WHERE dbo.rccrelation(@GEOM_c, area2.geom) = 2
```

Abschließend werden die Indizes erstellt:

```
ALTER TABLE m_eq_' + @table1 + '_' + @table2 + 'c WITH NOCHECK ADD  
CONSTRAINT PK_' + @table1 + '_' + @table2 + 'c_pre_idx1 PRIMARY KEY  
CLUSTERED (id1,id2);
```

6 Abfragen an GIS Datenbank und Performancetests.

6.1.1 Test der verschiedenen Implementierungen der RCC-Relationen

Es wurde insgesamt vier verschiedene Arten von Tests durchgeführt. Für jede Art von Mapping wurde in dem Sever für Jede Abfrage ein Test durchgeführt.

NAIVE – Naive Mapping wurde mit genau definiertem DE-9I-Code Befehl `STRelate()` (für SQL Server) durchgeführt. Das heißt, für jede Stelle in gibt Matrix es einen Wert aus der Array , F‘ , ,T‘ , ,0‘ , ,1‘ , ‘2‘

SEMINAIVE - Im seminaiven Mapping wurde aus OGC bekannte Befehle benutzt, die genau definieren was geschehen werden muss, und zwar ohne DE-9I-matrix zu benutzen. Beispiel: `STDisjoint` (für SQL Server). Wie es schon geschrieben wurde, sind diese Befehle nicht so aussagenkräftig wie `STRelate`, dafür unterstützen sie spatial Indizes, was uns einen Gewinn an Performance bringt, denen wir nicht vernachlässigen dürfen.

CSE – Es wird das gleiche Model wie für naive Mapping benutzt, mit einer Ausnahme, dass es jetzt für die Stellen, deren Wert den Mapping nicht beeinflusst, das Symbol Stern (Wildcard) benutzt wird.

PREINDEXED – In diesem Fall werden Tabellen (`m_rcc_*`) mit verkalkulierten Werten benutzt um noch mehr Optimierung anzubieten. Für Vollständigkeit der Daten in vorkalkulierten Tabellen sorgen die Triggern, die Änderungen im geografischen Datenbestand registrieren und automatisch neue Datensätze in vorkalkulierte Tabellen einfügen (noch nicht implementiert).

Da es schon beim ersten Testen bestätigt wurde, dass der SEMINAIVE Ansatz (zusammen mit PREINDEXED oder als alleinige Verfahren) am meisten Performance bringt, wurden weitere Tests mit NAIVE und CSE Ansätze vernachlässigt. Test INDEX-basierter Tabellen und Tabellen ohne INDEX

6.2 Tests mit LUBM-basierte Abfragen

LUBM ist ein Benchmark, der das Benchmarking von Semantik Web Wissensbasierten Systemen in OWL Applikationen ermöglicht. LUBM bietet 14 Metriken, die als Abfragen in LUBM-Style formuliert werden können.

Hier ist ein Beispiel für solche Abfragen, die in SQL-ähnlicher Sprache „Sparql“ geschrieben wurde:

```

SELECT ?X ?Y WHERE
{
  ?X rdf:type VotingDistrict2000.
  ?Y rdf:txpe VotingDistric200.
  ?*X ec ?*Y
}

```

Diese Abfrage bedeutet: Alle Wahlbezirke aus der Datenbank auswählen, die als RCC-Relation EC aufweisen. Um Performance von „New-York“ Datenbank zu testen, wurden folgende Abfragen in SQL Sprache übersetzt und auf zwei Arten von Daten angewendet. Die erste SQL Abfrage bezieht sich immer auf GIS-basierte Tabellen mit Tiger/Line Data und die zweite auf Tabellen mit vorkalkulierten RCC-Relationen (Relationen bestehen ebenfalls aus Tiger/Line Daten).

