Evaluatie van Open Source Software voor het ontwikkelen van Desktop GIS-toepassingen

Benny GIESBERS

promotor :
Prof. dr. Frank NEVEN

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica afstudeerrichting databases

Abstract

De laatste jaren is de interesse in spatiale databanken en Open Source software sterk toegenomen, mede dankzij de opkomst van het internet. In deze thesis worden enkele spatiale databanken besproken. Enerzijds zijn er Oracle en ESRI's ArcSDE, twee commerciële databanken en anderzijds zijn er PostGIS en MySQL, twee Open Source databanken. Op vraag van GIM, een bedrijf in Leuven, worden voor elk van deze systemen enkele belangrijke aspecten van spatiale databanken besproken. Zo is er de Simple Features Specification, een specificatie die voorschrijft hoe geometrische types in een spatiale databank geïmplementeerd moeten worden. Daarnaast wordt er belang gehecht aan transacties en locking, iets wat van cruciaal belang is wanneer veel gebruikers dezelfde databank delen. Ook komen spatiale indexen aan bod, een „must" in grote spatiale databanken.
Aansluitend is er een performantie vergelijking tussen PostGIS en MySQL en een studie naar de uitbreidingsmogelijkheden van uDig, een platform voor de ontwikkeling van GIS applicaties. uDig is ontworpen in Eclipse en is extreem modulair opgebouwd. Het kan uitgebreid worden door het schrijven van plugins of door het gebruik van de applicatie uDig als basis voor een nieuwe GIS applicatie.

Voorwoord

Deze thesis is geschreven ter voltooiing van mijn studie Informatica - optie Databases aan de transnationale Universiteit Limburg.

Ik zou graag enkele mensen bedanken die me geholpen helpen bij het tot stand brengen van de thesis. In de eerste plaats mijn begeleider en promotor Frank Neven, voor zijn steun, zijn talloze tips en het nalezen van mijn tekst. Steven Smolders van GIM in Leuven zonder wie deze thesis nooit tot stand zou zijn gekomen. Ook wil ik mijn medestudenten bedanken voor de steun.

Tenslotte wil ik ook mijn familie, mijn vrienden en vooral mijn vriendin Jennifer bedanken voor de grote steun en het nalezen van mijn thesistekst.

Lijst van figuren

Lijst van tabellen

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 Doel

Het doel van deze thesis is om een analyse te maken van Open Source software voor het ontwikkelen en ondersteunen van spatiale databases. De thesis is uitgevoerd in opdracht van het bedrijf GIM in Leuven. Er worden getracht om twee grote vragen van GIM te beantwoorden:

- Is MySQL een goed alternatief voor PostGIS voor spatiale informatieverwerking?
- Wat houdt de GIS toepassing uDig in en in welke mate kan deze gecustomiseerd worden?

De thesis bestaat uit twee delen: een theoretisch en een praktisch gedeelte. In de literatuurstudie worden de fundamenten van Geografische Informatiesystemen besproken. In het praktische gedeelte worden in detail enkele Open Source en commerciële databanken besproken. De nadruk ligt hier op MySQL en PostGIS. Het praktische gedeelte wordt afgesloten met een studie over de uDig customisatie.

1.2 Indeling

Hoofdstuk 2 handelt over de fundamenten van geografische informatiesystemen. Eerst worden enkele belangrijke basisbegrippen uitgelegd die doorheen de thesis gebruikt zullen worden. Daarna worden de aspecten simple features specification, transacties & locking en spatiale indexen toegelicht. Dit concludeert het theoretische gedeelte.

In Hoofdstuk 3 worden enkele Open Source (PostgreSQL en MySQL) en commerciële (Oracle en ArcSDE) spatiale databanken toegelicht. Voor elk systeem worden de aspecten uit het theoretische gedeelte besproken. De nadruk ligt op de Open Source databanken PostgreSQL en MySQL. Met deze twee systemen wordt een performantie benchmark uitgevoerd.

In Hoofdstuk 5 wordt besproken in welke mate uDig gebruikt kan worden om eigen GIS applicaties te bouwen. Tenslotte wordt er een conclusie gegeven in Hoofdstuk 6.

Hoofdstuk 2

Fundamenten van GIS

2.1 Basisbegrippen

In deze Sectie worden enkele basisbegrippen uit de wereld van de spatiale databases uitgelegd die relevant zijn voor deze thesis.

2.1.1 GIS

Een GIS of **G**eografisch **I**nformatie**s**ysteem (zie [28]) is een informatiesysteem waarmee ruimtelijke of spatiale informatie over geografische objecten kan worden opgeslagen, beheerd, bewerkt, geanalyseerd en gepresenteerd. De kracht van een GIS ligt in het vastleggen, combineren, analyseren en presenteren van gegevens met een ruimtelijke component om zo informatie te verkrijgen.
De aanbieders van de voor GIS bruikbare basisgegevens vinden we bij: de overheid, de universiteiten en profit- en non-profit organisaties. Meestal moeten de gegevens eerst worden bewerkt om in een GIS gebruikt te kunnen worden. Gescande kaarten en luchtfoto's moeten getransformeerd worden naar het gebruikte referentiesysteem (of coördinatensysteem). Soms zijn de coördinaten al toegevoegd of geregistreerd tijdens het meetproces met behulp van GPS. Het is bij een GIS van groot belang te weten in welk referentiesysteem (coördinatenstelsel) er wordt gewerkt.
Vaak moeten data of attribuutwaarden worden gekoppeld aan een geografische positie of aan een geografisch object. We noemen dit proces ook wel geocoderen. Eens de spatiale date in een GIS is opgeslagen, kunnen er interessante queries uitgevoerd worden. Bijvoorbeeld, „Welke Vlaamse gemeenten liggen binnen een straal van 50km van Leuven?" of „Geef het deel van België en Nederland met een ligging onder zeeniveau". De eerste query combineert geometrische gegevens (de polygonen die de gemeenten voorstellen) met al-

fanumerieke attributen (de namen van de gemeenten). De tweede query gebruikt enkel geometrische data.
GIS heeft, naast het „in kaart brengen" (geocoderen) van gegevens met een ruimtelijke component, ook nog toepassingen in verscheidene andere vakgebieden. Zo komt men GIS tegen in combinatie met Computer Aided Design (CAD), Remote Sensing (RS), Automated Mapping (AM) en Facility Management (FM), Land Information Systems (LIS), Ruimtelijke Informatie Systemen (RIS) en de routeplanners.

2.1.2 Open Source Software

Open Source software (zie [29] en [17]) is software met twee basiskenmerken:

- De broncode van de software is vrij beschikbaar.
- In het licentiemodel is het intellectueel eigendom en het (her)gebruik van de software en bijbehorende broncode dusdanig geregeld dat de licentienemer de broncode mag inzien, gebruiken, verbeteren, aanvullen en distribueren.

Een Open Source licentie dwingt vaak af dat de broncode van het product vrij beschikbaar moet zijn. Bovendien dwingen veel Open Source licenties af dat software die afgeleid is van Open Source software of software die een gemodificeerde vorm is van Open Source software, zelf ook beschikbaar gemaakt moet worden onder dezelfde Open Source licentie. Als de licentie bepaalt dat de broncode vrij beschikbaar moet zijn, dient de broncode van afgeleide of gemodificeerde software ook vrij beschikbaar te zijn. Om duidelijk te maken wanneer software Open Source software genoemd mag worden, zijn door de Open Source Initiative voorwaarden[1] opgesteld waaraan een licentie moet voldoen, opdat de software die onder die licentie vrijgegeven wordt, Open Source software genoemd mag worden.
Deze vrijheid om de software aan te passen heeft ertoe geleid dat belanghebbenden gezamenlijk werken om de software te verbeteren of uit te breiden, zonder dat eigendomskwesties deze samenwerking in de weg zitten. Deze nieuwe vorm van samenwerking, waarbij personen uit verschillende organisatie, landen of op persoonlijke titel gezamenlijk de software verder ontwikkelen, wordt de Open Source ontwikkelmethode genoemd.

[1] zie http://www.opensource.org/docs/definition.php

2.1.3 Open Geospatial Consortium

Het OGC of Open Geospatial Consortium (of vroeger ook het Open GIS Consortium genoemd) is een wereldwijde non-profit organisatie die zich bezighoudt met het vastleggen van specificaties betreffende geospatiale services en uitwisseling en verwerking van GIS data. Het bestaat uit meer dan 380 commerciële, gouvernementele, non-profit en onderzoeksorganisaties verspreid over de ganse wereld. De belangrijkste OGC specificaties zijn:

- Geography Markup Language (GML)
- Simple Features for SQL (SFS)
- Catalog Service for the Web (CS-W)
- Web Coverage Service (WCS)
- Web Feature Service (WFS)
- Web Map Service (WMS)

De „Simple Features Specification for SQL" is van groot belang in deze thesis en komt aan bod in Sectie 2.2 en Subsectie 3.2.1, 3.3.1, 3.4.1 en 3.5.1.

2.1.4 Eclipse

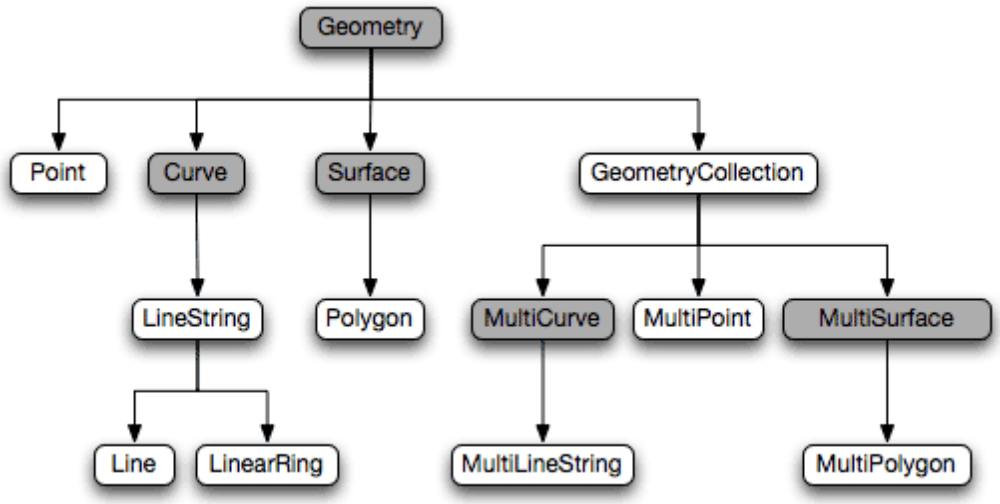
Eclipse is een Open Source platform voor integratie van tools en is ontwikkeld door IBM. Het is gratis te downloaden op het web (zie [7]). De bekendste toepassing van Eclipse is de Java ontwikkelomgeving. Voor veel mensen is die ontwikkelomgeving synoniem aan Eclipse, maar het is veel meer dan dat. Eclipse is een raamwerk met een gelaagde architectuur. Er is een basis die een plugin-infrastructuur definieert en een grote hoeveelheid reeds bestaande plugins. Op de bestaande plugins kunnen softwareontwikkelaars uitbreidingen bouwen in de vorm van nieuwe plugins, en op deze wijze kunnen allerlei verschillende tools met Eclipse gebouwd worden.

2.2 Simple Features Specification

In deze Sectie wordt de Simple Features Specification toegelicht.

2.2.1 Inleiding

De „Simple Features Specification for SQL" wordt gebruikt als basis voor de implementatie van de meeste GIS-functies in SQL-gebaseerde relationele databasesystemen. De specificatie is vastgelegd door het OGC (zie Sectie 2.1.3) en specificeert de opslag van geografische data met spatiale en niet-spatiale attributen. Deze geografische data kan verschillende geometriën voorstellen, gebaseerd op een 2D geometrie met lineaire interpolatie tussen vertices. De *SQL Geometry Type Hierarchy* is te zien in Figuur 2.1.



Figuur 2.1: SQL Geometry Type Hierarchy

De root, Geometry en de subtypes Curve, Surface, GeometryCollection, MultiSurface en MultiCurve zijn abstract en kunnen dus niet geïnstantieerd worden. De andere types Point, LineString, Polygon, MultiPoint, MultiLineString en MultiPolygon zijn wel instantieerbaar. Elk geometrisch object is geassocieerd met een *Spatial Reference System*, die de coördinaatruimte van het object bepaalt. Elk van de geometrische types (abstract of niet) heeft een aantal basismethodes. Deze zijn terug te vinden in Bijlage A.1.

2.2.2 Representatie

Om instanties van deze geometrische types voor te stellen, heeft de Simple Features Specification twee representaties: de *Well-Known Textual* (WKT) en de *Well-Known Binary* (WKB) representatie. De WKT representatie kan geometrische objecten voorstellen in ASCII vorm, terwijl de WKB representatie binaire streams in de vorm van BLOB waarden gebruikt hiervoor. Een punt op coördinaat (1,1) heeft als WKT representatie:

```
POINT(1 1)
```

en als WKB representatie:

```
0101000000000000000000F03F000000000000F03F
```

2.2.3 Metadata

Naast de standaard GIS object types en de functies om de objecten te manipuleren, zijn er ook 2 metadata tabellen of views beschreven in de Simple Features Specification:
`SPATIAL_REF_SYS` en `GEOMETRY_COLUMNS`.

`SPATIAL_REF_SYS` bevat informatie over de coördinaatsystemen die gebruikt worden in de spatiale database. De kolommen in de tabel zijn de *Spatial Reference System Identifier* (SRID), de *Spatial Reference System Authority Name* (AUTH_NAME), *Authority Specific Spatial Reference System Identifier* (AUTH_SRID) en een Well-Known Text beschrijving van het *Spatial Reference System* (SRTEXT). De SRID vormt de unieke key van het *Spatial Reference System*. Als een nieuwe geometrie wordt toegevoegd aan de database, dan moet de SRID die hierbij hoort verwijzen naar een rij uit `SPATIAL_REF_SYS`.

`GEOMETRY_COLUMNS` bevat een rij voor elke geometrie kolom in de database. De SRID, het type geometrie, de coördinaat dimensie en een verwijzing naar de feature en de geometrie tabel worden opgeslagen hierin. Een feature is een object met geometrische attributen en elke feature is een rij in de feature tabel. De eigenlijke instanties van geometriën worden opgeslagen in de geometrie tabel.

2.3 Transacties en locking

In deze Sectie wordt een korte inleiding gegeven over transacties en locking. In Hoofdstuk 3 wordt dit dan besproken voor verschillende databasesystemen.
Een transactie is een groep van sequentiële operaties die de database ondervragen of aanpassen en die uitgevoerd moeten worden alsof het één enkele opdracht is. Een transactie zal dus nooit afgehandeld zijn zonder dat alle verschillende individuele operaties succesvol uitgevoerd zijn. Transacties zijn van groot belang binnen databasesystemen. Neem bijvoorbeeld een spatiale database met geografische informatie over percelen bouwgrond. Een gebruiker wil een stuk van een bepaald perceel wegnemen en toevoegen aan een ander perceel. Dit zou uitgevoerd kunnen worden met de volgende SQL statements.

1. Controleer of de twee percelen de verwachte vorm en grootte hebben
2. Pas het eerste perceel aan door het stuk weg te nemen (een verwijdering van vertices)
3. Pas het tweede perceel aan door het stuk toe te voegen (een toevoeging van vertices)

Zonder transacties zou de uitvoering hiervan heel omslachtig zijn. Er kan veel mislopen. Het proces kan na SQL statement 2 afhaken, met een voorgoed verdwenen perceel als gevolg. Een tweede proces kan tussen SQL statement 1 en 2 het eerste perceel aanpassen. Deze veranderingen zullen na SQL statement 2 verloren zijn. Om deze mogelijke problemen te vermijden, zouden er meerdere extra controles uitgevoerd moeten worden. Met transacties moet men zich hierover geen zorgen maken. Een transactie heeft vier standaard eigenschappen, omvat in het acroniem ACID:

- *Atomicity*: Alle operaties moeten succesvol uitgevoerd worden. Als er één operatie mislukt, dan wordt er een rollback uitgevoerd.
- *Consistency*: Een transactie zal de database altijd in een goede en correcte toestand achterlaten. Als een transactie in de helft mislukt, dan wordt er naar de vorige (correcte) toestand teruggegaan, in plaats van een (foute) half-afgewerkte database achter te laten.
- *Isolation*: Transacties werken onafhankelijk en transparant van elkaar.
- *Durability*: In het geval van een faling van het systeem, dan zal het effect van een afgewerkte transactie nog zichtbaar zijn.

2.4 Spatiale Indexen

In deze Sectie worden diverse datastructuren voor spatiale indexering besproken.

2.4.1 Inleiding

Wanneer er een query op een grote database wordt uitgevoerd, dan zou het heel tijdrovend en inefficiënt zijn als al de spatiale data bekeken en vergeleken zou worden. Net als bij de gewone relationele databasesystemen zijn er specifieke datastructuren, genaamd indexen, ontworpen die snel en efficiënt toegang verlenen tot de spatiale gegevens. Voor alfanumerieke gegevens kunnen B-trees gebruikt worden, maar voor spatiale data zijn deze ontoereikend, omdat een B-tree een totale orde verwacht op de data en dit is niet mogelijk bij spatiale, multi-dimensionale data. Een spatiale index kan helpen bij queries die betrekking hebben op bepaalde regio's in de spatiale ruimte. Als een query vraagt naar spatiale objecten in een bepaalde regio, dan kan een spatiale index de objecten uit deze regio selecteren en de rest wegfilteren, zodat de intensieve berekeningen niet op alle spatiale objecten uitgevoerd moeten worden.
Neem bijvoorbeeld twee databasetabellen, één met alle landen en een andere met alle provincies ter wereld. Een query die alle provincies in België opvraagt zou, zonder spatiale index, alle provincies ter wereld laten vergelijken met België. Met spatiale index zou er eerst een ruwe selectie uitgevoerd worden, die een aantal provincies gelegen in de buurt van België teruggeeft. Dankzij de index moeten er nu veel minder intensieve berekeningen (het vergelijken van de geometrie van elke provincie met de geometrie van België) uitgevoerd worden.

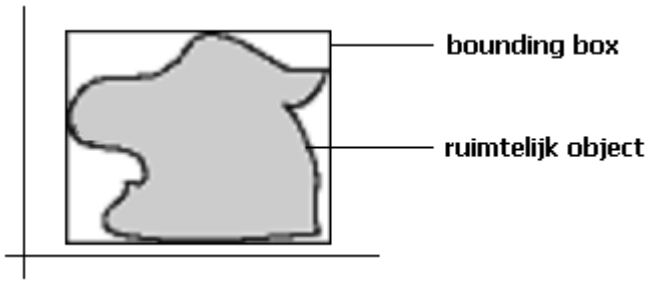
In de volgende subsecties zullen enkele spatiale indexen besproken worden.

2.4.2 R-tree

Inleiding

De R-tree, voorgesteld door Guttman in [10], is gebaseerd op de benadering van een spatiaal object door een minimale omvattende rechthoek of *bounding box* waarvan de randen parallel zijn met de assen van de data ruimte. Een voorbeeld van een object en de bounding box ervan is zichtbaar in Figuur 2.2. Het maakt niet uit of het spatiale object simpel (een punt) of complex (een polygoon) is, er kan namelijk altijd een bounding box geconstrueerd

worden. Een R-tree is hiërarchische boomstructuur van bounding boxes die mogelijk overlappen.



Figuur 2.2: Ruimtelijk object en bounding box

Structuur en eigenschappen

Een knoop in een R-tree bevat entries van de vorm $< key, pointer >$. Bij een niet-bladknoop verwijst *pointer* naar een kindknoop in de R-tree en *bbox* naar de minimale omvattende rechthoek of bounding box van alle rechthoeken die entries zijn in deze kindknoop. Bij een bladknoop verwijst *key* naar een record van een spatiaal object in de database en is *bbox* de omvattende rechthoek van dit object.

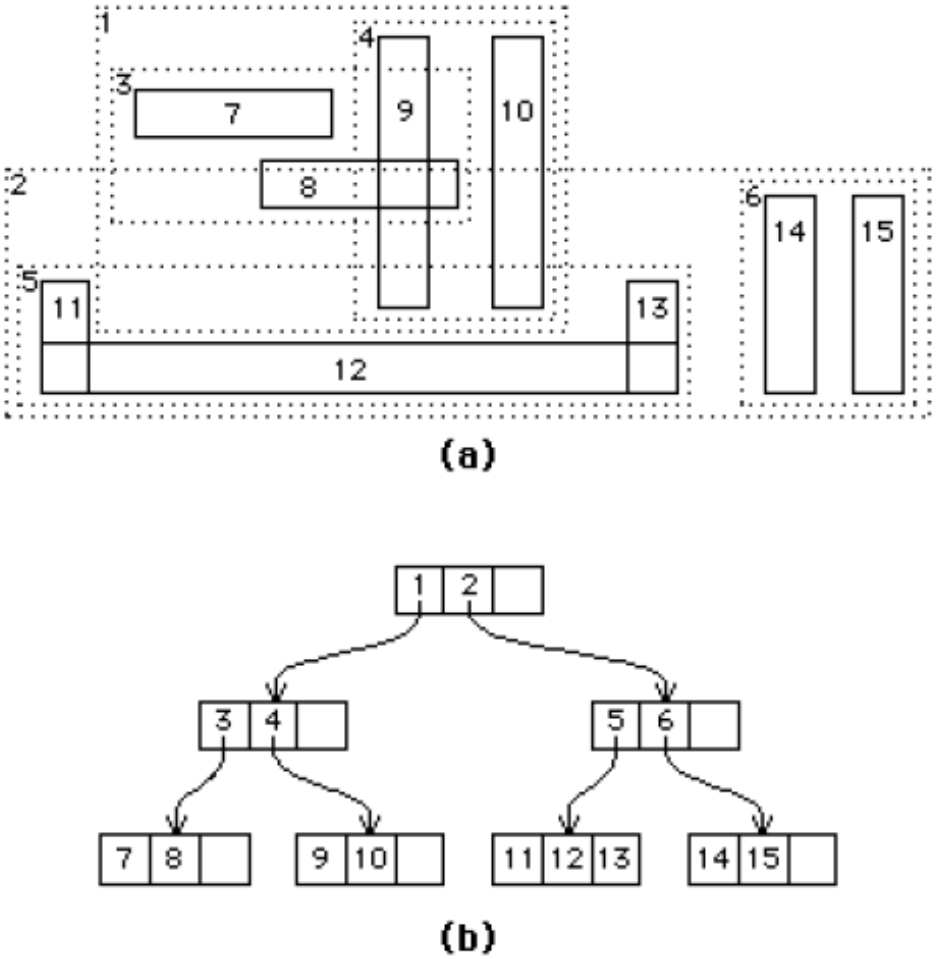
Laat M het maximum aantal entries zijn in 1 knoop en m de parameter die het minimum aantal entries specificeert in een knoop:

$$2 \leq m \leq M/2$$

Een R-tree heeft de volgende eigenschappen:

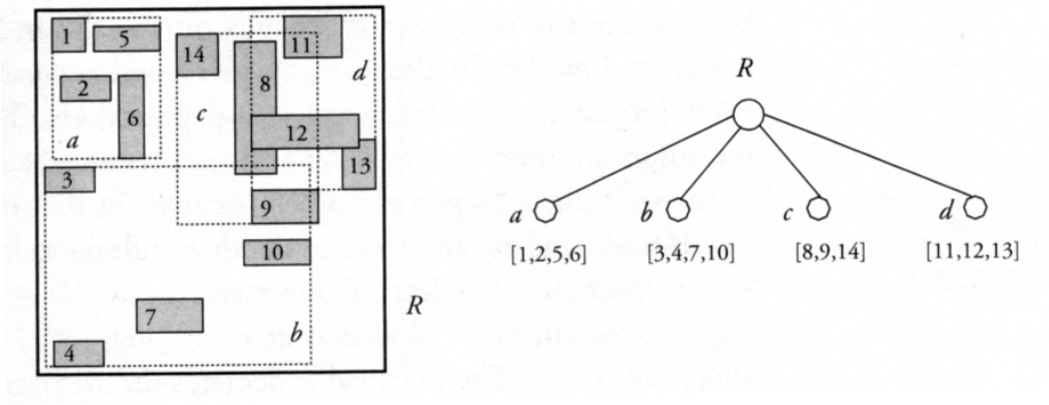
- De root heeft minstens 2 kinderen, tenzij het een bladknoop is.
- Elke niet-bladknoop heeft tussen m en M kinderen, tenzij het de root is.
- Elke bladknoop heeft tussen m en M entries, tenzij het de root is.
- Alle bladknopen bevinden zich op hetzelfde niveau.