Tabelle 7: Testergebnisse

Beschreibung		
LUBM (Sparkl) Abfragen	SQL Abfragen	Ausführungszeit (Mittelwert, 6 Ausführungen):
LUBM-Abfrage 1a: Eine Anfrage mit hohem Input und hoher Selektivität. Alle Votingdistricts, die einen bestimmten berühren (ec) ein existierendes Wahlbezirk und ein ausgedachtes Wahlbezirk (nicht vorkalkuliert, Bezirk existiert)		
SELECT ?X WHERE {?X rdf:type VotingDistrict2000. ?*X:ec *vtd4711}	SELECT auxBezirk1.GEOID10 FROM _vtd10 as auxBezirk1, _vtd10 as auxBezirk2 WHERE auxBezirk1.geom.STTouches(auxBezirk2.geom)=1 AND auxBezirk2.GEOID10 = '36001103' ORDER BY auxBezirk1.GEOID10	546 ms ²
vorkalkuliert, Bezirk existiert		
	SELECT ID1 FROM dbo.m_ec_vtd10_vtd10 WHERE ID2 = '36001103' ORDER BY ID1	23 ms
nicht vorkalkuliert, Bezirk NICHT existiert		
	SELECT auxBezirk1.GEOID10 FROM _vtd10 as auxBezirk1, _vtd10 as auxBezirk2 WHERE auxBezirk1.geom.STTouches(auxBezirk2.geom)=1 AND auxBezirk2.GEOID10 = '36001103232342433' ORDER BY auxBezirk1.GEOID10	0.1 ms
vorkalkuliert (Bezirk NICHT existiert)		
	SELECT ID1 FROM dbo.m_ec_vtd10_vtd10 WHERE ID2 = '36001103232342433' ORDER BY ID1	16 ms
LUBM-Abfrage 1c: Eine Anfrage mit hohem Input und geringerer Selektivität. Alle Votingdistricts, die einen bestimmten nicht berühren (dc) ein existierendes Wahlbezirk und ein ausgedachtes Wahlbezirk (nicht vorkalkuliert)		

² Es geht um CPU Zeit (reine Server-Zeit, unabhängig vom Client)

SELECT ?X WHERE {?X rdf:type VotingDistrict2000. ?*X :dc *vtd36061386}	SELECT auxBezirk1.GEOID10 FROM _vtd10 as auxBezirk1, _vtd10 as auxBezirk2 WHERE auxBezirk1.geom.STDisjoint(auxBezirk2.geom)=1 AND auxBezirk2.GEOID10 = '36001103' ORDER BY auxBezirk1.GEOID10	901 ms
vorkalkuliert (teilweise), Bezirk existiert		
	SELECT auxBezirk1.GEOID10 FROM _vtd10 as auxBezirk1 WHERE auxBezirk1.GEOID10 <> '36001103' AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_ec_vtd10_vtd10 WHERE ID2 = '36001103') AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_eq_vtd10_vtd10 WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_ntpp_vtd10_vtd10 WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_ntppi_arealm_arealm WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_tpp_arealm_arealm WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_tppi_arealm_arealm WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_po_arealm_arealm WHERE ID2 = '36001103') ORDER BY auxBezirk1.GEOID10	40 ms
nicht vorkalkuliert, Bezirk NICHT existiert		
	SELECT auxBezirk1.GEOID10 FROM _vtd10 as auxBezirk1, _vtd10 as auxBezirk2 WHERE auxBezirk1.geom.STDisjoint(auxBezirk2.geom)=1 AND auxBezirk2.GEOID10 = '36001103' ORDER BY auxBezirk1.GEOID10	982 ms
Vorkalkuliert, Bezirk NICHT existiert		
	SELECT auxBezirk1.GEOID10 FROM _vtd10 as auxBezirk1 WHERE auxBezirk1.GEOID10 <> '36001103' AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_ec_vtd10_vtd10 WHERE ID2 = '36001103') AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_eq_vtd10_vtd10 WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_ntpp_vtd10_vtd10 WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_ntppi_arealm_arealm WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_tpp_arealm_arealm WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_tppi_arealm_arealm WHERE ID2 = '36001103')AND auxBezirk1.GEOID10 NOT IN (SELECT ID1 FROM dbo.m_po_arealm_arealm WHERE ID2 = '36001103') ORDER BY auxBezirk1.GEOID10	31 ms
LUBM-Abfrage 2a: Erhöhte Komplexität und Dreiecksmuster. Der Input müsste hoch sein, die Selektivität auch. Nicht vorkalkuliert		
SELECT ?X,?Y,?Z WHERE {?X rdf:type VotingDistrict2000 . ?Y rdf:type LakePond . ?Z rdf:type NationalParkService . ?*X :ec ?*Z . ?*Z :tppi ?*Y . ?*X :ec ?*Y .	SELECT LakePond.HYDROID, NPS.AREAID, VTD.GEOID10 FROM [New-York full].[dbo].[_areawater_LakePond] as LakePond, [New-York full].[dbo].[_arealm_NationalParkService] as NPS, _vtd10 as VTD WHERE VTD.GEOM.STTouches(NPS.GEOM)=1 AND VTD.GEOM.STTouches(LakePond.GEOM)=1 AND (LakePond.GEOM.STContains(LakePond.GEOM)=1 AND NOT LakePond.GEOM.STEquals(LakePond.GEOM)=1 AND NOT LakePond.GEOM.STRelate(LakePond.GEOM, "T**FF*FF*")=1)	13073 ms
vorkalkuliert		
	SELECT va.ID1, aw.ID1, vw.ID2 FROM m_ec_vtd10_arealm as va,	33 ms

	<pre>m_tppi_arealm_areawater as aw, m_ec_vtd10_areawater as vw WHERE (va.ID2 = aw.ID1 AND aw.ID2=vw.ID2 AND vw.ID1=va.ID1) AND aw.ID1 in (SELECT AREAID FROM [New-York full].[dbo],[_arealm_NationalParkService]) AND vw.ID2 in (SELECT HYDROID FROM [New-York full].[dbo],[_areawater_LakePond])</pre>	
LUBM-Abfrage 2b: Wie in 2a, allerdings unterschiedliche RCC-Beziehung zwischen ?X und ?Y. Nicht vorkalkuliert		
<pre>SELECT ?X,?Y, ?Z WHERE {?X rdf:type VotingDistrict2000 . ?Y rdf:type LakePond . ?Z rdf:type NationalParkService . **X :ec **Z . **Z :tppi **Y. **X :dc **Y . }</pre>	<pre>SELECT LakePond.HYDROID, NPS.AREAID, VTD.GEOID10 FROM [New-York full].[dbo],[_areawater_LakePond] as LakePond, [New-York full].[dbo],[_arealm_NationalParkService] as NPS, _vtd10 as VTD WHERE VTD.GEOM.STTouches(NPS.GEOM)=1 AND VTD.GEOM.STDisjoint(LakePond.GEOM)=1 AND (LakePond.GEOM.STContains(LakePond.GEOM)=1 AND NOT LakePond.GEOM.STEquals(LakePond.GEOM)=1 AND NOT LakePond.GEOM.STRelate(LakePond.GEOM, "T**FF*FF*")=1)</pre>	12121 ms
vorkalkuliert		
	<pre>SELECT va.ID1, va.ID2, aw.ID2 FROM m_ec_vtd10_arealm as va, m_tppi_arealm_areawater as aw WHERE va.ID2 = aw.ID1 AND va.ID1+'+'+aw.ID2 NOT IN (SELECT ID1+'+'+ID2 FROM m_ec_vtd10_areawater UNION SELECT ID1+'+'+ID2 FROM m_eq_vtd10_areawater UNION SELECT ID1+'+'+ID2 FROM m_po_vtd10_areawater UNION SELECT ID1+'+'+ID2 FROM m_ntpp_vtd10_areawater UNION SELECT ID1+'+'+ID2 FROM m_ntppi_vtd10_areawater UNION SELECT ID1+'+'+ID2 FROM m_tpp_vtd10_areawater UNION SELECT ID1+'+'+ID2 FROM m_tppi_vtd10_areawater) AND aw.ID1 in (SELECT AREAID FROM [New-York full].[dbo],[_arealm_NationalParkService]) AND aw.ID2 in (SELECT HYDROID FROM [New-York full].[dbo],[_areawater_LakePond])</pre>	
LUBM-Abfrage 3a: ähnlich wie für 1a), nur alle hydrographik Objekten (appendix F in Tiger Docu) explizit definieren (Beispiel, dass es nicht optimal ist). Nicht vorkalkuliert.		