Een voorbeeld van een R-tree wordt getoond in Figuur 2.3. In Figuur 2.3(a) is de ruimtelijke verdeling te zien en in Figuur 2.3(b) wordt de R-tree representatie ervan getoond. Hierbij verwijzen de getallen naar bounding boxes in Figuur 2.3(a).

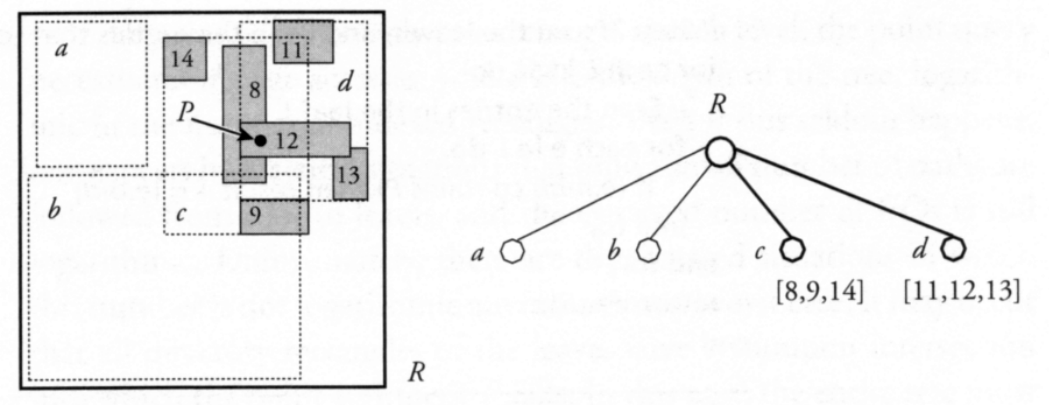


Figuur 2.3: Voorbeeld van een R-tree ([25])

De R-tree is volledig dynamisch in de zin dat alle optimalisaties aan de structuur gebeuren tijdens het toevoegen en verwijderen van entries. Er zijn geen periodische globale herstructureringen nodig. De R-tree maakt gebruik van een heuristische optimalisatie. Hierbij wordt de oppervlakte van de bounding boxes van de knopen in de boom geminimaliseerd. Het algoritme hiervoor wordt in de volgende subsectie (2.4.2) besproken. De structuur laat toe dat bounding boxes van verschillende knopen elkaar overlappen. Dit impliceert dat een query meerdere zoekpaden kan opleveren. Een voorbeeld van deze situatie wordt getoond in Figuur 2.4 en 2.5. Figuur 2.4 geeft de R-tree waarmee begonnen wordt en Figuur 2.5 toont een zoekopdracht met meerdere paden.



Figuur 2.4: Voorbeeld van een R-tree met M=4 en m=2 ([23])

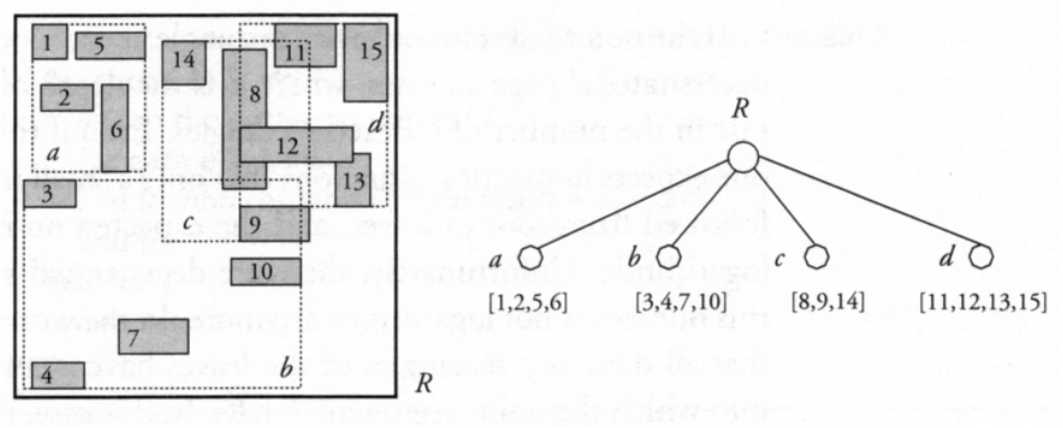


Figuur 2.5: Een zoekopdracht in een R-tree die meerdere paden teruggeeft ([23])

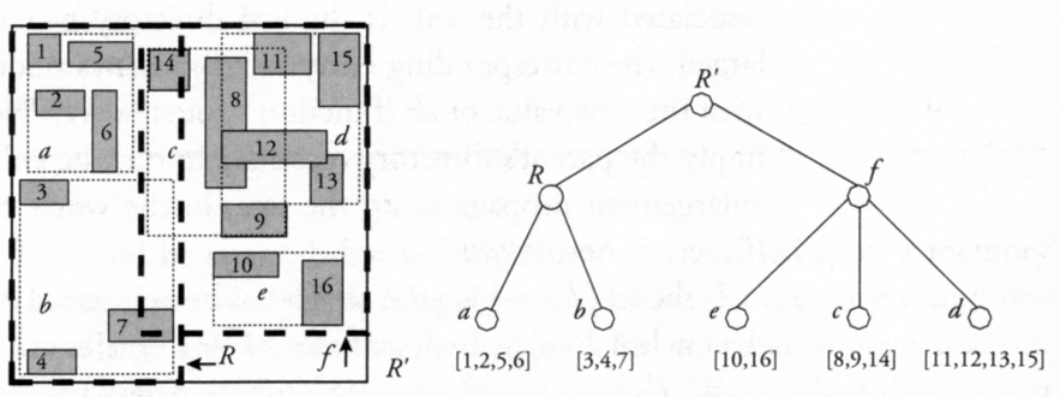
Algoritmen

Tijdens het toevoegen van een entry worden twee algoritmen aangeroepen die cruciaal zijn voor de performantie van de R-tree. Het eerste algoritme, **ChooseSubTree**, begint bij de root en zoekt recursief de beste subtree om een nieuwe entry aan toe te voegen. Het tweede algoritme, **Split**, wordt gebruikt wanneer een knoop te vol wordt. Dit wordt ook een overflow genoemd en is het geval wanneer ChooseSubTree eindigt in een knoop die het maximum aantal entries ($=M$) al heeft bereikt. Split zal de $M + 1$ entries dan zo goed mogelijk verdelen onder 2 knopen.

Een voorbeeld van het toevoegen van een object wordt getoond in Figuur 2.6 en 2.7, met Figuur 2.4 als startpunt. Voor al deze figuren geldt $M = 4$. In Figuur 2.6 wordt object 15 toegevoegd in bladknoop d (omdat ChooseSubTree deze knoop uitkiest). De bounding box van d moet vergroot worden omdat object 15 erbij komt en deze verandering wordt ook gepropageerd naar de root R. In Figuur 2.7 wordt, vertrekkende vanuit Figuur 2.6, object 16 in bladknoop b toegevoegd (omdat ChooseSubTree deze knoop uitkiest). Omdat b al vier entries bevat, veroorzaakt het toevoegen van het nieuwe object een overflow en moet er gesplit worden (met het Split-algoritme). Een nieuwe bladknoop e wordt gemaakt en de vijf entries (de vier entries van b en het nieuwe object) worden verdeeld onder b en e. Entry 10 wordt samen met entry 16 verplaatst naar e en entries 3, 4 en 7 blijven in b. De nieuwe entry e moet worden toegevoegd in de parent van b, namelijk de root. De root bevat al vier kinderen, dus moet er gesplit worden. Een nieuwe knoop f wordt gemaakt, met als kinderen e, c en d. a en b blijven in R. Tenslotte wordt er een nieuwe root R' gemaakt, met als kinderen de oude root R en de nieuwe knoop f.



Figuur 2.6: Toevoegen van object 15 zonder overflow als gevolg ([23])



Figuur 2.7: Toevoegen van object 16 met overflow als gevolg (([23])

Het ChooseSubTree algoritme van Guttman zoals beschreven in [10] gaat als volgt te werk:

1. Input is een nieuw spatiaal object E.
2. Stel N (de huidige knoop) in als de root.
3. Is N een bladknoop?
   - Indien antwoord 'ja', geef N terug.
   - Indien antwoord 'nee', kies de entry van N waarvan de oppervlakte van de bounding box het minste zou toenemen indien de bounding box van het nieuwe object E eraan wordt toegevoegd. Bij meerdere mogelijkheden wordt de entry gekozen waarvan de bounding box de kleinste oppervlakte heeft.
4. Stel N in op de kindknoop waar de pointer van de gekozen entry en herhaal stap 2.

Dit algoritme probeert om de oppervlakte van de omvattende rechthoeken zo klein mogelijk te houden. Guttman bespreekt drie verschillende split-algoritmen, die gekenmerkt worden door de computationele kost (ten opzichte van het aantal knopen): exponentieel, kwadratisch en lineair. Ze hebben alle drie het doel om de oppervlakte na het splitten te minimaliseren.

Exponentieel Split-algoritme

Het **exponentieel** of exhaustief algoritme probeert alle mogelijke gevallen en geeft de oplossing met het globale minimum terug. Dit geeft het beste

resultaat, maar heeft een grote CPU kost, waardoor het in de praktijk minder bruikbaar is.

Kwadratisch Split-algoritme

Het **kwadratische** algoritme probeert de beste oplossing te benaderen. Het begint door twee van de $M+1$ entries te zoeken die, moesten ze gegroepeerd worden, het meeste oppervlakte zouden verspillen. Deze twee entries vormen de twee begingroepen. De resterende entries worden één voor één toegevoegd aan een groep waarvan de oppervlakte het minst vergroot zou worden indien het werd toegevoegd.

De implementatie hiervan volgens Guttman bevat drie functies. De eerste functie is **QuadraticSplit** en is de hoofdfunctie die de $M+1$ entries over twee groepen verdeelt. Deze verloopt als volgt:

1. Roep PickSeeds aan om de twee entries te vinden die het begin van de twee groepen zullen vormen.
2. Roep PickNext aan om de volgende toe te wijzen entry te kiezen. Voeg deze toe aan de groep waarvan de bounding box het minst zou vergroten na het toevoegen van de entry.
    - Als er meerdere mogelijkheden zijn, voeg de entry dan toe aan de groep met de kleine oppervlakte.
    - Als er nog meerdere mogelijkheden zijn, voeg de entry dan toe aan de groep met het minst aantal entries.
    - Als er nog meerdere mogelijkheden zijn, kies dan een willekeurige groep.
3. Herhaal stap 2 totdat alle entries verdeeld zijn of één van de twee groepen te veel entries ($=M-m+1$) bevat. Als er entries overblijven, voeg deze dan toe aan de andere groep, zodat deze er minimum m heeft.

De hulpfunctie **PickSeeds** construeert de twee startgroepen en werkt als volgt:

1. Voor elk paar entries (E_1, E_2), construeer een minimale bounding box R die $E_1.bbox$ en $E_2.bbox$ bevat.
2. Bereken d met
   $d = oppervlakte(R) - oppervlakte(E_1.bbox) - oppervlakte(E_2.bbox)$

3. Kies het paar met de grootste d.

De vierde en laatste functie, **PickNext**, kiest de volgende entry om in een groep toe te voegen en verloopt als volgt:

1. Voor elke entry E die nog niet in een groep zit, bereken d_1 en d_2 door de groei in oppervlakte indien E aan respectievelijk groep 1 of 2 zou toegevoegd worden.

2. Kies de entry met het grootste verschil tussen d_1 en d_2.

Lineair Split-algoritme

Het **lineaire** algoritme om te splitten is bijna identiek aan het kwadratische algoritme, behalve dat het een andere versie van PickNext en PickSeeds gebruikt. Het lineaire algoritme kiest voor elke dimensie de twee entries met de grootste afstand ertussen en plaatst deze in twee verschillende groepen. De resterende entries worden willekeurig verdeeld.
PickSeeds bij het lineaire algoritme ziet er als volgt uit:

1. Voor elke dimensie, zoek de bounding box met de hoogste lage kant en die met de laagste hoge kant. Hou het afstandsverschil bij.

2. Voor elke dimensie, normaliseer nu de gehele set door het gevonden verschil door de volledige lengte te delen. Met volledige lengte wordt de afstand tussen de hoogste hoge kant en de laagste lage kant van alle bounding boxes voor de betreffende dimensie bedoeld.

3. Kies nu het paar met het grootste genormaliseerde verschil langs elke dimensie.

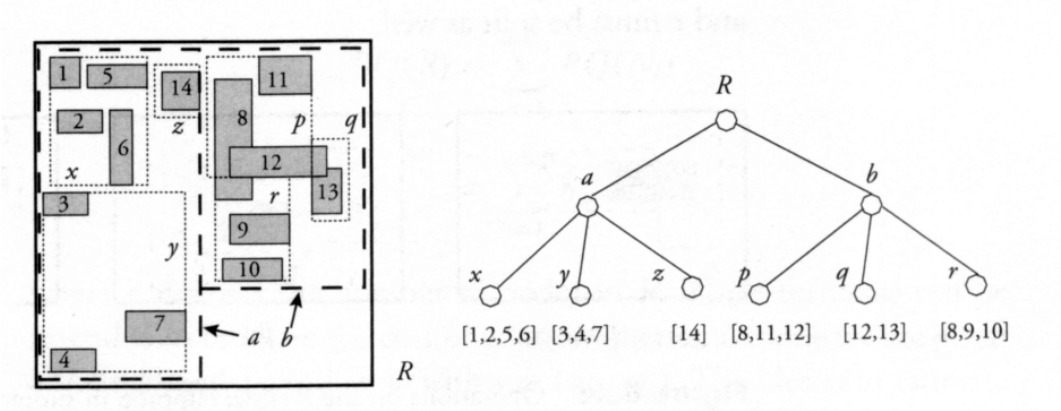
Dit algoritme geeft een zwakke performantie bij het zoeken, maar door het snelle splitten wordt dit gedeeltelijk of volledig goedgemaakt. Volgens de experimenten van Guttman zijn de kwadratische en de lineaire variant ongeveer gelijkwaardig, maar latere experimenten door Beckmann ([2]), waarbij meerdere parameters gevarieerd werden, toonden dat de kwadratische variant een betere performantie geeft.

2.4.3 R+tree

In '87 stelt Sellis in [26] de R+tree voor. De R+tree kan gezien worden als een mengeling tussen de R-tree en de K-D-B tree[2]. Het overlappen van knopen

[2]De K-D-B tree, voorgesteld door Robinson in [24], is een datastructuur die multidimensionale ruimtes splitst als een aanpasbare k-d tree, maar die de resulterende tree balanceert als een B-tree.

wordt vermeden door het toevoegen van een object in meerdere bladknopen. Het is ook niet gegarandeerd dat knopen minstens halfvol zijn. Een voorbeeld van een R+tree is te zien in Figuur 2.8.



Figuur 2.8: Voorbeeld van een R+tree ([23])

Alhoewel R+trees (en R-trees) werken op alle soorten geometrische objecten (in twee dimensies), halen R+trees een grote snelheidswinst tegenover de R-tree bij puntqueries (tot 50%). Dit komt omdat elke ruimte bij puntqueries bedekt wordt door maximaal één knoop. Een ander voordeel is dat er altijd maar één zoekpad doorheen de R+tree is en er dus minder knopen doorlopen moeten worden.

De R+tree heeft twee nadelen: meer opslagruimte en een complexere implementatie. Een R+tree neemt meer geheugenruimte in, omdat de bounding boxes vaak verdeeld worden onder meerdere knopen, bij het vermijden van overlappingen. Het opbouwen en het dynamische onderhoud zijn veel complexer dan bij de R-tree. Omdat dieper ingaan op de R+tree te ver zou leiden, wordt er voor meer informatie verwezen naar [26].

2.4.4 Greene's variant

In '89 stelt Greene in [9] een alternatief split-algoritme voor. Het zoekt een split-as die de entries in twee zo gelijk mogelijke groepen verdeelt. Dit algoritme werkt goed, behalve wanneer het algoritme faalt om de juiste as te vinden. Dit resulteert dan in een 'slechte split'. Het nadeel van Greene's variant is dus dat het de keuze van de split-as als enige geometrische criterium gebruikt om te splitten.

ChooseSubtree van Guttman blijft gehouden en PickSeeds wordt gebruikt. **Greene's Split** ziet er als volgt uit:

1. Roep ChooseAxis om de split-as te bepalen.
2. Roep Distribute aan.

De hulpfunctie **ChooseAxis** ziet er als volgt uit:

1. Roep PickSeeds om uit de huidige knoop de twee entries die het verst van elkaar liggen te vinden.
2. Voor elke as, bereken het afstandsverschil van de twee gekozen seeds.
3. Voor elke as, normaliseer dit verschil nu door het te delen door de lengte van de bounding box van de huidige knoop langsheen de as.
4. Geef de as met het grootste genormaliseerde verschil terug.

De hulpfunctie **Distribute** gaat als volgt:

1. Sorteer de entries volgens de lage waarde van hun bounding box langs de gekozen as.
2. Plaats de eerste $(M + 1)div2$ entries in de eerste groep en de laatste $(M + 1)div2$ entries in de tweede groep.
3. Als $M + 1$ oneven is, plaats de laatste overgebleven entry dan in de groep die het minst in grootte zou toenemen door de toevoeging.

2.4.5 R*tree

Inleiding

De originele R-tree van Guttman probeert om de oppervlakte van de bounding boxes te minimaliseren. De variant van Greene gebruikt een split-as om de entries te verdelen. In '90 stelt Beckmann in [2] dat er meerdere geometrische criteria geöptimaliseerd moeten worden om tot een betere structuur te komen. Dit onderzoek leidt dan tot de R*tree.

Optimalisatie criteria

Beckmann gebruikt de volgende criteria, die bepalend zijn voor een goede performantie:

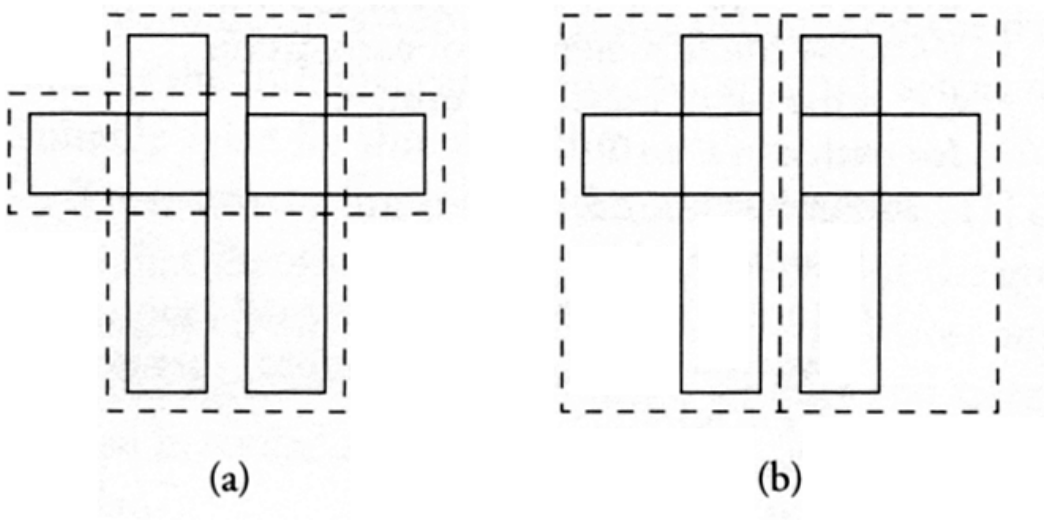
1. *De oppervlakte van de bounding box minimaliseren.* Hierdoor verschuiven beslissingen over 'te doorzoeken paden' naar een hoger niveau.

2. *De overlap van de bounding box minimaliseren.* Hierdoor moeten er minder paden doorlopen worden.

3. *De marge van de bounding box minimaliseren.* De marge is hier de som van de lengtes van de randen van de bounding box. Bij een vaste oppervlakte, is het object met de kleinste marge het vierkant. Dus door de marge in plaats van de oppervlakte te optimaliseren, worden de bounding box meer vierkant.

4. *Opslaggebruik optimaliseren.* Een verbeterde opslaggebruik vermindert de query kost, omdat de hoogte van de boom lager wordt.