	<pre>SELECT auxWater.HYDROID FROM (SELECT * FROM [New-York full].[dbo],[_areawater_LakePond] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_Connector] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_SwampMarsch] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_Reservoir] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_TreatmentPond] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_BayEstuaryGulfSound] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_OceanSee] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_GravelPitQuarry] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_Glacier] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_StreamRiver] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_BraidedStream] UNION ALL SELECT * FROM [New-York full].[dbo],[_areawater_CanalDitchAqueduct]) as auxWater, _vtd10 as auxBezirk WHERE auxBezirk.geom.STTouches(auxWater.geom)=1 AND auxBezirk.GEOID10 = '3600114' ORDER BY auxBezirk.GEOID10</pre>	0.01 ms
vorkalkuliert		

	<pre>SELECT ID2 FROM m_ec_vtd10_areawater WHERE id1 = '3600114'</pre>	0.1 ms
Hier Anfragen zu Parks mit Seen, mit denen wir die Umformulierungen bzgl. einer Ontologie und verschiedene Optimierungen testen:		
	<pre>SELECT auxPark.AREAID FROM (select * from _arealm where MTFCC='K2180' or MTFCC = 'K2181') as auxPark, (select * from _areawater as water where water.MTFCC = 'H2030') as auxLake WHERE auxLake.geom.STWithin(auxPark.geom) = 1 AND NOT (auxPark.geom.STEquals(auxLake.geom) = 1)</pre>	672160 ms
q(x) <- Park(x) und Parwkwithlake und Votingdistrict(y) und ec(x*,y*)		
	<pre>SELECT AREAID, vtd.GEOID10 from _arealm, _areawater, (select * from _vtd10) as vtd where (_arealm.MTFCC='K2180' or _arealm.MTFCC = 'K2181') and _areawater.MTFCC = 'H2030' and _areawater.geom.STWithin(_arealm.geom) = 1 AND NOT (_arealm.geom.STEquals(_areawater.geom) = 1) and vtd.GEOM.STTouches(_arealm.geom) = 1</pre>	727914 ms
q(x) <- Park(x) and ParkWithLake(x) -- M-Entfaltung, seminaives DE-9IM, semantisch nicht optimiert, ohne vorberechnete RCC-Relation		
	<pre>SELECT DISTINCT 'alm(' + auxPark.AREAID + ')' from (select AREAID from _arealm where MTFCC='K2180') as auxPark, (select Distinct AREAID, HYDROID from _arealm as land, _areawater as water where land.MTFCC= 'K2180' and water.MTFCC = 'H2030') as auxHasLake, (select AREAID, geom from _arealm) as auxLoc1, (select HYDROID, geom from _areawater) as auxLoc2, (select area1.geom as ageom1, area2.geom as ageom2 from _arealm as area1, _areawater as area2 where area2.geom.STWithin(area1.geom) = 1 and not area1.geom.STEquals(area2.geom) = 1) as auxTppOrNtpp WHERE auxPark.AREAID = auxHasLake.AREAID and auxHasLake.HYDROID = auxLoc2.HYDROID and auxPark.AREAID = auxLoc1.AREAID and auxTppOrNtpp.ageom2.STAsText() = auxLoc2.geom.STAsText() and auxTppOrNtpp.ageom1.STAsText() = auxLoc1.geom.STAsText();</pre>	Länger als 10 min.
q(x) <- Park(x) and ParkWithLake(x) --- optimiert, vorberechnete RCC-Tabelle		
	<pre>SELECT ID1 FROM dbo.m_ntppi_arealm_areawater as ntppi, _arealm, _areawater where ID1 = AREAID and ID2 = HYDROID and (_arealm.MTFCC='K2180' or _arealm.MTFCC = 'K2181') and _areawater.MTFCC = 'H2030'</pre>	0.02 ms
LUBM-Abfrage 12: Die 12. LUBM-Anfrage soll die Realisation von Instanzen durch die Ontologie testen, d.h. sie testet, ob Instanzen eines Konzeptes (Chair), die nicht explizit in den Daten als Instanzen dieses Konzepts genannt sind, durch Inferenz aus der Ontologie als solche erkannt werden.		
<pre>SELECT ?X, ?Y WHERE {?X rdf:type ParkWithLake. ?Y rdf:type AreaLandmark. ?*X :ec ?*Y. ?*Y :ec *alm4711.}</pre>	<pre>SELECT auxParkHasLakeX.AREAID as X, auxArealmY.AREAID as Y from (select AREAID, GEOM from _arealm where AREAID='not exists') as auxArealmZ, _arealm as auxArealmY, (select AREAID, _arealm.GEOM from _arealm, _areawater where _arealm.MTFCC= 'K2180' and _areawater.MTFCC = 'H2030' and (_arealm.GEOM.STContains(_areawater.GEOM)=1) and _arealm.GEOM.STEquals(_areawater.GEOM)=0) as auxParkHasLakeX WHERE auxArealmY.GEOM.STTouches(auxParkHasLakeX.GEOM)=1 and auxArealmY.GEOM.STTouches(auxArealmZ.GEOM)=1</pre>	0.01 ms
Vorkalkuliert, Objekt nicht nicht existiert		
	<pre>SELECT ID1 as X, ID2 as Y FROM m_ec_arealm_arealm WHERE (ID1 in (</pre>	0,03 ms

	<pre> SELECT ID1 FROM m_ntppi_arealm_areawater WHERE ID2 in (SELECT hydroid FROM _areawater WHERE MTFCC = 'H2030') AND ID1 in (SELECT AREAID FROM _arealm WHERE MTFCC='K2180')) or ID1 in (SELECT ID1 FROM m_tppi_arealm_areawater WHERE ID2 in (SELECT hydroid FROM _areawater WHERE MTFCC = 'H2030') AND ID1 in (SELECT AREAID FROM _arealm WHERE MTFCC='K2180'))) AND (ID2 in (SELECT ID1 FROM m_ec_arealm_arealm WHERE ID2 = 'not exists')) </pre>	
Nicht vorkalkuliert, Objekt nicht existiert		
	<pre> SELECT auxParkHasLakeX.AREAID as X, auxArealmY.AREAID as Y from (select AREAID, GEOM from _arealm where AREAID='1101042398095') as auxArealmZ, _arealm as auxArealmY, (select AREAID, _arealm.GEOM from _arealm, _areawater where _arealm.MTFCC='K2180' and _areawater.MTFCC = 'H2030' and (_arealm.GEOM.STContains(_areawater.GEOM)=1) and _arealm.GEOM.STEquals(_areawater.GEOM)=0) as auxParkHasLakeX WHERE auxArealmY.GEOM.STTouches(auxParkHasLakeX.GEOM)=1 and auxArealmY.GEOM.STTouches(auxArealmZ.GEOM)=1 </pre>	624 ms
Vorkalkuliert, Objekt nicht existiert		
	<pre> SELECT ID1 as X, ID2 as Y FROM m_ec_arealm_arealm WHERE (ID1 in (SELECT ID1 FROM m_ntppi_arealm_areawater WHERE ID2 in (SELECT hydroid FROM _areawater WHERE MTFCC = 'H2030') AND ID1 in (SELECT AREAID FROM _arealm WHERE MTFCC='K2180')) or ID1 in (SELECT ID1 FROM m_tppi_arealm_areawater WHERE ID2 in (SELECT hydroid FROM _areawater WHERE MTFCC = 'H2030') AND ID1 in (SELECT AREAID FROM _arealm WHERE MTFCC='K2180'))) AND (ID2 in (SELECT ID1 FROM m_ec_arealm_arealm WHERE ID2 = '1101042398095')) </pre>	16 ms
LUBM-Abfrage 15, Geometry rausgeben		
SELECT ?X ?*X WHERE {?X rdf:type Ar- ealandmark. ?X loc ?*X }	<pre> SELECT _arealm.GEOM.STAsText() as 'Geometry' from _arealm SELECT _areawater.GEOM.STAsText() as 'Geometry' from _areawater </pre>	7800 ms

7 Fazit

In dieser Arbeit wurde der MS SQL Server als GIS Datenbanksystem ausführlich getestet. Dabei wurden alle Schritte ab Erwerb und Installation bis LUBM Tests durchgegangen. Installation der Development-Edition erfolgte problemlos auf einem Stationären PC mit Internetzugang und CPU Intel I5. Obwohl die Universität Lizenz für die Developer Edition besitzt, würde kostenlose Express Version hat gleiche GIS Funktionalität.