Wanneer oppervlakte of overlap van bounding boxes klein wordt gehouden, dan is er meer vrijheid in het aantal entries per knoop nodig en is het opslaggebruik lager. Er is ook meer vrijheid in de vorm nodig, wat zorgt voor bounding boxes die minder vierkant zijn. Het optimaliseren van criteria 1 en 2 heeft dus een negatief effect op criteria 3 en 4. Wanneer de oppervlakte geminimaliseerd wordt, dan vermindert mogelijk ook de overlap. Dit is niet altijd het geval. Zoals getoond wordt in Figuur 2.9, kan het voorkomen dat een split met minimale oppervlakte (Figuur 2.9a) er heel anders uitziet dan een split met minimale overlap (Figuur 2.9b).



Figuur 2.9: Het mogelijke verschil tussen een split met minimale oppervlakte (a) en een split met minimale overlap (b) ([23])

Wanneer de marges geminimaliseerd worden, dan verbetert het opslaggebruik ook. Het is duidelijk dat deze vier criteria sterk afhankelijk zijn van elkaar. Beckmann heeft in zijn onderzoek verschillende combinaties geprobeerd en is tot de volgende ChooseSubtree -en Split-algoritmen gekomen.

ChooseSubtree

Het **ChooseSubtree** algoritme ziet er als volgt uit:

1. Input is een nieuw spatiaal object E.
2. Stel N (de huidige knoop) in als de root.
3. Is N een bladknoop?
   - Indien antwoord 'ja', geef N terug.
   - Indien antwoord 'nee', wijzen de kinderen in N naar bladknopen?
     - Indien antwoord 'ja', kies de entry van N waarvan de overlap van de bounding box het minste zou toenemen indien de bounding box van het nieuwe object E wordt toegevoegd. Bij meerdere gevallen wordt de entry gekozen waarvan de oppervlakte van de bounding box het minste zou toenemen indien de nieuwe data wordt toegevoegd. Zijn er dan nog meerdere

gevallen, dan wordt de entry gekozen waarvan de bounding box de kleinste oppervlakte heeft.
- Indien antwoord 'nee', kies de entry van N waarvan de oppervlakte van de bounding box het minste zou toenemen indien de bounding box van het nieuwe object E wordt toegevoegd. Bij meerdere gevallen wordt de entry gekozen waarvan de bounding box de kleinste oppervlakte heeft.

4. Stel N in op de kindknoop waar de pointer van de gekozen entry en herhaal stap 2.

Wanneer de bladknoop gekozen moet worden met ChooseSubtree, dan geeft het minimaliseren van de overlap een betere performantie. Bij niet-bladknopen geven alternatieve methoden geen voordeel tegenover Guttman's originele algoritme (dat de oppervlakte minimaliseert). Deze twee ideeën zijn verwerkt in het ChooseSubtree algoritme dat zonet beschreven is.

De complexiteit van het berekenen van de overlap is kwadratisch met het aantal entries. Omdat dit voor grote knopen een nadeel is, is er een aanpassing aan het algoritme aangebracht die maar een aantal entries beschouwt en zo de hoge CPU kost drukt:

1. Sorteer alle bounding boxes van entries in de huidige knoop N volgens toenemende oppervlaktevergroting, indien de nieuwe datarechthoek aan die bladknoop toegevoegd zou worden.

2. Laat A de groep van de eerste p entries zijn.

3. Kies uit A de entry waarvan de overlap van de bounding box het minste zou toenemen indien de bounding box van de nieuwe datarechthoek wordt toegevoegd. Bij meerdere gevallen wordt de entry gekozen waarvan de oppervlakte van de bounding box het minste zou toenemen indien de bounding box van de nieuwe data wordt toegevoegd. Zijn er dan nog meerdere gevallen, dan wordt de entry gekozen waarvan de bounding box de kleinste oppervlakte heeft.

Proefondervindelijk is Beckmann ([2]) tot de conclusie gekomen dat met $p = 32$ en in twee dimensies, er bijna geen verschil te merken is in de performantie, vergeleken met het gewone algoritme. De CPU kost blijft hoger dan die van de originele ChooseSubtree van Guttman, maar dit wordt goedgemaakt door een verbeterde werking van de R-tree, met minder schijftoegangen. Deze nieuwe versie blijkt beter dan Guttman's versie te werken bij „queries met een kleine query rechthoek op databestanden met niet-uniform verdeelde kleine rechthoeken of punten". Voor andere queries is de performantie vergelijkbaar.

Split

Voor het splitten worden eerst alle entries gesorteerd langs elke as. Eerst volgens de lage en dan volgens de hoge waarde van hun bounding box. Voor elke sortering (één per dimensie-as), worden er $M - 2m + 2$ distributies van de $M + 1$ entries in twee groepen bepaald. Bij de k^{de} verdeling, bevat de eerste groep de eerste $(m - 1) + k$ entries en de tweede groep de resterende entries. Om de kwaliteit van een distributie te beoordelen, worden er drie waarden gebruikt (hierbij staat bb voor bounding box):

- **oppervlakte-waarde** = $opp[bb(groep1)] + opp[bb(groep2)]$
- **marge-waarde** = $marge[bb(groep1)] + marge[bb(groep2)]$
- **overlap-waarde** = $opp[bb(groep1) \cap bb(groep2)]$

Onderzoek van Beckmann ([2]) naar het beste gebruik van deze waarden heeft tot het volgende split-algoritme geleid.

Het algoritme **Split**:

1. Roep ChooseSplitAxis aan om de juiste split-as te kiezen.
2. Roep ChooseSplitIndex aan om de beste distributie in twee groepen langs deze as te kiezen.
3. Verdeel de entries in de twee groepen.

Het algoritme **ChooseSplitAxis**:

1. Voor elke as, sorteer de entries, eerst volgens de lage en dan volgens de hoge waarde van hun bounding box. Bepaal alle distributies zoals eerder beschreven. Bereken S, de som van alle marge-waarden van de verschillende distributies.
2. Kies de as met de kleinste S.

Het tweemaal sorteren in ChooseSplitAxis voor elke as heeft een tijdscomplexiteit van $O(Mlog(M))$. Uit experimenten blijkt dat dit ongeveer de helft van de CPU kost van het splitten is. De andere helft wordt besteed aan het berekenen van de marge-waarde van $2 * (2 * (M - 2m + 2))$ bounding boxes en de overlap-waarde van $2 * (M - 2m + 2)$ distributies, voor elke as. De marges worden berekend in ChooseSplitAxis en de overlap-waarden in ChooseSplitIndex.

Het algoritme **ChooseSplitIndex**:

1. Kies de distributie langs de gekozen as met de kleinste overlap-waarde. Als er meerdere gevallen zijn, kies dan de distributie met de kleinste oppervlakte-waarde.

Forced Reinsert

Zowel de R-tree als de R*tree zijn niet-deterministisch van aard wat het toevoegen van entries in knopen betreft. Wanneer entries in een verschillende volgorde worden toegevoegd, dan geeft dit mogelijk een heel ander resultaat. Een entry die wordt toegevoegd, kan de bounding boxes in de R-tree op die manier beïnvloeden dat de performantie er onder lijdt en blijft lijden. Tijdens het splitten wordt er een kleine lokale herorganisatie uitgevoerd, maar deze kan onmogelijk alle 'slechte bounding boxes' verbeteren. Daarom voert Beckmann in [2] een experiment uit waarbij hij een lineaire R-tree gebruikt en deze vult met 20000 uniform verdeelde rechthoeken. Daarna verwijderde hij de eerste 10000 en voegde deze opnieuw toe. Het resultaat was een snelheidswinst van 20% tot 50%, afhankelijk van het soort query.

Om die reden verplicht de R*tree het opnieuw toevoegen tijdens het toevoegen van entries. Deze vorm van dynamische herorganisatie wordt ook „Forced Reinsert" genoemd. De algoritmen voor het toevoegen van entries worden nu gegeven.

Het algoritme **InsertData**:

1. Roep Insert aan met het boomniveau l van de bladknopen en de nieuw toe te voegen entry E als parameters.

Het algoritme **Insert**:

1. Roep ChooseSubtree aan, met het boomniveau l als parameter, om de juiste knoop N te vinden waar de nieuwe entry E geplaatst moet worden.

2. Heeft N minder dan M entries, voeg E dan toe.
   Heeft N M entries, roep dan OverflowTreatment aan met het niveau van N als parameter.

3. Als OverflowTreatment werd aangeroepen met een split als resultaat, propageer dan OverflowTreatment naarboven in de boom.
   Als OverflowTreatment een split van de root veroorzaakte, maak dan een nieuwe root.

4. Pas alle rechthoeken in het pad van de toegevoegde knoop aan, zodat deze de bounding box van hun kinderen zijn.

Het algoritme **OverflowTreatment**:

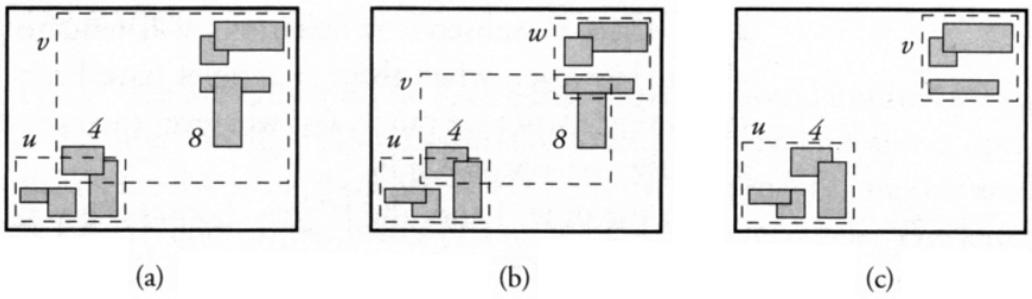
1. Is het niveau niet de root en is dit de eerste aanroep van OverflowTreatment op een gegeven niveau gedurende het toevoegen van een nieuw data object?

   - Indien antwoord 'ja', roep dan ReInsert aan.
   - Indien antwoord 'nee', roep dan Split aan.

Het algoritme **ReInsert**:

1. Voor alle $M+1$ entries van een knoop N, bereken de afstand d van het midden van hun rechthoek tot het midden van de bounding box van N.
2. Sorteer de entries dalend volgens d.
3. Verwijder de eerste p entries van N en pas de bounding box van N aan.
4. Voeg nu deze entries één voor één weer toe met Insert, beginnende met de entries met de grootste d.

Een voorbeeld van een „Forced reinsert" is te zien in Figuur 2.10. In 2.10a wordt rechthoek 8 toegevoegd, waardoor knoop v te vol geraakt en een overflow gebeurt. Het traditionele R-tree split algoritme (zie Figuur 2.10b) zal enkel een lokale herorganisatie uitvoeren. De R*tree (zie Figuur 2.10c) zal de rechthoeken in v (rechthoek 4 en 8) nu opnieuw toevoegen, beginnende met de rechthoek die het verst van het midden verwijderd is. Rechthoek 4 zal hierdoor verplaatst worden naar knoop u. Hierna zal rechthoek 8 aan v toegevoegd kunnen worden, zonder dat er een split veroorzaakt wordt.



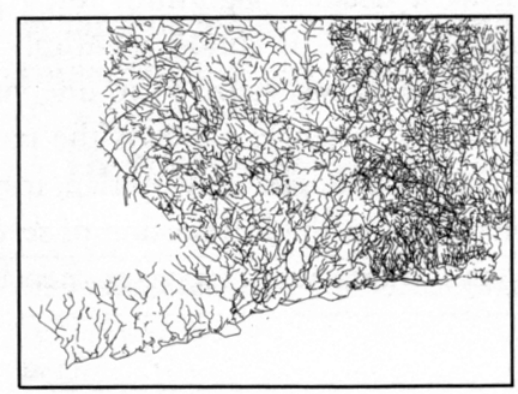
Figuur 2.10: Voorbeeld van een „Forced reinsert" ([23])

Wanneer een nieuw data object wordt toegevoegd, dan zal de eerste 'overflow behandeling' altijd resulteren in het opnieuw toevoegen van p entries. Wanneer al deze entries dan terug worden toegevoegd aan dezelfde knoop, zal er gesplit worden. Een „Forced reinsert" verplaatst entries tussen naburige knopen en vermindert zo de overlap. Hierdoor verbetert het opslaggebruik. Ook zullen er minder splits gebeuren door de herstructureringen. Omdat de entries die verder van het midden van de bounding box liggen opnieuw worden toegevoegd, wordt de vorm van de bounding box meer vierkant. Dit heeft een positief effect op het opslaggebruik, zoals eerder vermeld.

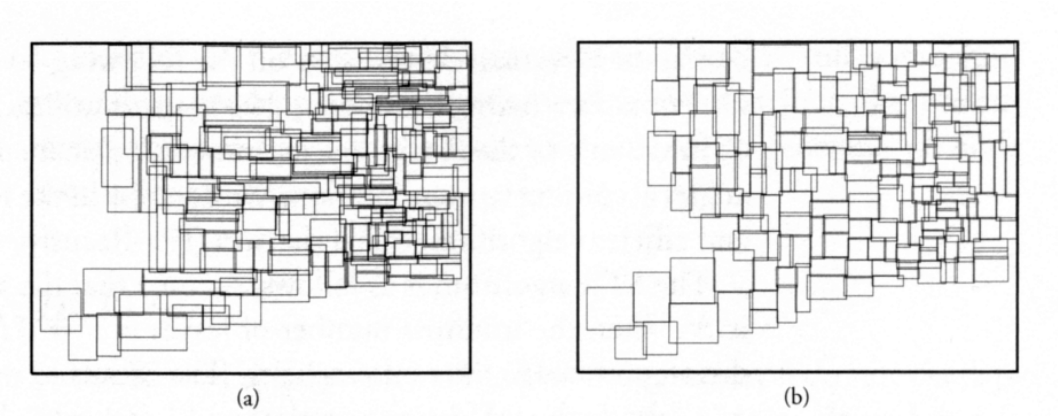
Uiteraard is de CPU kost nu een stuk hoger, omdat het Insert algoritme veel vaker wordt aangeroepen. Dit wordt goedgemaakt doordat er minder splits uitgevoerd moeten worden.

Conclusie

De R*tree is een robuuste uitbreiding van de R-tree. De datastructuren zijn identiek, maar de performantie van de R*tree is beter dan die van de originele R-tree van Guttman, de variant van Greene en de R+tree van Sellis. Door het gebruik van „Forced Reinserts" kunnen splits vermeden worden, de structuur wordt dynamisch geherstructureerd en het opslaggebruik wordt verbeterd. Hierdoor is de gemiddelde kost om nieuwe data kost toe te voegen lager dan die van de andere R-tree structuren. Het enige nadeel is de iets complexere implementatie. Een praktijkvoorbeeld van hoe een R-tree en een R*tree verschillen is zichtbaar in Figuur 2.11 en 2.12. Figuur 2.11 toont de originele dataset, een visualisatie van het rivierennetwerk in Connecticut. Figuur 2.12a toont de visualisatie van de R-tree en Figuur 2.12b toont de visualisatie van de R*tree hiervan, op bladknoopniveau. Het is duidelijk te zien dat de R*tree beter gestructureerd is door de dynamische herstructureringen, met minder overlap en een kleinere marge als gevolg.



Figuur 2.11: Dataset die het rivierennetwerk in Connecticut voorstelt ([23])



Figuur 2.12: R-tree (a) en R*tree (b) van de dataset in 2.11 op bladknoopniveau ([23])

2.4.6 GiST

Inleiding

GiST staat voor *Generalized Search Tree* en wordt door Hellerstein, Naughton en Pfeffer voorgesteld in [11]. Het is een gebalanceerde, tree-gebaseerde indexeringsmethode die als een template werkt. De GiST kan gebruikt worden om indexeringsmethodes te implementeren, zoals B+trees, R-trees, R+trees en R*trees.

B+trees ondersteunen enkel (1-dimensionale) bereik-predicaten, bijvoorbeeld

„Geef alle films uitgebracht na 2000". R-trees ondersteunen enkel multidimensionale bereik-predicaten, bijvoorbeeld „Geef alle Limburgse gemeentes kleiner dan Diepenbeek". Als het nu nodig zou zijn om domein-specifieke queries te gebruiken, zoals „Geef alle films die coole explosies bevatten" of „Geef alle Limburgse gemeentes met een ziekenhuis", dan schieten structuren als de B+tree of R-tree tekort, maar niet de GiST. De GiST ondersteunt elk mogelijk predicaat. Er moeten hiervoor vier user-defined methods geïmplementeerd worden, die het gedrag van de keys in de tree beschrijven. Deze methoden kunnen ingewikkeld zijn als de gevraagde queries ingewikkeld zijn, maar voor de standaard queries (zoals die van B+trees en R-trees) zijn ze triviaal.

Een voordeel van de GiST is dus dat het makkelijk uitbreidbaar is (zowel op gebied van data als queries) en dat het de ontwikkeling van indexeringsmogelijkheden van eigen datatypes toestaat. Deze kunnen dan gemaakt worden door een expert in het datatype domein, en niet noodzakelijk door een database expert zoals bij de andere methoden nodig was. De GiST ondersteunt ook het hergebruik van code en biedt een mooie, propere interface.

Structuur en eigenschappen

De GiST is gebalanceerd met een variabel aantal kinderen per knoop: tussen kM en M, waarbij k de vulfactor is.

$$\frac{2}{M} \leq k \leq \frac{1}{2}$$

De root van de boom is een uitzondering op deze regel: deze kan 2 tot M kinderen bevatten.

De knopen bevatten $< key, pointer >$ paren (of entries), zoals bij de andere besproken structuren. De pointers wijzen naar een kindknoop bij een niet-bladknoop en naar een record in de database bij een bladknoop. Alle bladknopen komen voor op hetzelfde niveau in de boom. De keys omvatten, in tegenstelling tot de andere structuren (B+trees, R-trees, ...), een user-defined class. Ze vertegenwoordigen een eigenschap die waar is voor alle kinderen die bereikbaar zijn via de pointer. Als men een B+tree zou implementeren met GiST, dan zouden de keys bereiken van getallen zijn. Bijvoorbeeld, „alle kinderen bereikbaar via deze pointer liggen tussen 10 en 20". Bij een R-tree implementatie zouden de keys bounding boxes zijn. Bijvoorbeeld, „alle kinderen bereikbaar via deze pointer liggen in Diepenbeek". Om een eigen GiST te maken, moet er enkel bepaald worden wat er in de

keys voorgesteld moet worden en er moeten vier methoden geschreven worden.

Deze vier methoden zijn:

1. **Consistent**: Deze methode zorgt ervoor dat de tree correct doorlopen kan worden. Gegeven een key p en een query q, dan geeft deze methode FALSE terug als het zeker is dat p en q niet TRUE kunnen zijn voor een gegeven entry. Als dit niet gegarandeerd kan worden, dan worden de kinderen van de entry doorzocht.

2. **Union**: Gegeven een set S van entries, dan geeft deze methode een nieuwe key p terug die TRUE is voor alle keys onder S. Een simpele, maar inefficiënte manier om dit implementeren is de key teruggeven die de disjunctie is van alle keys in S: '$p_1 \vee p_2 \vee p_3 \vee ...$'.

3. **Penalty**: Gegeven de keuze van het toevoegen van een nieuwe entry E_2 in een subtree van $E_1 =< p, ptr >$, dan geeft deze methode een getal terug dat weergeeft hoe goed of slecht het zou zijn wanneer de toevoeging zou gebeuren. Het zegt dus iets over de groei van de subtree. Deze methode wordt gebruikt door de split en insert algoritmen. Typisch wordt hiervoor de groei van de grootte van $E_1.p$ naar $Union(E_1, E_2)$ gebruikt.

4. **PickSplit**: Wanneer er elementen toegevoegd worden in een GiST, dan moeten er af en toe knopen gesplitst worden. Deze methode beslist welke entries dan naar een nieuwe knoop gaan en welke er bij de oude blijven.

Er zijn ook nog twee optionele methoden die de performantie kunnen verbeteren:

1. **Compress**: Gegeven een entry $< p, ptr >$, dan geeft deze methode een nieuwe entry $< \pi, ptr >$ terug. Hierbij is π een gecomprimeerde versie van p. De implementatie van deze compressie is afhankelijk van het domein.

2. **Decompress**: Gegeven een gecomprimeerde entry $< \pi, ptr >$ waarbij $\pi = Compress(p)$, dan geeft deze methode een entry $< r, ptr >$ terug, zodat $p \rightarrow r$. Dit is mogelijk een 'lossy' compressie, omdat we niet eisen dat $p \leftrightarrow r$. Deze kan gemaakt worden afhankelijk van het domein waarin gewerkt wordt.

Indien performantiewinst niet nodig is, dan kan de identiteits-functie gebruikt worden als Compress en Decompress. De manier waarop een GiST gebruik

maakt van deze methoden bij het zoeken, toevoegen en verwijderen wordt nu besproken.

Werking: Zoeken

Er zijn twee manieren waarop gezocht kan worden. De eerste manier is een algemene methode die altijd gebruikt kan worden. Gegeven een GiST met als root R en als query q worden alle records in de databases die voldoen aan q teruggegeven. Voor elke key in de knoop wordt gecontroleerd of deze key p consistent is met q (gebruik makende van **Consistent**). Als dit zo is, dan worden de kinderen van de knoop recursief op dezelfde manier gecontroleerd. Wanneer een knoop gecontroleerd is, dan wordt de unie van het resultaat van alle kinderen teruggegeven (gebruik makende van **Union**). Als p niet consistent is met q, dan wordt de lege set teruggegeven. Bij bladknopen wordt het singleton teruggegeven als de key consistent is en de lege set als dat niet zo is.

De tweede manier is enkel bruikbaar op domeinen met een lineaire orde. Om deze manier te kunnen gebruiken, moet er aan 4 voorwaarden voldaan worden bij het creëren van de tree:

- De boolean **IsOrdered** moet op TRUE staan. Deze tree-variabele is statistisch en wordt ingesteld bij de creatie. Standaard staat deze variabele ingesteld op FALSE.
- Een extra methode **Compare**: Gegeven twee entries met keys p_1 en p_2, dan zegt deze functie of p_1 voor of achter p_2 komt of dat ze equivalent zijn. Deze methode wordt gebruikt om entries op een gesorteerde manier toe te voegen in een knoop.
- De **PickSplit** methode moet ervoor zorgen dat voor elke entry E_1 op P_1 en E_2 op P_2, waarbij E_1 zich links van E_2 bevindt in de boom, **Compare**(E_1,E_2) teruggeeft dat „E_1 voor E_2 komt".
- De methoden moeten ervoor zorgen dat geen twee keys in een knoop overlappen. Met andere woorden, voor elk paar entries E_1, E_2 in een knoop moet **Consistent**($E_1.p$,$E_2.p$) evalueren tot FALSE.

Als deze vier voorwaarden voldaan zijn, dan kunnen queries die controleren op gelijkheid of kijken of iets binnen een bereik ligt uitgevoerd worden. Dit gebeurt door de methode **FindMin** uit te voeren en daarna herhaaldelijk **Next** op te roepen.
FindMin en **Next** gaan als volgt te werk:
Gegeven een GiST met als root R en een query q, dan zal **FindMin** de tree

afdalen, daarbij zo links mogelijk blijven en enkel de knopen bekijken waarvan de keys Consistent zijn met q. Wanneer een bladknoop bereikt wordt, dan wordt de eerste entry van deze knoop teruggegeven waarvan de key Consistent is met q.

Gegeven een entry E die aan de query q voldoet (dit is het resultaat van FindMin), dan geeft de methode **Next** de volgende entry terug die voldoet aan q, of $NULL$ als deze niet bestaat.

Deze manier is efficiënter dan de eerste manier, omdat de bladknopen langs één pad vanaf de root bezocht worden (dankzij de totale orde). Deze techniek is vergelijkbaar met het opzoeken van bereiken in B+Trees. Bij spatiale data kan deze techniek echter niet gebruikt worden, omdat er geen totale orde is op spatiale data.

Werking: Toevoegen

De manier waarop toegevoegd wordt in een GiST is belangrijk omdat de GiST altijd gebalanceerd moet blijven. De manier die nu besproken wordt lijkt veel op die van R-Trees. Het toevoegingsalgoritme laat toe om het niveau waar toegevoegd wordt te specifiëren. We zullen aannemen dat het niveau stijgt naarmate men omhoog gaat in de tree, waarbij bladknopen als niveau 0 hebben. Dus nieuwe entries worden toegevoegd op niveau $n = 0$.

De methode **Insert** werkt als volgt:

- Gegeven een GiST met als root R, een entry $E = (p, ptr)$ en een niveau l. p is de key van de tuples, waar ptr naar verwijst.
- Met behulp van de methode **ChooseSubTree** wordt gezocht in welke subtree van de GiST de entry E toegevoegd moet worden: $L = ChooseSubTree(R, E, l)$.
- Als er genoeg plaats is voor E in L, dan wordt E toegevoegd (op een geordende manier met **Compare**, indien **isOrdered** TRUE is).
- Als er niet genoeg plaats is, dan wordt de methode Split aangeroepen: $Split(R, L, E)$.
- Tenslotte wordt **AdjustKeys** aangeroepen, opdat de verandering van de keys ook voor de knopen bovenaan worden doorgevoerd: $AdjustKeys(R, L)$.

De functie **ChooseSubTree** geeft de knoop terug die zich op niveau l bevindt en waar de entry het best aan wordt toegevoegd. De methode loopt recursief door de tree waarbij het voor alle entries telkens de penalty met de functie **Penalty** (die door de gebruiker is geïmplementeerd) berekent en verdergaat met de knoop die de laagste penalty heeft. Als de methode op een knoop met niveau l komt, dan wordt deze teruggegeven. Vooral als **IsOrdered** TRUE is, dan is het belangrijk dat **Penalty** zeer nauwkeurig werkt, opdat **ChooseSubTree** goed zou werken.

Split wordt, net als bij de R-tree, gebruikt als een knoop te groot wordt en in twee verdeeld moet worden. Gegeven een GiST met als root R, een geselecteerde knoop N en een nieuwe entry $E = (p, ptr)$, dan geeft $Split(R, N, E)$ een GiST terug waarbij N in twee is verdeeld en E is toegevoegd. Het maakt gebruik van **PickSplit** (die door de gebruiker is geïmplementeerd) om de knoop in twee groepen te verdelen en E toe te voegen.

De methode **AdjustKeys** doorloopt de GiST R van onder naar boven, beginnende bij knoop L (die als parameter wordt meegegeven). Bij elke knoop worden de keys aangepast zodat deze de subtrees nog goed omschrijven. Hierbij wordt gebruik gemaakt van **Union** om een nieuwe key te maken. Er wordt gestopt als er geen aanpassing aangebracht moet worden of als de root bereikt wordt.

Werking: Verwijderen

Bij het verwijderen van een entry uit de GiST wordt deze entry eerst gezocht met **Search** en daarna verwijderd. Daarna worden de veranderingen naar boven toe ook gepropageerd, vergelijkbaar met de **AdjustKeys** van **Insert**. Het kan echter gebeuren dat een knoop niet meer vol genoeg is na een verwijdering. Bij een geordende GiST (als **IsOrdered** TRUE is), wordt er dan geleend van naburige knopen. Indien lenen niet kan, dan vloeit de knoop samen met één van zijn buren (zoals bij de B+tree). Als **IsOrdered** FALSE is, dan worden alle entries van de knoop verwijderd en toegevoegd aan andere naburige knopen (zoals bij de R-tree).

Performantie

Bij gebalanceerde bomen zoals de B+tree kan het aantal te verkennen knopen vastgelegd worden. Voor een puntquery in een tree zonder duplicaten

bijvoorbeeld is de bovengrens $O(\log n)$. Deze worst-case complexiteit kan echter niet verzekerd worden als de keys in de knopen kunnen overlappen, zoals bij de R-tree en de GiST het geval is. Dit is omdat er dan mogelijk meerdere paden doorlopen moeten worden. De worst-case complexiteit loopt op tot $O(2n)$ wanneer de volledige tree doorlopen moet worden. Daarom hangt de performantie van de GiST erg af van de mate waarin de keys van de knopen overlappen.

Er zijn twee hoofdoorzaken voor het overlappen van keys: data overlap en informatieverlies door compressie van keys. Key overlap ten gevolge van data overlap is logisch: als veel objecten overlappen, dan zullen ook hun keys overlappen. Als voorbeeld hiervoor wordt in [11] een bord spaghetti gebruikt, waarbij elk object een spaghetti sliert voorstelt. De slierten zullen elkaar nooit snijden in de drie dimensies, maar hun bounding boxes zullen dat wel doen. Een dataset waarin veel objecten vrijwel identiek zijn, zal ook een inefficiënte index opleveren. Key overlap ten gevolge van informatieverlies door compressie komt voor wanneer **Compress** en **Decompress** geen identieke keys teruggeven (en er dus *lossy compression*[3] wordt gebruikt.

GiST vs. R-tree

R-trees verdelen de data in rechthoeken, sub-rechthoeken, sub-sub-rechthoeken, ... Ze worden in databases gebruikt om spatiale data te indexeren. Met de GiST kan net hetzelfde gedaan worden, maar ook nog veel meer. De GiST is veel robuuster dan de R-tree. Het kan alle data indexeren volgens eender welk criterium, zolang de vier benodigde methoden maar op een juiste manier geïmplementeerd zijn.

De GiST heeft, naast de bredere waaier aan mogelijkheden door de uitbreidbaarheid, ook nog een belangrijk voordeel tegenover de R-tree: De GiST is efficiënter. Het gebruikt key compressie, waardoor een betere verdeling van het aantal kinderen per knoop bekomen wordt.

2.4.7 Conclusie

Het gebruik van een index bij spatiale data is eerder een „must" dan een luxe. B-trees kunnen niet gebruikt worden bij spatial data, omdat er geen totale orde mogelijk is. De meeste spatial databasesystemen maken gebruik van de R-tree of een variant hiervan. R-trees maken gebruik van *bounding*

[3]Compressie wordt *lossy* genoemd wanneer het comprimeren en decomprimeren niet de originele data teruggeeft, maar iets wat in de buurt komt van het origineel.

boxes als benadering van geometriën.

De GiST of *Generalized Search Tree* is een template index structuur die gebruikt kan worden om eender welke *search tree* te bouwen voor eender welk type van data. Alles wat een database beheerder moet doen om de GiST te kunnen gebruiken op zijn data types is het implementeren van (ten minste) vier functies. Met de GiST is het mogelijk om B+ trees, kd-trees, hB-trees, RD-trees of R-trees te bouwen. De GiST wordt momenteel ondersteund in PostGIS. In vergelijking met de R-tree is de GiST flexibeler (door zijn uitbreidbaarheid) en efficiënter (door de *lossy compression*).

Hoofdstuk 3

Open Source en commerciële databanken

3.1 Inleiding

In dit hoofdstuk worden enkele Open Source en commerciële databanken besproken. Er wordt voor elk systeem een korte beschrijving gegeven, waarna de fundamenten van GIS (zie hoofdstuk 2) besproken worden. Dit wordt gedaan voor Oracle (Locator en Spatial), ESRI's ArcSDE, PostGIS/PostgreSQL en MySQL. Eerst wordt er gezegd in welke mate de Simple Features Specification ondersteund wordt. Daarna worden transacties en locking in het specifieke systeem uitgelegd, gevolgd door de ondersteuning van spatiale indexen.

Tenslotte volgt een korte sectie over de manier waarop de spatiale ondersteuning geïntegreerd is in de databank en een overzicht van ondersteuning van databanken door softwaretoepassingen. Daarna wordt er in hoofdstuk 4 een vergelijkende performantiebenchmark uitgevoerd met PostGIS/PostgreSQL en MySQL.

3.2 Oracle

Er zijn in Oracle Database ([15]) twee spatiale databanken beschikbaar: Oracle Locator en Oracle Spatial.

Oracle Locator is een standaard onderdeel van de Oracle database en heeft dezelfde geometrische datastructuur en indexmogelijkheden als Oracle Spatial, maar biedt minder functionaliteiten. Zo ontbreken bijvoorbeeld de functies en procedures voor coördinatentransformatie, ruimtelijke aggregaties en

tuning in Oracle Locator. Met andere woorden, Oracle Locator is een subset van Oracle Spatial.

Oracle Locator kost niets als men al gebruik maakt van Oracle Database Standard en Enterprise. Dit in tegenstelling tot Oracle Spatial, dat als uitbreiding van Oracle Database Enterprise gekocht moet worden.

Oracle Spatial is een uitbreiding van de Oracle Locator, gericht op toepassing in GIS-oplossingen. Het laat toe om ruimtelijke gegevens op te slaan, te controleren, te indexeren, te bevragen en te bewerken. Oracle Spatial biedt ook onder meer mogelijkheden voor het leggen van nieuwe geometrische verbanden, transformatie van cordinatensystemen en opslag van lineaire meetgegevens.

3.2.1 Simple Features Specification en Oracle Spatial/Locator

Oudere versies van Oracle Spatial en Locator verschillen veel met de OpenGIS Simple Features Specification. Zo is de functionele namespace bijna compleet anders, worden andere namen voor de object types gebruikt en wordt er gebruik gemaakt van een eigen representatie (in plaats van well-known text/binary). Deze representatie heet SDO_GEOMETRY of Spatial Data Option. Het gebruikt, net als well-known text/binary, één veld in een tabel. Een HHCODE of Helical Hyperspatial code wordt gebruikt om de ruimtelijke gegevens op te slaan. Het nadeel van SDO_GEOMETRY is dat deze niet zo overdraagbaar is als WKT of WKB.

Vanaf Oracle Database release 10g (version 10.1.0.4) zijn Oracle Spatial en Locator echter volledig conformant met de Simple Features Specification. Het ondersteunt alle geometrische types en functies, WKT en WKB. De twee metadata tabellen (`SPATIAL_REF_SYS` en `GEOMETRY_COLUMNS`) worden ook ondersteund, maar ze hebben een andere benaming. `SPATIAL_REF_SYS` heet `MDSYS.CS_SRS` en `GEOMETRY_COLUMNS` heet `SDO_GEOM_METADATA`.
`SDO_GEOM_METADATA` heeft 3 views:
`USER_SDO_GEOM_METADATA`, `ALL_SDO_GEOM_METADATA`
en `DBA_SDO_GEOM_METADATA`.

3.2.2 Transacties in Oracle

Oracle gebruikt row-level en table-level locking. Het gebruikt twee soorten sublocks: DML locks en DDL locks. DML of data manipulation language

locks zorgen ervoor dat data niet gemanipuleerd kan worden. DDL of data dictionary language locks verbieden structurele veranderingen aan een tabel.

Bij **row-level locking** kan elke rij binnen een tabel apart gelocked worden. Alleen het proces dat de lock aanvraagt kan de rij nog aanpassen. Alle andere rijen in de tabel kunnen nog aangepast worden door de andere processen. Alle rijen (inclusief degene die aangepast wordt) kunnen gelezen worden. Wanneer andere processen rijen lezen waarop een lock ligt, dan zullen ze de oude versie zien totdat de veranderingen doorgevoerd worden met een `COMMIT` of afgebroken met een `ROLLBACK`.

Wanneer een proces een row-level lock aanvraagt, dan wordt er een data manipulation language (DML) lock op de rij geplaatst. Deze lock zorgt ervoor dat het aanpassen van deze rij onmogelijk wordt voor andere processen. De DML lock blijft bestaan totdat de transactie afgebroken of gecommit wordt. Naast de DML lock, wordt er ook een data dictionary language (DDL) lock op de tabel van de betreffende rij geplaatst. Deze lock verbiedt, zoals eerder vermeld, structurele veranderingen aan de tabel, zoals een `DROP` of een `ALTER TABLE`. Ook deze DDL lock blijft bestaan totdat de transactie afgebroken of gecommit wordt. Oracle zorgt automatisch voor een row-level lock bij een transactie die een individuele rij aanpast met een `INSERT`, `UPDATE`, `DELETE` of `SELECT` (met `FOR UPDATE`) statement.

Naast row-level locking laat Oracle ook **table-level locking** toe. Hierbij worden de tabellen als een geheel gelocked. Alleen het proces dat de lock aanvraagt kan de tabel nog aanpassen. Alle processen kunnen de tabel nog lezen. Wanneer andere processen rijen lezen waarop een lock ligt, dan zullen ze de oude versie zien totdat de veranderingen doorgevoerd worden met een commit. Oracle zorgt automatisch voor een table-level lock bij een transactie die een volledige tabel aanpast met een `INSERT`, `UPDATE`, `DELETE`, `SELECT` (met `FOR UPDATE`) of `LOCK TABLE` statement. Table-level locking gebruikt ook DML en DDL sublocks, om de data en tabelstructuur te beschermen.

Er zijn verschillende table lock modes: row share (RS), row exclusive (RX), share (S), share row exclusive (SRX) en exclusive (X). Deze staan in volgorde van „strengheid”: de meest soepele eerst, de strengste als laatste.

Row share table locks (RS) betekenen dat de transactie rijen wil gaan aanpassen. Het wordt automatisch verkregen nadat één van de volgende statements wordt uitgevoerd:

```
SELECT ... FROM table ... FOR UPDATE OF ... ;
LOCK TABLE table IN ROW SHARE MODE;
```

Een row share lock laat nog alles toe aan andere processen, behalve het leggen van een exclusieve write lock:

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

Row exclusive table locks (RX) betekenen dat de transactie één of meerdere rijen zal aanpassen. Het wordt automatisch verkregen nadat één van de volgende statements wordt uitgevoerd:

```
INSERT INTO table ... ;
UPDATE table ... ;
DELETE FROM table ... ;
LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

Een row share lock laat nog alles toe aan andere processen, behalve het leggen van een exclusieve read/write lock:

```
LOCK TABLE table IN SHARE MODE;
LOCK TABLE table IN SHARE EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;
```

Share table locks (S) zorgen ervoor dat andere transacties enkel een `SELECT`-query kunnen uitvoeren, specifieke rijen kunnen locken met `SELECT ... FOR UPDATE` of zelf ook een share table lock kunnen aanvragen. Meerdere transacties kunnen dus een share table lock hebben op een tabel. Enkel als er maar één share table lock op een tabel ligt, kan de transactie die de lock heeft aanpassingen uitvoeren. Als er meerdere share table locks zijn, dan kunnen de transacties enkel lezen. Een share table lock wordt verkregen nadat de volgende statement wordt uitgevoerd:

```
LOCK TABLE table IN SHARE MODE;
```

Een share table lock verbiedt het aanpassen van de tabel door andere transacties en het uitvoeren van één van de volgende statements:

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;
LOCK TABLE table IN ROW EXCLUSIVE MODE;
```

Share row exclusive table locks (SRX) verbieden andere transacties om een tabel aan te passen. Andere transacties mogen wel nog `SELECT`-queries uitvoeren en specifieke rijen locken met `SELECT ... FOR UPDATE`. Er kan op elk moment maar hoogstens één transactie een share row exclusive table lock hebben op een tabel. Een share row exclusive table lock wordt verkregen nadat de volgende statement wordt uitgevoerd:

```
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
```

Een share row exclusive table lock verbiedt het aanpassen van de tabel door andere transacties en het uitvoeren van één van de volgende statements:

```
LOCK TABLE table IN SHARE MODE;
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
LOCK TABLE table IN ROW EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;
```

Exclusive table locks (X) zijn de meest restrictieve locks. Andere transacties mogen alleen nog `SELECT`-queries (zonder `FOR UPDATE`) uitvoeren. Er kan op elk moment maar hoogstens één transactie een exclusive table lock hebben op een tabel. Een exclusive table lock wordt verkregen nadat de volgende statement wordt uitgevoerd:

```
LOCK TABLE table IN EXCLUSIVE MODE;
```

3.2.3 Indexen in Oracle Spatial

Oracle Spatial gebruikt R-trees en Quadtrees als spatiale indexen. Een quadtree is een hiërarchische boomstructuur waarbij elke knoop vier kinderen heeft. Een quadtree wordt opgebouwd door een twee dimensionale ruimte recursief op te delen in vier delen, totdat het aantal spatiale objecten in elk deel kleiner is dan een bepaalde drempelwaarde.

R-trees krijgen vaak de voorkeur, omdat Quadtrees niet met geodetische data[1] kan werken. Quadtrees zijn beter bij niet-geodetische data die veel geupdate moet worden, wanneer update performantie van groot belang is.
Een spatiale index kan in Oracle gemaakt worden met

```
CREATE INDEX myindex ON mytable(mygeometrycolumn)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
```

Hierbij is `myindex` een naam voor de index, `mytable` de tabel en `mygeometrycolumn` de kolom die de geometrische data bevat. Deze query maakt een R-tree aan. Indien het gewenst is om een Quadtree te maken, dan moeten enkele parameters toegevoegd worden:

```
CREATE INDEX myindex ON mytable(mygeometrycolumn)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('sdo_level=6');
```

[1] geodetische data=data die betrekking heeft tot de vorm van de aardbol, met een bepaalde breedte -en lengtegraad

Wanneer `sdo_level` groter is dan `0`, dan wordt een Quadtree in plaats van een R-tree gemaakt. Omdat het uitleggen van de precieze betekenis van `sdo_level` te ver zou leiden, wordt er verwezen naar de „Oracle Spatial User's Guide and Reference" ([16]).
Een index kan simpelweg gedropt worden met

```
DROP INDEX myindex;
```

Hierbij is `myindex` de naam van de index.

3.3 ESRI's ArcSDE

ArcSDE ([5]) is een applicatie en middle-ware server dat zich tussen een client toepassing en een database server bevindt. Het werkt als een gateway die zorgt voor het beheren van spatiale data in een Relational Database Management System (RDBMS) en het beschikbaar maken van spatiale data aan diverse client toepassingen. Mogelijke client toepassingen zijn ArcInfo, ArcView, ArcIMS, ArcGIS Server of MapObjects. De onderliggende database server kan Oracle, Micrsoft SQL Server, Informix, Sybase of IBM DB2 zijn.

3.3.1 Simple Features Specification en Esri's ArcSDE

ArcSDE volgt de Simple Features Specification en is volledig geslaagd voor de 'conformance test'. Het ondersteunt alle geometrische types en functies, WKT en WKB. Naast de metadata tabellen `SPATIAL_REF_SYS` en `GEOMETRY_COLUMNS` gebruikt het ook nog een eigen tabel, genaamd `LAYERS`.

3.3.2 Transacties in ArcSDE

ArcSDE kent 3 soorten locks: object locks, table locks en row locks. Er wordt bij elk van deze drie een onderscheid gemaakt tussen shared en exclusive locks, zoals bij de andere systemen. Alle locks worden beheerd door de client toepassing (ArcInfo, ArcView, ArcIMS, ...).

Object locks komen niet voor in de andere besproken systemen, omdat deze de locking implementatie op een lager niveau hebben. Een shared object lock wordt verkregen wanneer een object aangepast wordt. Omdat het om een shared lock gaat, kunnen meerdere transacties deze lock delen. Pas wanneer de aanpassingen doorgevoerd (gecommit) worden, wordt het shared lock gepromoveerd naar een exclusive lock. Dit zorgt ervoor dat geen andere transacties een shared lock kunnen hebben op het object, of met andere

woorden, dat geen andere transactie het object kan aanpassen.

Omdat ArcSDE middleware is, hangt de implementatie van transacties en locking ondersteuning af van het onderliggende databasesysteem. Zo gebeurt het bij Oracle 8 volledig in het geheugen, terwijl bij Oracle 9I gebruik wordt gemaakt van systeem tabellen en stored procedures.

3.3.3 Indexen in ArcSDE

De spatiale index van ArcSDE hangt af van het onderliggende systeem. Het gebruikt meestal een multi-level grid. Zoals uitgelegd wordt Sectie 3.6 over de integratie van de spatial support is het in ArcSDE vaak ondoenbaar om een spatiale index zoals de R-tree of de GiST te implementeren. Alleen wanneer het onderliggende databasesysteem geometrische types en spatiale indexes ondersteunt, dan kan ArcSDE deze gebruiken. Dit is het geval bij IBM Informix en Oracle Spatial/Locator[2].

3.4 PostGIS

PostGIS ([20]) is een uitbreiding van PostgreSQL ([19]), een object-relationele database. Het is net als PostgreSQL een Open Source project. PostGIS is wat Oracle Spatial is voor Oracle.
PostGIS is inbegrepen in PostgreSQL, maar het is aangeraden om de nieuwste versie van de PostGIS website ([20]) te downloaden, omdat de versie in PostgreSQL meestal een paar versies achterloopt.