Da MS SQL Server keine Werkzeuge für Import von Shape Dateien besitzt wurde beim Import von GIS Daten einige Probleme mit kostenfreier Software festgestellt und man war gezwungen auf kostenpflichtige FME Desktop umzusteigen.

Beim Verwaltung von GIS Daten hat sich der Server sehr flexibel gezeigt, wegen zahlreiche Einstellungsmöglichkeiten und kostenloser Verwaltungswerkzeug: SQL Server Management System.

Dank Programmierbarkeit des Servers, wurde benutzerdefinierte Funktion und gespeicherte Prozeduren implementieren. Dabei wurden eigene Technologien des Servers benutzt, wie zum Beispiel „Cursor“

Für Tests wurde zwei Arten von Daten verwendet: geometrische Tabellen mit Tiger/Line Daten und vorkalkulierte Tabellen, die durch spezielles Verfahren mit benutzerdefinierte Funktion und gespeicherte Prozeduren erstellt worden sind. Als Ergebnis des Abfragen-Tests kann erheblicher Unterschied zwischen beiden Methoden festgestellt werden. Direkte Selektierung der Daten aus Tiger/Line Tabellen dauerte durchschnittlich um 300% länger als bei der Verwendung von vorkalkulierten Tabellen. Andererseits, die Erstellung der Vorkalkulierten Tabellen dauert länger und übersetzen von Abfragen, die solche Tabellen ansprechen, erfordert gewisse Erfahrung.

MS SQL Server als GIS-Datenbanksystem hat Nachteile und Vorteile, aktuelle Version kann mit Erfolg eingesetzt werden, doch ist empfohlen auf eine neuere Version zu warten, wo alle Anregungen und Wünsche der Benutzer berücksichtigt werden.

8 Literaturverzeichnis

- [1] G. Matthiessen, Relationale Datenbanken und standart-SQL, München: Addison-Wesley, 2008.
- [2] G. D. G. D. L. M. L. A. P. M. R.-M. R. R. D. Calvanese, Ontologies and Databases: DL-Lite approach, Berlin: Springer, 2009.
- [3] A. Aitchison, Beginning Spatial with SQL Server 2008, New-York: Springer-Verlag, 2009.
- [4] S. Z. a. J. Stoter, The role of DBMS in the new generation GIS architecture, New York: Springer, 2006.
- [5] OGC 06-103r4, OGC, 2011.
- [6] C. L. Z. L. G. Jun Chen, A Voronoi-based 9-intersection model for spatial relations, Geographical Information Science, 2001.
- [7] J. S. D. M. Max Egenhofer, A Critical Comparsion of the 4-Intersection and 9-Intersection Models for Spatial Relations: Formal Analysis, Orono, Buffalo: National Center for Geographic Information and Analysis.
- [8] J. Bendoraityte, R-Baum und seine Spezialisierungen: R* und R+ Baum, Leipzig: Universität Leipzig, 2004.
- [9] J. L. Fred Hamker, „Vorlesung "Künstliche Intelligenz", der RCC-Kalkül,“ TU-Chemnitz, Chemnitz, 2001.
- [10] A. Yeung und B. Hall, Spatial Database Systems: Design, Implementation and Project, Dordrecht: Springer, 2007.

Ich versichere hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe. Wörtlich übernommene Sätze und Satzteile sind als Zitate belegt, andere Anlehnungen hinsichtlich Aussage und Umfang unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist auch noch nicht veröffentlicht.