3.4.1 Simple Features Specification en PostGIS

PostGIS is gebouwd rond de Simple Features Specification en is bij versie 1.0 ingediend voor het 'conformance testing'. Alhoewel de resultaten hiervan nog niet zijn gepubliceerd, volgt PostGIS de specificatie volledig. Het ondersteunt alle geometrische types en functies. Naast WKT en WKB ondersteunt PostGIS ook de uitgebreidere versies EWKB en EWKT (Extended Well-Known Binary en Text). EWKB en EWKT voegen 3dm, 3dz en 4d coördinaten toe en ze embedden SRID informatie in de representatie. In de toekomst kunnen EWKB en EWKT mogelijk veranderen, omdat ze kunnen conflicteren met de OGC standaard als deze wordt uitgebreid. De twee metadata tabellen

[2]zie http://tinyurl.com/ocsz4 voor een volledig overzicht van beschikbare spatiale indexen

(SPATIAL_REF_SYS en GEOMETRY_COLUMNS) zijn ook geïmplementeerd. Deze worden aangemaakt tijdens de installatie van PostGIS.

3.4.2 Transacties in PostgreSQL/Postgis

PostgreSQL gebruikt een mechanisme genaamd MVCC of MultiVersion Concurrency Control. Het werkt als row-level locking, maar er worden meerdere versies van de data worden bijgehouden. Zo kan men een lock op tabelrijen houden in een sessie en deze tabelrijen ongeroerd in een andere sessie weergeven. MVCC is beter dan row-level locking, omdat een lezende transactie nooit gehinderd wordt door een schrijvende transactie.

In PostgreSQL beginnen transacties met de statement BEGIN en eindigen ze met COMMIT of ROLLBACK. Het COMMIT-commando wordt gebruikt wanneer de transactie succesvol was, waarna de betreffende tabellen worden aangepast. ROLLBACK wordt gebruikt als er iets misloopt. Er wordt dan teruggegaan naar oude de status (voor WORK aangeroepen werd).

Naast MVCC is row-level en table-level locking ook aanwezig in PostgreSQL. Dit is gedaan voor applicaties die zich niet makkelijk kunnen aanpassen aan het MVCC gedrag.

Table-level locking kan gedaan worden met het LOCK commando. Een transactie met een SELECT query vereist impliciet een *shared lock* (ACCESS SHARE). Een *exclusive lock* kan expliciet genomen worden met ACCESS EXCLUSIVE. Een *exclusive lock* zal het aanvragen van een *shared lock* verhinderen.

Bij **row-level locking** is er ook een *shared* en een *exclusive* type. Een exclusive lock wordt automatisch verkregen wanneer een rij aangepast of verwijderd wordt. De lock blijft bestaan totdat de transactie wordt afgesloten met een COMMIT of ROLLBACK commando. De betreffende rij kan wel nog door een andere transactie gelezen wordt, maar niet aangepast. Om een exclusive row-level lock te verkrijgen zonder de rij aan te passen, kan SELECT ... FOR UPDATE gebruikt worden. Een shared lock kan verkregen worden met SELECT ... FOR SHARE. Op een rij met een shared lock kunnen geen updates, verwijderingen of exclusieve locks toegepast worden.

3.4.3 Indexen in PostgreSQL/Postgis

PostgreSQL gebruikt de GiST en de R-tree met *quadratic splitting* als spatiale index. De GiST implementatie is een R-tree, maar met de voordelen van een

GiST. Quadratic splitting werd besproken in Sectie 2.4.2.

GiST indexen hebben twee voordelen tegenover traditionele R-tree indexen in PostgreSQL. Ten eerste zijn ze „null safe": ze kunnen kolommen indexeren die „null" waarden bevatten. En ten tweede ondersteunen ze, dankzij de key compressie, „lossiness", wat belangrijk is wanneer er met GIS objecten wordt gewerkt die groter zijn dan de *8K page size* van PostgreSQL. Door deze „lossy" compressie kan PostgreSQL enkel de bounding box van een object opslaan. Als de objecten groter worden dan de *page size* van PostgreSQL, dan zal het opbouwen van een R-tree index mislukken.
In PostgreSQL/PostGIS kan een GiST aangemaakt worden met

```
CREATE INDEX myindex ON mytable
  USING GIST ( mygeometrycolumn GIST_GEOMETRY_OPS );
```

Hierbij is `myindex` een naam voor de index, `mytable` de tabel en `mygeometrycolumn` de kolom die de geometrische data bevat.
`GIST_GEOMETRY_OPS` betekent dat de GiST met geometriën zal werken. Deze parameter kan verschillende waarden hebben. Een lijst met mogelijke waarden (en dus mogelijke indexen in PostGIS) kan opgevraagd worden met de query

```
SELECT opcname FROM pg_opclass;
```

Door

```
USING GIST ( mygeometrycolumn GIST_GEOMETRY_OPS);
```

te vervangen door

```
USING RTREE (mygeometrycolumn BOX_OPS);
```

zal er een R-tree in plaats van een GiST gemaakt worden. Nadat de index gebouwd is, is het belangrijk dat PostgreSQL gedwongen wordt om tabel statistieken te verzamelen, met:

```
VACUUM ANALYZE;
```

Om de spatiale index in PostgreSQL te benutten, moeten de queries herschreven worden. Er moet gebruik worden gemaakt van de operator `&&`. Deze kan gelezen worden als „bounding box overlapt". Zo zal de query

```
SELECT a.ID,b.ID FROM polygoontabel1 a, polygoontabel2 b
  WHERE Touches(a.geometry,b.geometry);
```

ervoor zorgen dat voor elke geometrie in polygoontabel1 berekend wordt of deze geometrie een geometrie in polygoontabel2 „raakt". Als de query herschreven wordt tot

```
SELECT a.ID,b.ID FROM polygoontabel1 a, polygoontabel2 b
  WHERE a.geometry && b.geometry
  AND Touches(a.geometry,b.geometry);
```

dan zal voor elke geometrie in polygoontabel1 eerst gekeken of de bounding box van de geometrie overlapt met de bounding box van elke geometrie in polygoontabel2. Op deze manier wordt de zware berekening (`Touches(a.geometry,b.geometry)`) enkel uitgevoerd voor mogelijke kandidaten.

3.5 MySQL

MySQL ([13]) is de meest populaire Open Source database. Het bevat een serverapplicatie, genaamd mysqld (waarbij de d voor daemon staat). Er wordt met de server gecommuniceerd met behulp van een clientprogramma, bijvoorbeeld mysql, mysqldump of php. De combinatie met php is zeer populair op het internet.
MySQL 4.1 introduceerde spatiale functionaliteit in MySQL. Op dat moment was dit enkel het geval voor het MyISAM tabeltype. Sinds MySQL 5.0.16 is er ook ondersteuning bij andere tabeltypes, zoals InnoDB, NDB, BerkeleyDB en ARCHIVE. Momenteel wordt er ook gewerkt aan een nieuw tabeltype voor MySQL, namelijk solidDB. Dit zou vanaf de zomer van 2006 beschikbaar zijn in MySQL([14]). Het is voorlopig onduidelijk of deze spatiale ondersteuning zal hebben.

3.5.1 Simple Features Specification en MySQL

Sinds de 4.1 release van MySQL is er spatiale ondersteuning aanwezig in MySQL. MySQL Spatial Extensions volgt de Simple Features Specification gedeeltelijk. Er zijn enkele punten die nog niet zijn geïmplementeerd, maar er zijn plannen om deze in de toekomst toe te voegen.
In de recentste implementatie van MySQL (v5.1[3]) zijn er enkele onderdelen die (gedeeltelijk of volledig) verschillen van de Simple Features Specification:

1. **Functies**: Een aantal functies/methodes ontbreken:

[3]te downloaden op http://dev.mysql.com/downloads/mysql/5.1.html

- Enkele spatiale analyse functies: Buffer(afstand), ConvexHull(), Difference(andereGeometrie), Intersection(andereGeometrie), SymDifference(andereGeometrie) en Union(andereGeometrie).
- Alle functies die testen op spatiale relaties tussen geometriën: Contains, Crosses, Disjoint, Distance, Equals, Intersects, Overlaps, Related, Touches en Within. Voor deze functies (excl. Distance en Related) zijn er wel MBR[4]-gebaseerde alternatieven, maar deze volgen de Simple Features Specification niet.
- Bij Geometry: de methodes Boundary(), isEmpty() en isSimple().
- Bij LineString: de methodes IsRing().
- Bij MultiPolygon: de methodes Centroid() en PointOnSurface().

2. **Functienamen**: De OpenGIS functie Length() van LineString en MultiLineString heet GLength() in MySQL. Dit komt omdat er momenteel al een functie Length() is in MySQL die de lengte van een string bepaalt.

3. **SRID**: De SRID of Spatial Reference Identifier defineert de eigenschappen van het coördinaatstelsel. MySQL slaat de SRID van spatiale objecten wel op, maar gebruikt deze niet. Het coördinaat systeem wordt verondersteld Cartesisch te zijn. Dit is een serieuze tekortkoming, omdat het geodetische coördinaatstelsel (het coördinaatstelsel van de aarde) niet ondersteund wordt.

4. **Metadata tabellen/views**: De 2 metadata tabellen (`SPATIAL_REF_SYS` en `GEOMETRY_COLUMNS`) ontbreken in MySQL.

Buiten deze uitzonderingen zijn alle features van de Simple Features Specification in de Spatiale Extensies van MySQL 5.1 inbegrepen.

3.5.2 Transacties in MySQL

Het standaard tabel type van MySQL, MyISAM, heeft geen volwaardige ondersteuning voor transacties. MyISAM gebruikt table-level locking, wat wil zeggen dat een volledige tabel ontoegankelijk is tijdens een transactie. Row-level locking is mogelijk met MySQL wanneer een ander tabel type gebruikt wordt. Dit moet echter beslist worden tijdens het creëren van de tabel. De tabel types Berkeley DB en InnoDB ondersteunen row-level locking.

[4]MBR=Minimal bounding rectangles of kleinste omvattende rechthoeken; dit is hetzelfde als bounding boxes

In MySQL beginnen transacties met het statement `BEGIN WORK` en eindigen ze met `COMMIT` of `ROLLBACK`. Het `COMMIT`-commando wordt gebruikt wanneer de transactie succesvol was, waarna de betreffende tabellen worden aangepast. `ROLLBACK` wordt gebruikt als er iets misloopt. Er wordt dan teruggegaan naar de oude versie (voor `BEGIN WORK` aangeroepen werd).

Tabelrijen zullen een *shared lock* krijgen wanneer een `SELECT` query wordt uitgevoerd. Zo kunnen verschillende transacties een `SELECT` query op dezelfde tabel uitvoeren. De shared locks verdwijnen als de transacties afgesloten zijn met een `COMMIT` of `ROLLBACK` commando. Een *exclusive lock* kan verkregen worden door `FOR UPDATE` aan de `SELECT` query toe te voegen. Dit betekent dat enkel de huidige transactie de tabelrij kan lezen, totdat de transactie beëindigd is.

Er wordt momenteel gewerkt aan een nieuw tabeltype voor MySQL, genaamd solidDB ([14]). Dit nieuw tabeltype zou een zeer robuuste ondersteuning hebben voor transacties en locking. Het ondersteunt ook de MVCC of MultiVersion Concurrency Control die PostgreSQL biedt. solidDB zou in de zomer van 2006 beschikbaar zijn voor MySQL. Voorlopig is er niet meer informatie beschikbaar.

3.5.3 Indexen in MySQL

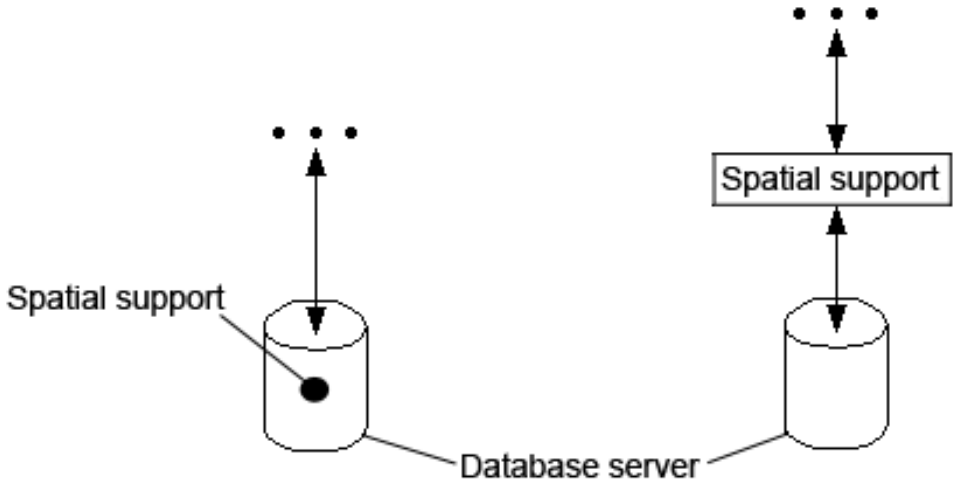
MySQL gebruikt R-trees met *quadratic splitting*. Dit werd besproken in Sectie 2.4.2. Een R-tree kan in MySQL aangemaakt worden met

```
ALTER TABLE mytable ADD SPATIAL INDEX(mygeometrycolumn);
```

Hierbij is `mytable` de tabel en `mygeometrycolumn` de kolom met de geometrische data. De index kan verwijderd worden met

```
ALTER TABLE mytable DROP INDEX mygeometrycolumn;
```

Ook hierbij is `mytable` de tabel en `mygeometrycolumn` de kolom met de geometrische data. MySQL zal de R-tree indien mogelijk automatisch gebruiken, er moeten hiervoor geen speciale parameters in de query gebruikt worden.



Figuur 3.1: Integratie van spatial support

3.6 Integratie van spatial ondersteuning

Deze Sectie bespreekt hoe de spatial support binnen het relationele databasesysteem geïntegreerd is. Zoals in Figuur 3.1 te zien is, kan dit in de kern van het databasesysteem (links) of bovenop het databasesysteem (rechts) gebeuren. De manier waarop dit gedaan is, heeft een groot effect op het gebruik en de performantie van het systeem. We kunnen de systemen zo grofweg in twee groepen onderverdelen. Oracle Spatial/Locator, PostGIS en MySQL zitten in de groep van de 'echte' spatial databasesystemen. ArcSDE zit alleen in de andere groep, de groep van de 'middleware'. Op Figuur 3.1 is links de eerste groep geïllustreerd en rechts de tweede groep.

De spatial support van Oracle Spatial/Locator, PostGIS en MySQL bevindt zich in de kern van de database. De geometrische objecten zijn gedefinieerd als types die evenwaardig zijn met de andere types van de database. Dit heeft veel gevolgen. Spatial indexing en query optimalisatie kunnen gebeuren in de kern van de database, onder het abstractie-niveau van tabellen. Er kan ook gebruik gemaakt worden van andere faciliteiten die een gemiddeld databasesysteem biedt, zoals security (specifieke rechten voor gebruikers), versioning (het bijhouden van versies en het toelaten om terug te gaan naar een oudere versie), parallellisme (meerdere acties parallel uitvoeren), clustering (het samenwerken van meerdere fysieke servers op een laag niveau), ...

ArcSDE is 'middleware', wat wil zeggen dat het in een laag bovenop een databasesysteem draait. ArcSDE kan werken bovenop IBM's DB2, IBM's

Informix, Microsoft SQL Server en Oracle. ArcSDE slaat zijn geometrische objecten op als BLOBs van binaire data. Daardoor is ArcSDE beperkt tot de tabellen en SQL queries. Enkel ArcSDE kent dus de betekenis van deze objecten, de database zelf en eventuele applicaties die de database rechtstreeks gebruiken zien het gewoon als binaire data. Voor specifieke database faciliteiten (zoals veiligheid, clustering, ...) hangt ArcSDE af van het onderliggende databasesysteem.

Een belangrijk gevolg van dit onderscheid is de spatiale index die gemaakt kan worden door de spatiale database. Oracle, PostgreSQL (de RDBMS achter PostGIS) en MySQL gebruiken de R-tree (zie Sectie 2.4.2), een efficiënte spatiale indexeringsmethode. ArcSDE gebruikt een multi-level grid index. Deze spatiale indexeringsmethode is minder efficiënt dan de R-tree index, maar de enige redelijke methode gebruik makende van tabellen en SQL queries. Een R-tree implementeren voor ArcSDE zou zeer inefficiënt zijn.

Wat gezegd wordt in deze sectie is niet altijd het geval. IBM Informix heeft een Spatial Extender die geometrische types ondersteunt. Deze worden gebruikt door ArcSDE, waardoor er wel een R-tree gebruikt kan worden (de R-tree van Informix zelf). Bij Oracle heeft men de keuze uit drie verschillende datatypes: Oracle Spatial/Locator's geometry type (`SDO_GEOMETRY`), ArcSDE's compressed binary en OGC's Well Known Binary. Bij de eerste is het mogelijk voor ArcSDE om de R-tree of Quadtree van Oracle te gebruiken. Bij de andere twee is het weer beperkt tot tabellen en SQL queries, waardoor er een multi-level grid index wordt gebruikt. Bij de andere systemen (Microsoft SQL Server en IBM DB2) wordt er altijd een multi-level grid index gebruikt.

3.7 Ondersteuning door toepassingen

In deze Sectie wordt voor een aantal desktop -en server-toepassingen en API's besproken welke databases ondersteund worden. Het resultaat wordt in tabel 3.1 getoond. Deze gegevens zijn verzameld op vraag van GIM, omdat het voor hen nuttig is.

In de kolommen staan de verschillende toepassingen, onderverdeeld in drie categoriën: desktop, mapserver en API. In de rijen staan de belangrijkste databasesystemen en Shapefile. Een r of r/w wil zeggen dat deze in respectievelijk read-only of read/write mode ondersteund wordt. Een schuingedrukte *r* of *r/w* betekent dat het een uitbreiding is en dat er mogelijk een plugin geïnstalleerd moet worden. Een leeg vakje betekent dat het databasesysteem niet met de toepassing kan werken.

	Desktop						Mapserver					API	
	ArcGIS Desktop	GeoMedia	GeoMedia Pro	MapInfo Pro	Jump	uDig	ArcIMS	WebMap	MapXtreme	GeoServer	deegree	GeoTools	JTS
PostGIS					*r*	r/w				r/w	r/w	r	r/w
Oracle	r	r	r/w	r/w		*r/w*		r	r/w	r/w	r/w	r	r/w
MS SQL	r	r	r/w	r/w				r					
ArcSDE	r/w					*r/w*	r/w			r/w		r	r/w
IBM DB2		r	r/w			r/w			r/w				
MySQL										r/w	r/w	r	
Shapefile		r	r		r/w	r/w		r		r/w		r	

Tabel 3.1: GIS toepassingen en hun ondersteuning

3.8 Conclusie

In dit hoofdstuk zijn vier spatiale databanken in meer detail besproken. Aan de ene kant staan Oracle Spatial/Locator en ESRI's ArcSDE, twee commerciële databanken. Aan de andere kant staan PostGIS en MySQL, twee Open Source databanken.

Oracle Spatial/Locator heeft een goede, volledige implementatie van de Simple Features Specification en een uitstekende ondersteuning voor transacties en locking. Als indexstructuur is er keuze tussen de R-tree en de Quadtree. Oracle heeft echter een groot nadeel: het is erg duur. ESRI's ArcSDE behoort tot een andere klasse, die van de middle-ware. Het werkt als een gateway tussen een databank en clienttoepassingen. Het volgt de Simple Features Specification volledig en ondersteunt transacties en locking. De indexstructuur in ArcSDE hangt af van het onderliggende systeem.

Alhoewel dat ze volledig gratis zijn, zijn PostGIS en MySQL een waardig alternatief voor de commerciële pakketten. Vooral PostGIS biedt alles wat een betalende database biedt. Het heeft een goede, volledige implementatie van de Simple Features Specification en een goede ondersteuning voor transacties en locking. PostGIS ondersteunt ook als enige de indexstructuur GiST. MySQL's ondersteuning van de Simple Features Specification is onvolledig. Het standaard tabeltype (MyISAM) heeft geen volwaardige ondersteuning voor transacties en locking, maar er zijn alternatieve tabeltypes die dit wel hebben. Als indexstructuur gebruikt MySQL de R-tree. Het nadeel aan PostGIS en MySQL is dat de documentatie en klantenondersteuning minder goed is. Dit wordt echter grotendeels goedgemaakt door de grote bekendheid van de twee op het internet, waardoor er veel informatie te vinden is.

Hoofdstuk 4

Praktisch luik

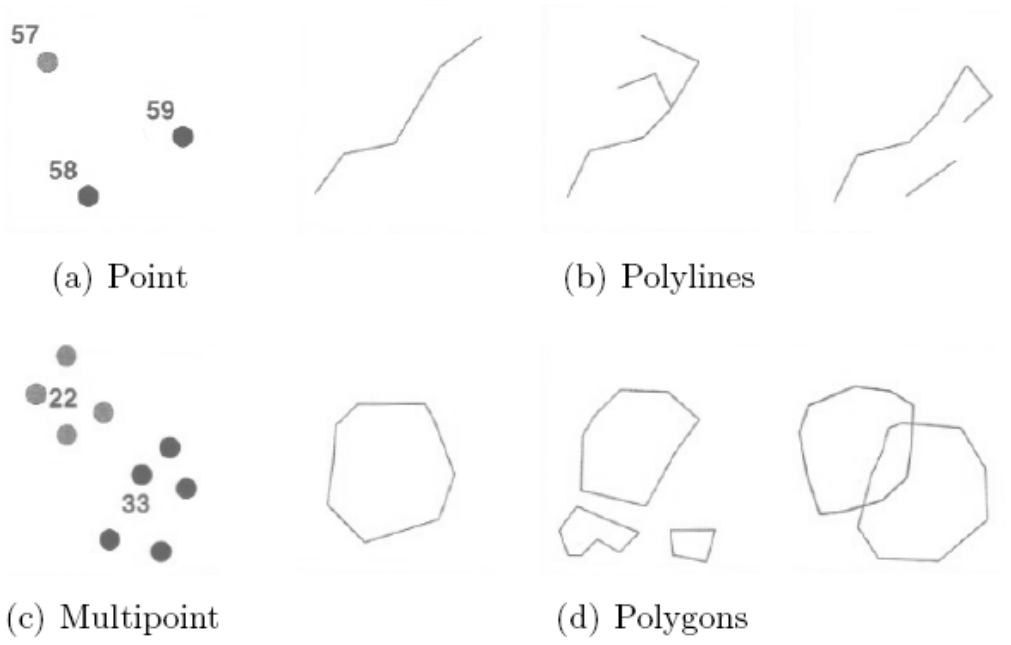
4.1 Inleiding

In deze Sectie wordt de performantie van de spatiale mogelijkheden in MySQL en PostGIS/PostgreSQL vergeleken. Eerst worden de gebruikte datasets besproken. Daarna worden de queries en de resultaten gegeven.
Tijdens het uitvoeren van deze benchmark, zijn er enkele problemen geweest. Omdat deze interessant zijn voor GIM, zijn ze opgenomen in Bijlage A.3.

4.2 Datasets

Voor het importeren van data wordt gebruik gemaakt van zogenaamde shapefiles. Shapefiles worden gebruikt om geometriën voor te stellen en zijn ontwikkeld door ESRI (zie [4]). Een shapefile stelt exact één layer voor en bevat enkel vector coördinaten. Shapefiles ondersteunen points, multipoints, polylines en polygons (zie Figuur 4.1). Polylines en polygons mogen uit meerdere delen bestaan, die niet verbonden hoeven te zijn en eventueel kunnen overlappen. Een shapefile kan maar één type geometrie tegelijkertijd bevatten.
Er worden in deze benchmark twee datasets gebruikt:
continents en **provinces**.

- De dataset **continents** stelt alle continenten van de wereld voor en bevat 8 polygonen. De shapefile van deze set is 2.8 mb groot. De drie kolommen van deze tabel die hier van belang zijn *id*, *description* en *ogc_geom*. *id* bevat een unieke ID, *description* bevat de naam van het continent en *ogc_geom* bevat de geometrie in kwestie.



Figuur 4.1: De verschillende types die door shapefile ondersteund worden

- De dataset **provinces** stelt alle provincies van de wereld voor en bevat 43949 polygonen. De shapefile van deze set is 44.5 mb groot. Bij deze dataset zijn dezelfde drie kolommen van belang als bij de eerste dataset: *id*, *description* en *ogc_geom*.

Er waren ook grotere datasets beschikbaar, maar door problemen met MySQL (zie Bijlage A.3) konden deze uiteindelijk niet gebruikt worden. Beiden shapefiles werden geconverteerd naar een uitvoerbaar SQL script (met extensie .sql) en daarna ingeladen in de database.

4.3 Queries en resultaten

In deze subsectie zullen de queries en de resultaten besproken worden. Eerst worden twee alfanumerieke queries uitgevoerd, daarna volgen de spatiale queries. Alfanumerieke queries maken enkel gebruik van niet-spatiale gegevens, terwijl spatiale queries wel gebruik maken van spatiale gegevens. Een typisch voorbeeld van een alfanumerieke query is „Hoeveel provincies zijn er in België?". Een voorbeeld van een spatiale query is „Wat is de grootste provincie van België?".
Nu volgen de queries die gebruikt zijn in de benchmark. De eerste twee que-

ries zijn alfanumeriek.

Query 1

- Hoeveel provincies zitten er in de database?
- `SELECT COUNT(*) FROM provinces;`

Query 2

- Geef alle provincies en sorteer deze alfabetisch.
- ```
  SELECT description AS name FROM provinces
    ORDER BY description;
  ```

De testresultaten van deze eerste twee queries worden getoon in Tabel 4.1. PostGIS presteert het best bij beide queries, alhoewel het verschil bij query 2 miniem is. De volgende twee queries gebruiken simpele geometrische functies.

Query	PostGIS	MySQL
1	500ms	1300ms
2	2201ms	2203ms

Tabel 4.1: Testresultaten van queries 1 en 2

In query 3 wordt de oppervlakte van alle continenten gevraagd en in query 4 de omtrekt van alle provincies in de database. De spatiale functies `AREA`, `LENGTH` en de andere gebruikte functies in dit hoofdstuk worden uitgelegd in Bijlage A.1.

Query 3

- Wat is de oppervlakte van alle continenten?
- `SELECT AREA(ogc_geom) FROM continents;`

Query 4

- Wat is de omtrek van alle provincies?
- PostGIS:

  ```
  SELECT LENGTH(ogc_geom) FROM provinces;
  ```

  MySQL:
  ```

```
SELECT GLENGTH(ogc_geom) FROM provinces;
```

De testresultaten van deze twee queries worden getoond in Tabel 4.2. Hierbij valt het op de MySQL veel beter is dan PostGIS, wanneer simpele geometrische functies gebruikt worden.

Query	PostGIS	MySQL
3	297ms	126ms
4	2188ms	490ms

Tabel 4.2: Testresultaten van queries 3 en 4

Nu volgen een aantal queries die een spatiale index aanmaken. Alleen queries met bounding-box-gebaseerde operatoren kunnen een spatiale index benutten. Standaard maken PostgreSQL/PostGIS en MySQL geen spatiale index aan bij spatiale data. Dit moet door de gebruiker zelf gedaan worden. In PostgreSQL/PostGIS kan dit met:

```
CREATE INDEX myindex ON provinces
  USING GIST ( the_geometry GIST_GEOMETRY_OPS );
```

Nadat de index gebouwd is, is het belangrijk dat PostgreSQL gedwongen wordt om tabel statistieken te verzamelen, met:

```
VACUUM ANALYZE;
```

PostgreSQL heeft deze tabel statistieken nodig bij complexere queries. Het is aangeraden om dit statement altijd uit te voeren wanneer een tabel wordt aangemaakt of drastisch gewijzigd wordt. Meer informatie over deze twee statements kan gevonden worden in Sectie 3.4.3.
In MySQL kan een spatiale index gebouwd worden met:

```
ALTER TABLE provinces ADD SPATIAL INDEX(ogc_geom);
```

Dit is enkel mogelijk wanneer de kolom met de geometrische data geen null-waarden bevat. De kolom moet dus van het type `GEOMETRY NOT NULL` zijn. Dit kan aangepast worden met:

```
ALTER TABLE provinces
  CHANGE ogc_geom ogc_geom GEOMETRY NOT NULL;
```

De GiST index van PostgreSQL is „NULL safe”, wat wil zeggen dat de kolom null-waarden mag bevatten.
Het droppen van een spatiale index gebeurt in PostgreSQL met

```
DROP INDEX myindex;
```

en in MySQL met

```
ALTER TABLE provinces DROP INDEX ogc_geom;
```

Het maken van een spatiale index voor de tabel **continents** verloopt op dezelfde manier. Omdat deze laatste tabel zo klein is en de performantie niet veelzeggend, worden de testresultaten hiervan weggelaten. De performantie van het maken van een spatiale index voor de tabel **provinces** wordt gemeten voor de volgende queries en wordt getoond in Tabel 4.3.

Query 5

- Maak een spatiale index aan.
- PostGIS:

  ```
  CREATE INDEX myindex ON provinces
    USING GIST ( ogc_geom GIST_GEOMETRY_OPS );
  VACUUM ANALYZE;
  ```

  MySQL:

  ```
  ALTER TABLE provinces ADD SPATIAL INDEX(ogc_geom);
  ```

Query 6

- Verwijder de spatiale index.
- PostGIS:

  ```
  DROP INDEX myindex;
  ```

  MySQL:

  ```
  ALTER TABLE provinces DROP INDEX ogc_geom;
  ```

De testresultaten van deze twee queries worden getoond in Tabel 4.3. Het aanmaken van de spatiale index verloopt sneller bij MySQL. PostGIS vereist ook dat er een `VACUUM ANALYZE;` wordt uitgevoerd. Dit doet meer dan enkel de spatiale index opbouwen, waardoor het resultaat vergelijken met MySQL niet helemaal eerlijk zou zijn. Het droppen van de index bij PostGIS neemt

Query	PostGIS	MySQL
5	1731ms+1895ms=3626ms	2550ms
6	≤1ms	2430ms

Tabel 4.3: Testresultaten van queries 5 en 6

vrijwel geen tijd in beslag, terwijl het bij MySQL bijna evenlang duurt als het opbouwen van de index.
Nu volgen de geteste queries die een spatiale index benutten. Enkel queries met bounding-box-gebaseerde operatoren (zoals &&) kunnen gebruik maken van een spatiale index. Een functie als distance() kan dus geen gebruik maken van een spatiale index.
Query 7 en 8 vragen beiden alle ID's op van provincies in Europa op. Query 7 gebruikt de spatiale index, query 8 niet. In query 8 wordt de index verwijderd bij PostGIS en genegeerd bij MySQL (met `IGNORE INDEX` in het `FROM` gedeelte van de query). De &&-operator uit query 7 voor PostGIS betekent zoveel als „bounding box overlapt".

Query 7

- Geef de ID's van de provincies in Europa. Met index.
- PostGIS:

```
SELECT id FROM provinces WHERE ogc_geom &&
  (SELECT ogc_geom FROM continents
   WHERE description = 'Europe');
```

- MySQL:

```
SELECT id FROM provinces WHERE
  (MBRContains((SELECT ogc_geom FROM continents
   WHERE description like 'Europe'),ogc_geom));
```

Query 8

- Geef de ID's van de provincies in Europa. Zonder index.
- PostGIS:

```
DROP INDEX myindex;
SELECT id FROM provinces WHERE ogc_geom &&
  (SELECT ogc_geom FROM continents
   WHERE description = 'Europe');
```

- MySQL:

```
SELECT id FROM provinces IGNORE INDEX (ogc_geom) WHERE
  (MBRContains((SELECT ogc_geom FROM continents
   WHERE description like 'Europe'),ogc_geom));
```

De testresultaten van deze twee queries worden getoond in Tabel 4.4. Met spatiale index is het verschil tussen PostGIS en MySQL klein. Zonder index presteert PostGIS echter een pak beter dan MySQL.

Query	PostGIS	MySQL
7	8862ms	9093ms
8	18916ms	64590ms

Tabel 4.4: Testresultaten van queries 7 en 8

In de volgende twee queries wordt er een zogenaamde „spatiale join" uitgevoerd. Voor elke provincie wordt gekeken of deze provincie snijdt met een andere provincie. De query heeft geen echte betekenis, omdat provincies niet horen te overlappen, maar de performantie ervan is wel interessant. Query 9 is de versie die een spatiale index gebruikt. Om PostGIS de index te laten gebruiken, moest de query herschreven worden met de „&&"-operator. Deze operator betekent zoveel als „bounding box overlapt". Query 10 gebruikt geen spatiale index.
Aan elke query is ook nog het stuk „`g.ID <> l.ID AND l.ID < 1000 AND g.ID < 1000`" toegevoegd in het `WHERE`-gedeelte, omdat de uitvoering anders te lang duurde (meer dan 5 uur zonder spatiale index).

Query 9

- Geef de koppels met ID's van provincies die snijden. Met index.
- PostGIS:

```
SELECT g.ID, l.ID FROM provinces g, provinces l
  WHERE g.ogc_geom && l.ogc_geom AND
  Intersects(g.ogc_geom,l.ogc_geom);
```

MySQL:

```
SELECT g.ID, l.ID FROM provinces g, provinces l
  WHERE Intersects(g.ogc_geom,l.ogc_geom);
```

Query 10

- Geef de koppels met ID's van provincies die snijden. Zonder index.
- PostGIS:

```
SELECT g.ID, l.ID FROM provinces g, provinces l
  WHERE Intersects(g.ogc_geom,l.ogc_geom);
```

MySQL:

```
SELECT g.ID, l.ID FROM provinces g IGNORE INDEX (ogc_geom),
  provinces l IGNORE INDEX (ogc_geom)
  WHERE Intersects(g.ogc_geom,l.ogc_geom);
```

De testresultaten van deze twee queries worden getoond in Tabel 4.5. Bij beide queries presteert MySQL veel beter dan PostGIS. MySQL kan dus beter overweg met „spatiale joins". Een mogelijke verklaring hiervoor is dat query planner van PostGIS/PostgreSQL minder efficiënt werkt dan die van MySQL.

Query	PostGIS	MySQL
9	42578ms	18820ms
10	808359ms	151492ms

Tabel 4.5: Testresultaten van queries 9 en 10

In de volgende twee queries wordt er eerst een polygoon geconstrueerd die een bounding box voorstelt. Daarna wordt voor elke provincie getest of deze in de bounding box ligt. Query 11 is de versie die een spatiale index gebruikt. Om PostGIS de index te laten gebruiken, moest de query herschreven worden met de „&&"-operator. Deze operator betekent zoveel als „bounding box overlapt". Query 12 gebruikt geen spatiale index. Dit wordt in MySQL gedaan door `IGNORE INDEX (ogc_geom)` toe te voegen in het `FROM`-gedeelte van de query.

Query 11

- Geef de ID's van de provincies die in polygoon *poly* liggen. Met index.
- PostGIS:

```
SET @poly = 'Polygon((-7 5,-6 5,-6 6,-7 6,-7 5))';
SELECT ID FROM provinces
  WHERE ogc_geom && GeomFromText(@poly)
  AND Contains(GeomFromText(@poly),ogc_geom);
```

MySQL:

```
SET @poly = 'Polygon((-7 5,-6 5,-6 6,-7 6,-7 5))';
SELECT ID FROM provinces
  WHERE Contains(GeomFromText(@poly),ogc_geom);
```

Query 12

- Geef de ID's van de provincies die in polygoon *poly* liggen. Met index.
- PostGIS:

```
SET @poly = 'Polygon((-7 5,-6 5,-6 6,-7 6,-7 5))';
SELECT ID FROM provinces
  WHERE Contains(GeomFromText(@poly),ogc_geom);
```

MySQL:

```
SET @poly = 'Polygon((-7 5,-6 5,-6 6,-7 6,-7 5))';
SELECT ID FROM provinces IGNORE INDEX (ogc_geom)
  WHERE Contains(GeomFromText(@poly),ogc_geom);
```

De testresultaten van deze twee queries worden getoond in Tabel 4.6. Query 11 (met spatiale index) duurt bij PostGIS en MySQL even lang. In query 12, wanneer er geen spatiale index is, is PostGIS een beetje sneller, maar dit verschil is verwaarloosbaar.

Query	PostGIS	MySQL
11	$\leq$1ms	$\leq$1ms
12	297ms	300ms

Tabel 4.6: Testresultaten van queries 11 en 12

4.4 Conclusie

In Tabel 4.7 worden de testresultaten van alle queries nog eens samen getoond. Over het algemeen presteert PostGIS beter dan MySQL, maar er zijn gevallen waar MySQL beter voor de dag komt (bij simpele geometrische functies en spatiale joins). Er kan geen eenduidige conclusie getrokken worden uit de resultaten van deze performantiebenchmark. PostGIS en MySQL hebben beiden hun sterkten en zwakten. Ook zijn er veel parameters die aangepast kunnen worden bij beide systemen, die de performantie kunnen beïnvloeden.
Het dient ook op te merken dat de documentatie van beide systemen niet perfect is. De documentatie van MySQL bevat bijvoorbeeld instructies over geometrische functies (zoals distance()) die helemaal niet geïmplementeerd zijn. De documentatie van PostGIS is niet heel duidelijk over het gebruik van spatiale indexen.

Query	PostGIS	MySQL
1	500ms	1300ms
2	2201ms	2203ms
3	297ms	126ms
4	2188ms	1037ms
5	3626ms	2550ms
6	$\leq$1ms	2430ms
7	8862ms	9093ms
8	18916ms	64590ms
9	42578ms	18820ms
10	808359ms	151492ms
11	$\leq$1ms	$\leq$1ms
12	297ms	300ms

Tabel 4.7: Alle testresultaten

Hoofdstuk 5

uDig

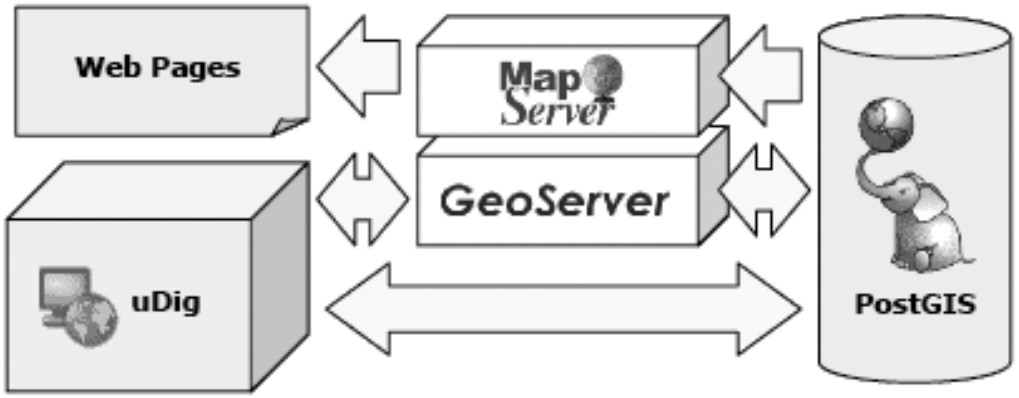
In dit Hoofdstuk van deze thesis worden de mogelijkheden van de uDig customisatie onderzocht. Allereerst wordt een inleiding gegeven over uDig. Dan wordt uitgelegd wat de structuur en uitbreidingsmogelijkheden van uDig zijn en hoe een simpele plugin in uDig geschreven kan worden. Het hoofdstuk wordt afgesloten met een conclusie.

5.1 Inleiding

uDig staat voor **U**ser Friendly **D**esktop **I**nternet **G**IS en is gratis te downloaden op het web (zie [22]). Het kan gezien worden als een Open Source viewer en editor van spatial data. Het is ontwikkeld als een plugin voor de Java ontwikkelingsomgeving Eclipse (2.1.4). uDig heeft twee gebruiken:

- Een Geospatiale applicatie, die „out of the box" gebruikt kan worden met een standaard workbench omgeving.
- Een platform waarmee ontwikkelaars nieuwe GIS applicaties kunnen afleiden, door herconfiguratie/programmatie van de Eclipse Plugin, daarbij gebruik makende van hetgeen reeds voorzien is binnen de uDig API.

In deze thesis is vooral het 2^e gebruik van belang.
De GIS markt wordt gedomineerd door een paar grote spelers. De systemen die worden aangeboden door deze spelers gebruiken traditioneel bestanden als eenheid van data. Ze kunnen echter niet overweg met de hoge *latency* en lage betrouwbaarheid van internetverkeer, iets wat steeds belangrijker is geworden de laatste jaren, met de opkomst van het internet. uDig kan dit wel: het ondersteunt zowel standaard databases en bestanden als web services als OpenGIS WFS (Web Feature Service) en OpenGIS WMS (Web



Figuur 5.1: Situering van uDig ten opzichte van bestaande systemen

Map Service). Het gebruikt een renderer die extreem tolerant is voor *latency*: het behandelt elke layer apart en tekent features zodra deze beschikbaar zijn. Het doel van uDig is om de functionele leegte in twee gemeenschappen te vullen: de Open Source geospatiale gemeenschap en de Geospatiale Open Standards gemeenschap. In de Open Source geospatiale gemeenschap vult uDig de rol die ArcView[1] in de merkgebonden softwarewereld speelt. In de Geospatiale Open Standards gemeenschap vervult uDig de rol van de „geïntegreerde client" die transparant en makkelijk met internet services en data kan omgaan. Kortom, uDig brengt genetwerkte spatiale data op een gebruiksvriendelijke manier naar de desktop.
In Figuur 5.1 wordt getoond waar uDig gesitueerd kan worden in de huidige GIS wereld.

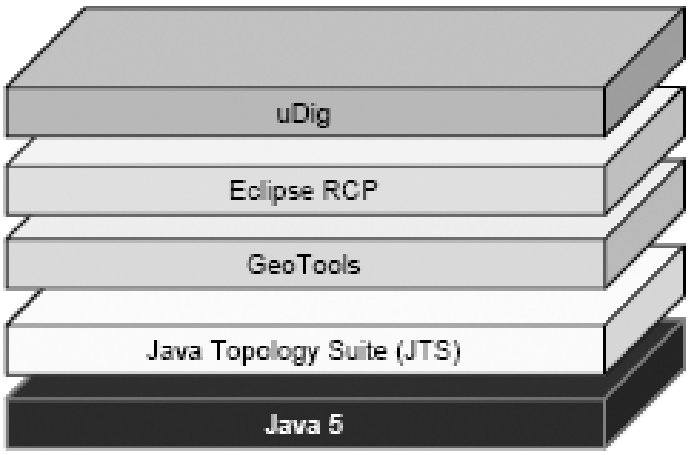
5.2 Structuur en uitbreidbaarheid

Eén van de belangrijkste eigenschappen van uDig is dat er zoveel mogelijk bestaande technologie in de ontwikkeling gebruikt worden. Zo wordt het *RCP* of „*Rich Client Platform*" van IBM in Eclipse ([18]) gebruikt. Het RCP is een verzamelnaam voor een minimale set van plugins die nodig zijn om een applicatie te bouwen. De componenten ervan kunnen gebruikt worden in vrijwel eender welke applicatie. Het is hierdoor dat uDig extreem modulair is opgebouwd.

[1] ArcView of ArcView GIS[6] is een desktop GIS software pakket ontwikkeld door ESRI. Het is een uitgebreide viewer voor spatiale data, met mogelijkheden om complexe analyses en databeheer te doen.

Verder wordt ook gebruik gemaakt van *GeoTools*, een Java software bibliotheek die een aantal structurele GIS taken verzorgt, zoals het inlezen van GIS bestanden, het connecteren met spatiale databases, het berekenen van projecties, het renderen van GIS objecten, ... uDig is gebaseerd op GeoTools, waardoor er tijdens de ontwikkeling geconcentreerd kan worden op de user interface.

Zoals in Figuur 5.2 te zien is, is uDig dus bovenop „Eclipse RCP" en „GeoTools" gebouwd. Structureel gezien bestaat uDig uit twee delen: GIS



Figuur 5.2: uDig architectuur

Platform en GIS Applicatie. Het **GIS Platform** is de kern van uDig in de zin dat het toegang geeft tot de spatiale informatie, zonder dat er iets visueels aan te pas komt. De hoofdcomponent hiervan is de *Catalog plugin*. Deze zorgt voor verschillende services, zoals het beheren van de spatiale data en het „ontdekken" van nieuwe resources. De Catalog plugin ondersteunt verschillende formaten „out of the box" en is uitbreidbaar voor eigen content. De API van deze plugin is ruwweg verdeeld in drie abstracties:

- **ICatalog**: catalogus van lokale en „remote" spatiale data.
- **IService**: stelt een externe service voor, zoals de Web Feature Service (WFS) of de shapefile.
- **IGeoResource**: direct beschikbare spatiale data zoals een Web Map Server (WMS) layer of de inhoud van een shapefile.

De IGeoResource is het meest nuttige onderdeel van de Catalog plugin. Het stelt bruikbare spatiale data voor, die getoond kan worden op het scherm.

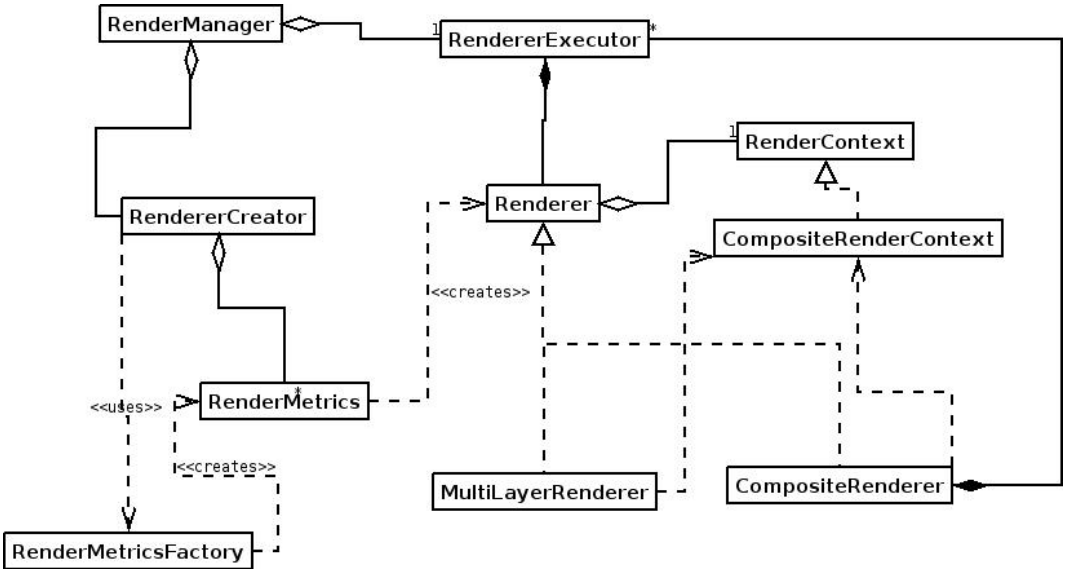
Voorbeelden hiervan zijn een GridCoverage uit een tabel of view uit een database, een FeatureCollection die beschikbaar is op een Web Feature Server (WFS), de inhoud van een shapefile of een Web Map Server (WMS) layer.
Het **GIS Platform** laat ook communicatie met andere onderdelen (zoals de User Interface) toe door middel van events en listeners, zoals drag & drop (met uDig zelf of met externe programma's) of het bijhouden van veranderingen aan de Catalog.
Het andere deel van uDig, de **GIS Applicatie**, verfijnt de ideeën van het GIS Platform in concepten als *projects*, *maps* en *layers*. De GIS Applicatie visualiseert deze concepten in de User Interface. In de volgende subsectie zal verder ingegaan worden op de GIS Applicatie.

5.3 GIS Applicatie

De basis van uDig kan gebruikt worden als simpele maar degelijke GIS applicatie. Vanuit dit startpunt kunnen er nieuwe GIS applicaties afgeleid worden. Dit kan op twee manieren gebeuren:

- uDig uitbreiden door naar behoeven plugins te schrijven.
- Een nieuwe applicatie schrijven die gebaseerd is op uDig.

Deze twee manieren spitsen zich beiden vooral toe op de GIS Applicatie, omdat het GIS Platform niets met de user interface te maken heeft. In deze tekst wordt de eerste manier van uitbreiden besproken.
De GIS Applicatie beheert de data in drie datastructuren: Projects, Maps en Layers. Een **Project** stelt de lokatie van data op een schijf voor. Een **Map** stelt de visualisatie van spatiale data voor, waarbij een ViewPoint de „area of interest" aanduidt. Een Map bestaat uit verschillende Layers, waarbij iedere **Layer** een laag van informatie voorstelt. Om te beslissen hoe een data set (een punt, lijn, polygoon, ...) gevisualiseerd wordt, maakt uDig gebruik van Styles. Een **Style** beschrijft een set van „painting rules" die dan door de **Renderer** gebruikt worden tijdens het visualiseren.
Het renderen van de data sets wordt beheerd door de **RenderManager**. De RenderManager gebruikt de **RendererCreator** om renderers te creëren. De RendererCreator gebruikt de **RenderMetricsFactory** die op zijn beurt de **RenderMetrics** creërt, die op zijn beurt dan weer de Renderer creërt. Een **Renderer** kan een **CompositeRenderer** zijn als het om één layer per renderer gaat en een **MultiRenderer** als het om meerdere layers per renderer gaat. De Renderers krijgen toegang tot de spatiale data door middel van een CompositeRenderContext. De **CompositeRenderContext** bevat een set



Figuur 5.3: UML diagram van de Renderer interface

van RenderContexts. De **RenderContext** communiceert met de Maps en de Layers en geeft de juiste GeoResource aan de Renderer. In het vakgebied der object-georiënteerd ontwerpen wordt dit ontwerppatroon de *façade* genoemd. Merk op dat ook het ontwerppatroon *factory* al gebruikt werd. Elke Renderer heeft ook zijn eigen **RendererExecutor**, die ervoor zorgt dat de renderer in zijn eigen thread werkt. In Figuur 5.3 staat een UML diagram van de Renderer interface.
Om de gebruiker te laten interageren met het data model, wordt er gebruik gemaakt van Commands en Tools. Deze twee klassen zullen nu in meer detail besproken worden.

5.3.1 Commands

Commands zijn de enige toegelaten manier om met het data model van uDig rechtstreeks aan te passen. Dit is verplicht opdat het mogelijk zou zijn dat de gebruiker acties ongedaan kan maken (undo) of opnieuw kan doen (redo). Commands worden in uDig gecreëerd door een specifieke **CommandFactory** en beheerd door de **CommandManager**.
Er zijn 6 soorten commands:

- **Basic Commands**: Basiscommando's.
  Voorbeeld: *Add Layer* of *Delete Layer*

- **Selection Commands**: Commando's die de selectie veranderen.
  Voorbeeld: *No Select* of *BBox Selection*

- **Draw Command**: Teken commando's die de MapDisplay veranderen. De klasse verschilt van de rest in de zin dat deze commando's een langere tijd kunnen blijven bestaan, bijvoorbeeld wanneer een nieuwe shape getekend wordt.
  Voorbeeld: *Translation* of *Draw Shape*

- **Edit Command**: Commando's die features veranderen.
  Voorbeeld: *Modify Feature* of *Delete Feature*

- **Navigation Command**: Navigatie commando's.
  Voorbeeld: *Pan* of *SetViewportCenter* of *ZoomCommand*

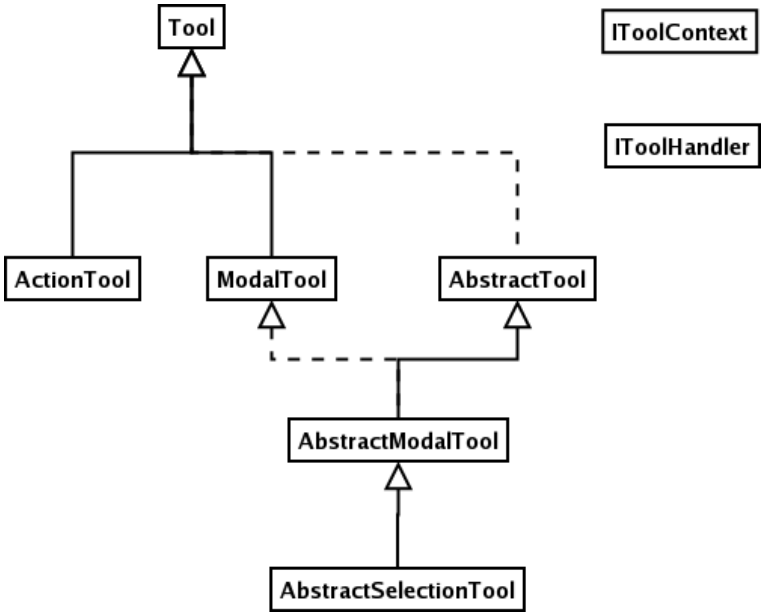
- **Composite Command**: Meerdere commando's samengevoegd tot één commando.

5.3.2 Tools

Naast Commands zijn er ook Tools. Tools hebben toegang tot uDig via de User Interface en kunnen opdrachten geven die het programma veranderen. Ze krijgen door uDig een ToolContext object toegewezen, waarmee ze toegang krijgen tot het model en Commands ernaar kunnen sturen die het veranderen.
Tools worden geplaatst in de toolbar en/of het menu en worden beheerd door de ToolManager. Er zijn 4 soorten tools:

- **Action Tool**: Dit soort tool voert één opdracht uit wanneer de tool geactiveerd wordt. Het heeft geen invloed op de muiscursor.
  Voorbeeld: *Zoom to Extent*
  Deze tool zoomt in of uit zodat de volledige kaart net in het venster past.

- **Modal Tool**: Een tool die aan -en uitgezet kan worden. In uDig kan er op elk moment maar maximaal één modal tool actief zijn. Het heeft geen run-methode, maar luistert naar de input van de muis. Het heeft een eigen cursor die gebruikt wordt wanneer de tool actief is.
  Voorbeeld: *Zoom*
  Deze tool zoomt in op een gebied van de kaart wanneer de gebruiker er klikt. De tool zal enkel werken wanneer de modal tool met als naam 'Zoom' geactiveerd is.



Figuur 5.4: Tool interfaces en abstracte klasses

- **Background Tool**: Een tool die altijd actief is op de achtergrond. Typische background tools zijn tools die de gebruiker constante feedback geven.
  Voorbeeld: *Cursor position*
  Deze tool geeft in een statusbar de wereldcoördinaat weer die overeenkomt met de positie van de muis op de kaart.

- **Edit Tool**: Deze tool is nieuw in uDig en nog in volle ontwikkeling. Edit Tools worden in meer detail besproken in de volgende paragraaf.

In Figuur 5.4 is een diagram gegeven die de tool interfaces en abstracte klasses weergeeft. Omdat de Edit tool nog in volle ontwikkeling is, zijn deze nog niet in het diagram opgenomen. De Action, Modal en Background tools implementeren respectievelijk de ActionTool, de ModalTool en de Tool interface.
Tools kunnen zelf gecreëerd worden door gebruik te maken van het uitbreidingspunt **net.refractions.udig.project.tool**. Uitbreidingspunten worden in meer detail besproken in Sectie 5.4.

5.3.3 Edit tools

De edit tool is nieuw in uDig en nog in volle ontwikkeling. Een mogelijke edit tool is bijvoorbeeld het verplaatsen of het verwijderen van een vertex.

Omdat een edit tool veel meer mogelijke toepassingen heeft en omdat deze tools vaak veel gemeenschappelijk hebben, wordt er een nieuw framework rond gebouwd.
Centraal in dit framework staan de klassen *AbstractEditTool*, *EditToolHandler*, *EditBlackboard*, *EventBehaviour* en *Activator*.

- **AbstractEditTool**: De centrale superklasse van alle edit tools. Deze doet bij de instantiatie weinig, behalve het configureren van *EditToolHandler*.

- **EditToolHandler**: Deze klasse bestaat uit *EventBehaviours*, *Activators* en *AcceptBehaviours*. Het vangt de muis -en toetsenbord events op die komen van *AbstractEditTool* en stuurt deze door naar de juiste *EventBehaviour*-klasse. Het heeft ook een locking mechanisme zodat een *EventBehaviour*-klasse andere *EventBehaviours* kan uitsluiten wanneer nodig. Zo zal bijvoorbeeld de MoveVertex *EventBehaviour* de SelectVertexBox *EventBehaviour* uitsluiten tijdens het verslepen van een vertex. Deze klasse verschaft ook nog toegang tot *MouseTracker*, de huidige geometrie/shape en de huidige edit state (in een enum *EditState*).

- **Activator**: Activators worden uitgevoerd wanneer de tool geactiveerd of gedeactiveerd wordt.

- **AcceptBehaviours**: Wordt gebruikt door *EditToolHandler* wanneer de tool gedeactiveerd wordt. Dit is typisch wanneer de gebruiker op de Enter-toets drukt.

- **EditBlackboard**: Deze klasse biedt verschillende methodes om efficiënt geometriën te bewerken. Wanneer een geometrie wordt toegevoegd aan *EditBlackboard*, dan wordt deze geometrie verdeeld in één of meerdere *EditGeom* objecten. Er kunnen dan vertices van deze objecten toegevoegd, verwijderd of aangepast worden. *EditBlackboard* zorgt ervoor dat de mappingen en transformaties juist worden uitgevoerd.

- **EventBehaviour**: Deze klasse wordt gebruikt wanneer het toetsenbord of de muis een event veroorzaakt. De klasse heeft een *isValid()*-functie die afhankelijk van de status in *EditToolHandler* bepaalt of een event valid is. Zoja, dan zal *EditToolHandler* de *getCommand()*-functie van *EventBehaviour* gebruiken.

5.4 Uitbreidingspunten

Een uitbreidingspunt of *extension point* is een klasse in uDig die uitgebreid kan worden om tot een eigen customisatie of plugin voor uDig te komen. Een customisatie wordt gedefinieerd in een plugin.xml bestand. Deze plugin.xml omschrijft de plugin volgens een vast XML Schema en verschaft Eclipse van de nodige informatie om de plugin te kunnen gebruiken.
Een voorbeeld van een plugin.xml voor een simpele „Hello World"-plugin ziet er als volgt uit:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
id="com.example.helloworld"
name="Helloworld Plug-in"
version="1.0.0"
provider="EXAMPLE"
class="com.example.helloworld.HelloworldPlugin">

  <runtime>
    <library name="helloworld.jar">
     <export name="*"/>
  </library>
  </runtime>

  <requires>
<import plugin="org.eclipse.ui"/>
<import plugin="org.eclipse.core.runtime"/>
<import plugin="org.eclipse.core.runtime.compatibility"/>
  </requires>

  <extension
      point="org.eclipse.ui.views">
    <category
       name="Hello Category"
       id="com.example.helloworld">
    </category>
    <view
      name="Hello View"
      icon="icons/sample.gif"
      category="com.example.helloworld"
```

```
      class="com.example.helloworld.HelloWorldView"
      id="com.example.helloworld.HelloWorldView">
    </view>
  </extension>
</plugin>
```

Het uitbreidingspunt van deze plugin is „org.eclipse.ui.views". Het plugin.xml bestand bevat de lokatie van het .jar bestand van de plugin (in het runtime-gedeelte), de benodigde libraries (in het requires-gedeelte), het uitbreidingspunt en verdere informatie over de implementatie (in het extension-gedeelte). In de volgende Sectie wordt het schrijven van een eigen plugin of customisatie toegelicht, hetgeen ook een plugin.xml bestand zal genereren.
Er zijn verschillende uitbreidingspunten beschikbaar in uDig:

- Tool extension point: Voor action, modal of background tools
- LayerOp extension: Voor operaties op één of meerdere layers
- Renderer extension: Voor het renderen van eigen datatypes
- Style extension: Wordt gebruikt bij door een Renderer; Nodig bij het renderen van eigen geometriën
- Decorator extension: Voor het weergeven van een map of een pagina in de GUI
- OutputDevice extension: Voor het gebruik van een eigen uitvoerapparaat
- Template extension: Voor het maken van een layout bij het printen

5.4.1 uDig - Schrijven van eigen plugin

In Bijlage A.2 wordt het schrijven van een nieuwe tool in uDig beschreven. Een meer gedetailleerde en concrete uitleg, met code, kan gevonden worden op het Internet([21]).
Het uitbreidingspunt in deze tutorial is „net.refractions.udig.project.ui.tool" en de Plugin.xml die Eclipse tijdens het maken van de plugin heeft gegenereerd ziet er als volgt uit:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
   <extension
```

```
         id="net.refractions.udig.tutorial.distancetool"
         name="Distance Tool Voorbeeld"
         point="net.refractions.udig.project.ui.tool">
      <modalTool
            categoryId="net.refractions.udig.tool.category.info"
            class="net.refractions.udig.tutorial.
            distanceTool.DistanceTool"
            icon="icons/etool16/measure_mode.gif"
            id="net.refractions.udig.tutorial.distanceTool"
            name="Distance"
            onToolbar="true"
            tooltip="Meet de afstand tussen 2 punten">
         <cursor
               hotspotX="10"
               hotspotY="10"
               image="icons/pointers/measure_source.gif"/>
      </modalTool>
   </extension>
</plugin>
```

5.5 Conclusie

In deze Sectie is een inleiding gegeven tot uDig, met een overzicht van de uitbreidingsmogelijkheden en een voorbeeld van hoe een uitbreidingsplugin geïmplementeerd kan worden. uDig, wat staat voor **U**ser Friendly **D**esktop **I**nternet **G**IS, is een Open Source spatiale data editor/viewer die in volle ontwikkeling is en waarbij de nadruk ligt op OpenGIS standaarden en Internet GIS (Web Map Server en Web Feature Server). Het biedt een sterk uitbreidbaar framework voor GIS applicatie ontwikkeling. Het uDig platform is gebaseerd op het Eclipse Rich Client Platform (RCP), wat voor een sterke modulariteit zorgt. Ook maakt het gebruik van de Java library GeoTools, voor verscheidene GIS functionaliteiten.
Het is mogelijk om een eigen GIS applicatie te maken met uDig op twee manieren: door het schrijven van een nieuwe applicatie die gebaseerd is op uDig, gebruik makende van de RCP structuur of door het schrijven van eigen plugins voor de gewenste functionaliteiten. Het biedt talloze (afleidbare) klassen en uitbreidingspunten, waarmee GIS applicatie ontwikkelaars bijna alles kunnen doen wat ze zouden willen doen. Het enige nadeel verbonden aan uDig is dat men gebonden is aan Eclipse.

Hoofdstuk 6

Conclusie

De laatste jaren is de interesse in zowel GIS als Open Source software sterk gestegen. GIS of Geografische Informatiesystemen worden steeds belangrijker in ons dagelijks leven, mede dankzij de opkomst van het internet. Het doel van deze thesis was om een evaluatie te maken van Open Source Software voor het ontwikkelen en ondersteunen van Desktop GIS-toepassingen. De thesis is gemaakt in opdracht van GIM, een bedrijf in Leuven dat zich specialiseert in dienstverlening rond Geografische Informatiesystemen.

In deze thesis is een vergelijking gemaakt tussen commerciële en Open Source software, op het gebied van spatiale databanken. Aan de ene kant staan Oracle Spatial/Locator en ESRI's ArcSDE, twee commerciële databanken. Aan de andere kant staan PostGIS en MySQL, twee Open Source databanken. Oracle is marktleider in de database-wereld. Oracle Spatial/Locator heeft een goede, volledige implementatie van de Simple Features Specification en een uitstekende ondersteuning voor transacties en locking. Als indexstructuur is er keuze tussen de R-tree en de Quadtree. Oracle heeft echter een groot nadeel: het is erg duur. ESRI's ArcSDE behoort tot een andere klasse, die van de middle-ware. Het werkt als een gateway tussen een databank en clienttoepassingen. Het volgt de Simple Features Specification volledig en ondersteunt transacties en locking. De indexstructuur in ArcSDE hangt af van het onderliggende systeem.

Aan de Open Source kant staan PostGIS en MySQL. PostGIS is de spatiale uitbreiding van PostgreSQL. MySQL is de meest bekende Open Source databank, met sinds versie 4.1 een spatiale extensie. Alhoewel dat ze volledig gratis zijn, bieden PostGIS en MySQL een waardig alternatief voor de commerciële pakketten. Vooral PostGIS biedt alles wat een betalende database biedt. Het heeft een goede, volledige implementatie van de Simple Featu-

res Specification en een goede ondersteuning voor transacties en locking. PostGIS ondersteunt ook als enige de indexstructuur GiST. MySQL's ondersteuning van de Simple Features Specification is onvolledig. Het standaard tabeltype (MyISAM) heeft geen volwaardige ondersteuning voor transacties en locking, maar er zijn alternatieve tabeltypes die dit wel hebben. Als indexstructuur gebruikt MySQL de R-tree. Het nadeel aan PostGIS en MySQL is dat de documentatie en klantenondersteuning minder goed is, omdat het gratis is, in tegenstelling tot commerciële databanken. Dit wordt echter grotendeels goedgemaakt door de grote bekendheid van de twee op het internet, waardoor er veel informatie te vinden is.

In de performantie-benchmark met PostGIS en MySQL neemt geen van de twee de bovenhand. Ze hebben beiden hun sterke en zwakke punten. Meestal presteert PostGIS beter, maar bij simpele geometrische functies (zoals oppervlakte of lengte/omtrek) en het combineren van spatiale rijen (spatial joins) is MySQL beter. Ook is het belangrijk op te merken dat er veel parameters aangepast kunnen worden bij beide systemen, waardoor de performantie nog kan variëren.

In het praktisch gedeelte van deze thesis is een studie en implementatie gemaakt met uDig. uDig staat voor **U**ser Friendly **D**esktop **I**nternet **GI**S en is een jonge Open Source GIS applicatie en platform voor de ontwikkeling van eigen GIS applicaties. Het heeft veel uitbreidingspunten voor het toevoegen van nieuwe functionaliteiten. Het kan gebruikt worden als basis wanneer plugins worden geschreven voor enkele simpele functionaliteiten die gewenst zijn of wanneer een volledig nieuwe applicatie gemaakt zal worden. uDig maakt gebruikt van het Rich Client Platform van Eclipse, waardoor men gebonden is aan Eclipse.

Bijlage A

Appendix

A.1 Geometrische types van de Simple Features Specification

In dit deel van de bijlage worden de verschillende geometrische types en de bijbehorende functies en methodes van de „Simple Features Specification for SQL" (zie Sectie 2.2) gegeven.

A.1.1 Geometry

Omdat Geometry de root is van de hierarchie, worden alle methodes overgeërfd door de andere types.
Dit zijn de basismethodes van Geometry :

- Dimension(): Geeft de inherente dimensie van de geometrie terug. Dit moet kleiner of gelijk aan de dimensie van het coördinaatsysteem zijn.
- GeometryType(): Geeft de naam van het geïnstantieerde subtype van Geometry terug.
- SRID(): Geeft de *Spatial Reference System ID* voor deze geometrie terug.
- Envelope(): Geeft de kleinste bounding box van de geometrie terug. Dit is een polygoon bepaald door de hoekpunten : ((MINX, MINY), (MAXX, MINY), (MAXX, MAXY), (MINX, MAXY), (MINX, MINY)).
- AsText(): Geeft de Well-Known Text representatie van de geometrie terug.

- AsBinary(): Geeft de Well-Known Binary representatie van de geometrie terug.
- IsEmpty(): Geeft TRUE terug als de geometrie de lege puntenset (ø) van het coördinaatsysteem voorstelt.
- IsSimple(): Geeft TRUE terug als de geometrie geen afwijkende punten bevat (bijvoorbeeld door zichzelf te snijden).
- Boundary(): Geeft de sluiting van de gecombineerde grens in de vorm van een geometrie terug.

Dit zijn de methodes voor het testen van spatiale relaties tussen geometrische objecten:

- Equals(andereGeometrie): Geeft TRUE terug als deze geometrie spatiaal gelijk is aan *andereGeometrie*.
- Disjoint(andereGeometrie): Geeft TRUE terug als deze geometrie spatiaal disjunct is met *andereGeometrie*.
- Intersects(andereGeometrie): Geeft TRUE terug als deze geometrie spatiaal snijdt met *andereGeometrie*.
- Touches(andereGeometrie): Geeft TRUE terug als deze geometrie spatiaal raakt met *andereGeometrie*.
- Crosses(andereGeometrie): Geeft TRUE terug als deze geometrie spatiaal kruist met *andereGeometrie*.
- Within(andereGeometrie): Geeft TRUE terug als deze geometrie spatiaal volledig omvat is door andereGeometrie.
- Contains(andereGeometrie): Geeft TRUE terug als deze geometrie *andereGeometrie* spatiaal bevat.
- Overlaps(andereGeometrie): Geeft TRUE terug als deze geometrie *andereGeometrie* spatiaal overlapt.
- Relate(andereGeometrie,intersectiePatroonMatrix): Geeft TRUE terug als deze geometrie spatiaal gerelateerd is met *andereGeometrie*, door de binnenkant, de grens en de buitenkant van de 2 geometriën voor intersecties te testen.

Verder zijn er nog enkele methoden die een spatiale analyse doen. Alle berekeningen gebeuren in het *Spatial Reference System* van de betreffende geometrie.

- Distance(andereGeometrie): Geeft een double terug die de kortste afstand is tussen twee punten van de twee geometriën.
- Buffer(afstand): Geeft een geometrie terug waarvan alle punten binnen een afstand kleiner of gelijk aan *afstand* van de betreffende geometrie liggen.
- ConvexHull(): Geeft een geometrie terug die het convexe omhulsel van de betreffende geometrie is.
- Intersection(andereGeometrie): Geeft een puntenset-geometrie die de doorsnede van de betreffende geometrie met *andereGeometrie* is.
- Union(andereGeometrie): Geeft een puntenset-geometrie die de unie van de betreffende geometrie met *andereGeometrie* is.
- Difference(andereGeometrie): Geeft een puntenset-geometrie die het verschil van de betreffende geometrie met *andereGeometrie* is.
- SymDifference(andereGeometrie): Geeft een puntenset-geometrie die het symmetrisch verschil van de betreffende geometrie met *andereGeometrie* is.

A.1.2 GeometryCollection

Een GeometryCollection is een geometrie die een verzameling is van één of meer geometriën. Het *Spatial Reference System* van de GeometryCollection is hetzelfde voor alle elementen.
Deze abstracte klasse heeft twee methoden:

- NumGeometries(): Geeft het aantal geometriën in de verzameling terug.
- GeometryN(N): Geeft de N^{de} geometrie van de verzameling terug.

A.1.3 Point

Een Point of punt is een 0-dimensionale geometrie en stelt één enkele locatie voor in de coördinaatruimte. Een punt heeft een x -en y-coördinaatwaarde.
Point heeft twee methoden:

- X(): Geeft de x-coördinaat van het betreffende Point.
- Y(): Geeft de y-coördinaat van het betreffende Point.

In de praktijk kan een Point bijvoorbeeld gebruikt worden om een stad (op een grote kaart) of een bushalte voor te stellen.

A.1.4 MultiPoint

Een MultiPoint is een 0-dimensionale GeometryCollection. De elementen van een MultiPoint zijn Points. Er is geen ordering of samenhang tussen de punten. In de praktijk kan een MultiPoint bijvoorbeeld gebruikt worden om een eilandengroep (op een wereldkaart) of alle bushaltes (op een stadskaart) voor te stellen.

A.1.5 Curve

Een Curve is een 1-dimensionaal geometrisch object die een curve of een sequentie van punten voorstelt. De klasse Curve is abstract, de afgeleide ervan bepaalt welke vorm van interpolatie er wordt gebruikt. De specificatie bepaalt maar één afgeleide subklasse van Curve : LineString. LineString gebruikt lineaire interpolatie tussen punten.
Curve heeft vijf methoden:

- Length(): Geeft de lengte van de Curve terug.
- StartPoint(): Geeft het startpunt van de Curve terug.
- EndPoint(): Geeft het eindpunt van de Curve terug.
- IsClosed(): Geeft TRUE terug als de Curve gesloten is. Dit wil zeggen dat het startpunt gelijk is aan het eindpunt.
- IsRing(): Geeft TRUE terug als de Curve gesloten EN simpel is. Een Curve is simpel als het nooit meer dan éénmaal door een punt gaat.

A.1.6 LineString, Line, LinearRing

Een LineString is een Curve met lineaire interpolatie tussen punten. Een Line is een LineString met exact twee punten. Een LinearRing is een LineString die gesloten en simpel is. LineString heeft drie methoden:

- NumPoints(): Geeft het aantal punten van de betreffende LineString terug.

- PointN(N): Geeft het N^{de} Point van de betreffende LineString terug.

In de praktijk kan een LineString bijvoorbeeld gebruikt worden om een rivier (op een grote kaart) of een straat (op een stadskaart) voor te stellen.

A.1.7 MultiCurve

Een MultiCurve is een abstracte (niet-instantieerbare) 1-dimensionale GeometryCollection. De elementen ervan zijn Curves. MultiCurve heeft twee methoden:

- IsClosed(): Geeft TRUE terug als elke Curve van de betreffende MultiCurve gesloten is.
- Length(): Geeft de som van de lengtes van elke Curve van de betreffende MultiCurve terug.

A.1.8 MultiLineString

Een MultiLineString is een MultiCurve met LineStrings als elementen. In de praktijk kan een MultiLineString bijvoorbeeld gebruikt worden om een rivierennet of een snelwegennetwerk voor te stellen.

A.1.9 Surface

Een Surface is een 2-dimensionaal geometrisch object en stelt een al dan niet planair oppervlak voor. Bij een simpele surface horen één „buiten"-grens en 0 of meerdere „binnen"-grenzen. De klasse is abstract en heeft drie methoden:

- Area(): Geeft de oppervlakte van de betreffende Surface terug.
- Centroid(): Geeft het wiskundige zwaartepunt van de betreffende Surface terug. Dit moet niet op het oppervlak liggen.
- PointOnSurface(): Geeft een Point terug dat gegarandeerd op de Surface ligt.

A.1.10 Polygon

Een Polygon is instantieerbare subklasse van Surface en stelt een planair oppervlak voor. De grens van een Polygon bestaat een verzameling van LinearRings die de buitenste en binnenste ringen voorstellen.
Polygon heeft drie methoden:

- ExteriorRing: Geeft een LineString terug die de buitenste ring rond de betreffende Polygon voorstelt.
- NumInteriorRing(): Geeft het aantal binnenringen terug van de betreffende Polygon terug.
- InteriorRingN(N): Geeft de N^{de} binnenste ring van de betreffende Polygon terug als LineString.

In de praktijk kan een Polygon bijvoorbeeld gebruikt worden om een provincie of een bos voor te stellen.

A.1.11 MultiSurface

Een MultiSurface is een 2-dimensionaal geometrische verzameling met Surfaces als elementen. Het binnenste van twee Surfaces van de verzameling mag niet snijden. De grens van twee Surfaces mag snijden in ten hoogste een eindig aantal punten.
De klasse is abstract en heeft drie methoden:

- Area(): Geeft de oppervlakte van de betreffende MultiSurface terug.
- Centroid(): Geeft het wiskundige zwaartepunt van de betreffende MultiSurface terug. Dit moet niet op het oppervlak liggen.
- PointOnSurface(): Geeft een Point terug dat gegarandeerd op de MultiSurface ligt.

A.1.12 MultiPolygon

Een MultiPolygon is instantieerbare subklasse van MultiSurface, met Polygons als elementen. De klasse heeft dezelfde drie methoden als MultiSurface. In de praktijk kan een MultiPolygon bijvoorbeeld gebruikt worden om een groep van meren en waterplassen op een kaart voor te stellen.

A.2 uDig - Schrijven van een plugin

In dit deel van de bijlage wordt uitgelegd hoe een eigen plugin in uDig geschreven kan worden.

A.2.1 uDig - Installatie

Dit zijn de stappen die doorlopen moeten worden om uDig te installeren. Er wordt ervan uitgegaan dat Eclipse en Java JRE nog niet geïnstalleerd zijn. Is dit wel het geval, dan kan onmiddelijk naar stap 3 gegaan worden.

1. Installeer en start Eclipse
    - Download de Eclipse SDK van http://www.eclipse.org
    - Download http://udig.refractions.net/downloads/eclipse_jre.zip
    - Download http://udig.refractions.net/downloads/extras.zip
    - Unzip de Eclipse SDK naar de Java directory:
      `C:\java\eclipse` wordt gemaakt
    - Unzip eclipse_jre.zip naar de Eclipse directory:
      `C:\java\eclipse\jre` wordt gemaakt
    - Unzip extras.zip naar de Java directory
    - Start Eclipse door eclipse.exe te runnen
2. Kies een workspace, bijvoorbeeld `C:\java\workspace`
3. Breng een aantal veranderingen aan in de instellingen, nodig om uDig te kunnen gebruiken
    - Open Window - Preferences
    - Verander Java - Compiler: Compiler compliance level: 5.0
    - Verander Java - Installed JREs: Als `C:\java\eclipse\jre` er niet staat, op 'Add...' klikken en JRE aanmaken met dit pad
4. Installeer uDig
    - Download de recentste uDig SDK van http://udig.refractions.net
    - Unzip dit bestand naar `C:\java\sdk`
    - Ga terug naar Eclipse
    - Open de instellingen: Window - Preferences

- Selecteer Plugin Development - Target Platform
- Verander het 'Target Platform' in `C:\java\sdk\udig`
- Klik 'Ok'

5. Start uDig
    - Selecteer Run¿Run...
    - Selecteer Eclipse Application
    - Klik „New"
    - Zorg dat „Run as Product" geselecteerd is
    - Zorg dat „net.refractions.udig.product" in het „Run as Product" veld staat
    - Klik „Run"

Nu wordt uDig gestart en kan je de functionaliteiten gebruiken alsof je de stand-alone versie (zonder Eclipse) had gebruikt.

A.2.2 uDig - Schrijven van een eigen tool plugin

In deze subsectie wordt uitgelegd hoe een simpele tool plugin geschreven kan worden in uDig. Het gaat in dit geval over een „Modal Tool".

1. Open het *Plug-in Development* perspective: **Window - Open Perspective - Other... - Plug-in Development**
2. Selecteer **File - New - Project...**, selecteer *Plug-in Project* en klik op „Next"
3. Vul bij Project Name de ID van de plugin in en klik op „Next". Bijvoorbeeld: „net.refractions.udig.myplugins.distancetool".
4. Vul bij *Plug-in Name* een naam voor de Plug-in in. Bijvoorbeeld: „Distance Tool Plugin". De naam en ID moeten enkele stappen verder nog eens ingevuld worden.
5. Configureer de nieuwe plugin
    - Open het *Plug-in Development* perspective: **Window - Open Perspective - Other... - Plug-in Development**
    - Ga in de *Package Explorer* op zoek naar het plugin manifest, te vinden als `META-INF/MANIFEST.MF`. Dubbelklik hierop om de editor te openen.

- Klik op de *Dependencies* tab, te vinden onderaan de editor.
- Klik op „Add...” in de kolom van 'Required plugins' en voeg de volgende plugin toe: „net.refractions.udig.project.ui”
- Sla je werk op, zodat de veranderingen worden gepropageerd in het project

6. Defineer een nieuwe extensie

   - Open de *Extensions* tab
   - Klik „Add...”
   - Selecteer „net.refractions.udig.project.ui.tool” uit de lijst
   - Klik „Finish”
   - Vul de naam en ID van je tool (zoals enkele stappen geleden gekozen) in bij *Extension details*

     Bijvoorbeeld:
     ID: „net.refractions.udig.myplugins.distancetool”
     Name: „Distance Tool Plugin”

7. Creëer een nieuwe tool

   - Rechterklik op de nieuwe extensie („net.refractions.udig.project.ui.tool”) en selecteer **New - ModalTool**
   - Vervang de gegevens in het *ID* veld door je ID. Bijvoorbeeld: „net.refractions.udig.myplugins.distancetool”
   - Vul een tooltip in in het *tooltip* veld. Bijvoorbeeld: „Meet de afstand tussen 2 punten”
   - Vul je class-naam in in het *class* veld. Bijvoorbeeld: „net.refractions.udig.myplugins.distancetool.DistanceTool”
   - Bij „Icon” kan een icoon gekozen worden, dat op voorhand geïmporteerd moet worden in het project.
   - Vul een naam in in het veld *name*. Bijvoorbeeld: „Distance”
   - Stel onToolbar in op true
   - Vul „net.refractions.udig.tool.category.info” in in het *categoryId* veld

8. Implementatie van de tool

   - Selecteer je extensie in de *Extensions editor*. Bijvoorbeeld: „net.refractions.udig.myplugins.distancetool (modalTool)”

- Klik op het woord „class” bij *Extension Element Details*
- In het dialoog dat verschijnt zal je zien dat de klasse een kindklasse is van „net.refractions.udig.project.ui.tool”. Vink „Generate comments” aan en klik op „Finish”.
- Rechterklik in de editor en selecteer **Source - Override/Implement Methods**
- Vink *onMousePressed(MapMouseEvent)* en *onMouseReleased(MapMouseEvent)* aan onder de „SimpleTool” knoop.
  Klik op „Ok”.
- Implementeer *onMousePressed(MapMouseEvent)*[1]
- Implementeer *onMouseReleased(MapMouseEvent)*
- Implementeer *displayError()*
- Implementeer *displayOnStatusBar(double)*
- Op dit moment zijn er veel rode kruisjes, door ontbrekende klassen. Door Ctrl+Shift+0 te duwen worden deze klassen geïmporteerd. Als er gevraagd wordt om tussen 2 Geotools klassen te kiezen, moet „org.geotools.geometry.jts.JTS” gekozen worden. Als nu het project opgeslagen wordt, verdwijnen alle rode kruisjes.

Wanneer nu uDig wordt uitgevoerd, dan zal de nieuwe Distance Tool beschikbaar zijn. Selecteer deze tool, sleep van één punt naar een ander op de kaart en linksonder in de statusbar zal de afstand verschijnen.

[1] voor volledige code, zie [21]

A.3 Benchmark logboek

Tijdens het uitvoeren van de performantie benchmark, zijn er enkele problemen geweest met het inlezen van de data in MySQL. Deze problemen en eventuele oplossingen zullen nu besproken worden.
De data die gebruikt wordt zijn een aantal shapefiles die ieder een bepaalde verzameling geometriën voorstelden. Hieruit werden de grootste bestanden geselecteerd die polygonen bevatten.

Importeren in PostgreSQL/PostGIS
Bij de standaard installatie van PostgreSQL zit een executable ('`shp2pgsql.exe`') die shapefiles naar een PostgreSQL compatibel .sql querybestand kan converteren. Deze operatie verliep snel en zonder problemen. Het importeren van dit .sql bestand in PostgreSQL verliep daarna ook vlot.

Importeren in MySQL
Er zit geen tool bij MySQL die overweg kan met shapefiles. De bestanden die de PostgreSQL tool genereert zijn niet compatibel. Op het internet[2] staat een downloadbare executable ('`shp2mysql.exe`') die volgens de auteur in staat is om een shapefile naar een MySQL compatibel .sql querybestand te converteren. Het converteren verliep zonder problemen, maar bij het uitvoeren van het .sql bestand liep het fout. Het werkte voor ruwweg 90% van de geometriën, maar bij de andere 10% gaf MySQL de fout :

```
Error Code: 1416 - Cannot get geometry object from data
you send to the GEOMETRY field.
```

De statements waarbij dit gebeurde, waren niet groter dan de andere (correcte) statements en waren syntactisch correct. Ik heb toen een post gemaakt op het forum van MySQL[3]. Dit is door velen gelezen, maar er is nooit antwoord op gekomen. Ik heb ook een bugreport ingediend bij MySQL, maar ook zonder antwoord. Het is me dus niet gelukt om te achterhalen of de gebruikte convertietool bugs bevatte of dat de oorzaak een bug in MySQL was.
Na lang tevergeefs zoeken heb ik beslist om een kleinere maar gelijkaardige dataset te gebruiken. Ook deze werd zonder problemen ingelezen in PostgreSQL. Het converteren naar een MySQL compatibel .sql querybestand met de executable '`shp2mysql.exe`' verliep zonder problemen, maar het uitvoeren van dit .sql bestand niet. Bij bepaalde statements gaf MySQL de foutmelding:

[2] zie http://kartoweb.itc.nl/RIMapper/
[3] zie http://forums.mysql.com/read.php?23,74163,74163#msg-74163

`MySQL server has gone away`

Deze fout kwam altijd voor bij dezelfde statements. Deze statements waren ook groter dan alle andere, correcte statements. De handleiding van MySQL[4] zegt dat er veel oorzaken kunnen zijn voor het wegvallen van de MySQL server met deze foutmelding. Een van de mogelijkheden volgens de tekst zegt dat de queries te lang zijn. De MySQL server denkt dan dat er aan de clientzijde iets foutliep en verbreekt de verbinding. In dit geval wordt voorgesteld[5] om de `max_allowed_packet`-variabele in het optiebestand ('`mysqld.ini`') te veranderen. Standaard staat deze ingesteld op *1M*. Een statement in MySQL mag dus met andere woorden niet groter zijn dan 1MB. De statements waarbij het fout liep voor mij waren ook allemaal groter dan 1MB. Na het verhogen van `max_allowed_packet` naar *16M* kon het volledige .sql bestand wel uitgevoerd worden en was de data correct ingelezen in MySQL.

[4]zie http://dev.mysql.com/doc/refman/5.0/en/gone-away.html
[5]zie http://dev.mysql.com/doc/refman/5.0/en/packet-too-large.html

Bibliografie

[1] *Proceedings of the Fifth International Conference on Data Engineering, February 6-10, 1989, Los Angeles, California, USA*. IEEE Computer Society, 1989.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In Garcia-Molina and Jagadish [8], pages 322–331.

[3] Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors. *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.* Morgan Kaufmann, 1995.

[4] ESRI. Shapefile technical description. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

[5] ESRI. Esri's arcsde. http://www.esri.com/software/arcgis/arcsde/, 2006.

[6] ESRI. Esri's arcview gis. http://www.esri.com/software/arcview/arcview3x.html, 2006.

[7] The Eclipse Foundation. Eclipse website. http://www.eclipse.org/, 2006.

[8] Hector Garcia-Molina and H. V. Jagadish, editors. *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990.* ACM Press, 1990.

[9] Diane Greene. An implementation and performance analysis of spatial data access methods. In *ICDE* [1], pages 606–615.

[10] Antonin Guttman. R-Trees: A dynamic index structure for spatial searching. In Yormark [30], pages 47–57.

[11] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In Dayal et al. [3], pages 562–573.

[12] Y. Edmund Lien, editor. *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, April 29 - May 1, 1981*. ACM Press, 1981.

[13] MySQL. Mysql. http://www.mysql.com/, 2006.

[14] MySQL. Mysql to promote new open source db engines from its partners and dev community. http://tinyurl.com/fkyek, 2006.

[15] Oracle. Oracle spatial & oracle locator. http://www.oracle.com/technology/products/spatial/index.html, 2006.

[16] Oracle. Oracle spatial documentation. http://www.oracle.com/technology/documentation/spatial.html, 2006.

[17] OSOSS. Open-sourcesoftware. 2006.

[18] Rich Client Platform. Eclipse website. http://www.eclipse.org/rcp/, 2006.

[19] PostgreSQL. Postgresql. http://www.postgresql.org/, 2006.

[20] Refractions Research. Postgis. http://postgis.refractions.net/, 2006.

[21] Refractions Research. uDig website - developing a simple distance tool plugin. http://udig.refractions.net/confluence/display/DEV/3+Plugin+Tutorial, 2006.

[22] Refractions Research. User-friendly desktop internet gis (uDig) website. http://udig.refractions.net/, 2006.

[23] Philippe Rigaux, Michel Scholl, and Agnès Voisard. *Introduction to Spatial Databases: Applications to GIS*. Morgan Kaufmann, 2000.

[24] John T. Robinson. The K-D-B-Tree: A search structure for large multidimensional dynamic indexes. In Lien [12], pages 10–18.

[25] Steven M. Rubin. *Computer Aids for VLSI Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[26] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In Stocker et al. [27], pages 507–518.

[27] Peter M. Stocker, William Kent, and Peter Hammersley, editors. *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*. Morgan Kaufmann, 1987.

[28] Wikipedia. Geografisch informatiesysteem. http://nl.wikipedia.org/wiki/GIS, 2006.

[29] Wikipedia. Open-sourcesoftware. http://nl.wikipedia.org/wiki/Open source, 2006.

[30] Beatrice Yormark, editor. *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*. ACM Press, 1984.

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen en uw akkoord te verlenen.

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Evaluatie van Open Source Software voor het ontwikkelen van Desktop GIS-toepassingen
Richting: **Master in de informatica** Jaar: **2006**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Deze toekenning van het auteursrecht aan de Universiteit Hasselt houdt in dat ik/wij als auteur de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij kan reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

U bevestigt dat de eindverhandeling uw origineel werk is, en dat u het recht heeft om de rechten te verlenen die in deze overeenkomst worden beschreven. U verklaart tevens dat de eindverhandeling, naar uw weten, het auteursrecht van anderen niet overtreedt.

U verklaart tevens dat u voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen hebt verkregen zodat u deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal u als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze licentie

Ik ga akkoord,

Benny GIESBERS

Datum: