# De Ondersteuning van Spatial Data in Commerciële Database Systemen.

Thesis voorgedragen tot het behalen van de graad van master in de Informatica, afstudeervariant Informatica-Databases.

Erwin Thoelen

**Promotor**: Prof. dr. Bart Kuijpers
**Begeleiders**: dr. Sofie Haesevoets
De heer Bart Moelans

**Academiejaar**: 2005-2006

# Voorwoord

Deze thesis wordt voorgedragen tot het behalen van de graad van master in de informatica, afstudeervariant Informatica-Databases. Graag zou ik enkele mensen willen bedanken zonder wiens hulp dit niet gelukt was.

Mijn promotor prof. dr. Bart Kuijpers en begeleiders dr. Sofie Haesevoets en de heer Bart Moelans voor hun suggesties en tips zowel bij de literatuurstudie als bij het uitvoeren van de testen.

De personen die mij voorzien hebben van de nodige software en die mij geholpen hebben met problemen hiermee. In het bijzonder Dirk Leinders, Ruben Coolen, Maarten Steegmans en mijn vader Luc.

Mijn familie voor het nalezen en verbeteren van deze tekst.

Mijn vrienden en medestudenten, die op tijd en stond voor ontspanning zorgden én mij aanmoedigden.

Mijn ouders voor hun steun en omdat ze mij de mogelijkheid gegeven hebben om deze studie te volgen.

Erwin Thoelen

# Inhoudsopgave

# Hoofdstuk 1

# Overzicht

Enkele tientallen jaren geleden was men voornamelijk op kaarten aangewezen voor het produceren en voorstellen van *spatial data* of *ruimtelijke gegevens*. Het manipuleren van deze informatie was een manueel en niet-interactief proces. Sindsdien is het dankzij de evolutie van de technologie mogelijk om spatial data te digitaliseren en samen met de stijgende vraag naar interactieve manipulatie en analyse van deze gegevens zijn er *dedicated* toepassingen ontstaan, namelijk *Geographic Information Systems* (GIS's).

Een GIS is meer dan een toepassing waarmee alleen kaarten geproduceerd kunnen worden. Het moet ook toelaten om spatial data, samen met zijn ondersteunende alfanumerieke data, te kunnen opslaan, terug op te vragen en te kunnen analyseren. In hoofdstuk 2 wordt uitgelegd wat een GIS juist inhoudt, wat ermee gedaan kan worden en waaruit een GIS opgebouwd is.

Door de voortdurende toename van ruimtelijke gegevens wordt het steeds belangrijker voor een GIS om grote *databanken* van complexe informatie te beheren. Voor klassieke software wordt hier meestal een relationeel Database Management System (DBMS) voor gebruikt. Maar deze zijn vooral gericht op bedrijfstoepassingen die met grote, maar simpele, hoeveelheden alfanumerieke gegevens werken. Hoofdstuk 3 beschouwt een aantal mogelijkheden om GIS en relationele DBMS met elkaar te integreren en de problemen die daarbij de kop opsteken.

In hoofdstuk 4 wordt zo één voorstel verder uitgewerkt. Het voorstel om ruimtelijke gegevens en databanken te combineren is door een relationele DBMS uit te breiden zodat hierin willekeurige objecten opgeslagen kunnen worden. Voor zo een database kan dan een object type gedefinieerd worden waarin spatial data kan worden opgeslagen, een *Spatial Database*.

Vanuit het standpunt van een GIS kan dit misschien al voldoende zijn, de analyse van de gegevens wordt, nadat deze uit de database opgehaald zijn, toch door het GIS uitgevoerd. Vanuit het standpunt van een DBMS ligt de nadruk niet alleen op het opslaan van gegevens maar ook op het efficiënt opvragen en de verwerking hiervan. Het opvragen van spatial data wordt gerealiseerd met behulp van indexen op het spatial datatype. Omdat de conventionele indexen voorzien zijn voor alfanumerieke gegevens en niet voor spatial data moest een nieuw type index bedacht worden: de *Spatial index*. Hierbij komt nog dat operaties die sterk afhankelijk zijn van indexen aangepast moeten worden aan deze nieuwe index. Voor het analyseren worden ook nieuwe operaties, specifiek voor ruimtelijke gegevens, toegevoegd aan het spatial datatype. Deze analysefuncties laten toe om eigenschappen van een spatial object op te vragen, om de topologische relatie tussen twee objecten te bepalen en om nieuwe ruimtelijke objecten te creëren aan de hand van bestaande objecten.

Na dit theoretisch deel over GIS en Spatial databases volgt een praktisch deel waarin drie commeciële (database) systemen aan bod komen. In hoofdstuk 5 worden *DB2 Spatial Extender* van IBM, *Oracle Spatial* van Oracle en *ArcGis* van ESRI besproken. DB2 Spatial Extender en Oracle Spatial zijn spatial database systemen terwijl ArcGIS een GIS is. Deze systemen zijn gekozen omdat IBM, Oracle en ESRI tot de grotere spelers op de spatial markt horen en al een behoorlijke periode spatial software ontwikkelen. Hierdoor zijn ze al geëvolueerd tot een punt waar ze geen kinderziektes meer hebben en waar ze al andere en/of betere theoretische algoritmen en structuren hebben kunnen implementeren in hun toepassingen.

In hoofdstuk 6 wordt met behulp van deze drie systemen een case study uitgevoerd. Hierbij worden verschillende queries uitgevoerd op ieder systeem. Deze queries zijn in drie categorieën in te delen: alfanumerieke queries, spatial queries en interactieve queries.

Tenslotte wordt in hoofdstuk 7 een vergelijking gegeven van de drie geteste systemen. Hierbij wordt enerzijds gekeken naar de mogelijkheden van het systeem, zoals het datatype, index en functies, en anderzijds naar het gebruiksgemak en voorkennis die nodig is om met het systeem te werken.

Ter informatie worden hier nog enkele systemen vermeld, zonder verdere bespreking, die spatial data ondersteunen: *DB2 Geodetic Extender* van IBM, *Informix Spatial DataBlade Module* van IBM, *Informix Geodetic DataBlade*

*Module* van IBM, *PostgreSQL* (een open source, object-relationele DBMS) en *SpatialWare* van MapInfo (een module voor Informix en MS SQL Server die de verwerking van spatial data mogelijk maakt in relationele DBMS).

# Hoofdstuk 2

# Wat is een Geographic Information System?

## 2.1 Definitie

[1, 3]. In de beginjaren van GIS was het eenvoudig te zeggen waarvoor dit acroniem stond: een Geographic Information System. Maar de laatste jaren wordt dit steeds moeilijker omdat steeds meer organisaties er mee bezig zijn en andere aspecten ervan onderzoeken en ontwikkelen. De *G* in GIS staat nu voor *Geographic* of *Geospatial*, beide termen betekenen ongeveer hetzelfde namelijk de leer van de geografie. De *I* in GIS staat nog altijd voor *Information*. Geographic of Geospatial Information is dus "informatie over locaties op het aardoppervlak". Of nog, "kennis over **wat** zich **waar** bevindt, op een bepaald **tijdstip**".

Ook de *S* in GIS heeft vandaag meerdere betekenissen, één voor elke onderzoekstak ervan:

- Systems: software/technologie om spatial data te verwerven en beheren
- Science: theorie/wetenschap achter de technologie
- Studies: sociale, wettelijke en ethische aspecten van de twee bovenstaande gevallen
- Services: (web) diensten voor routeplanning en/of reizigers

Deze tekst gaat hoofdzakelijk over Geographic Information *Systems* en in mindere mate over de Geographic Information *Science*.

Naast GIS zijn er ook nog andere technologieën die Geographic Information gebruiken. Twee hiervan zijn Global Positioning Systems (GPS) en Remote Sensing (RS). GPS is een systeem van satellieten in een baan rond de aarde die met grote nauwkeurigheid de positie van een zender op het aardoppervlak kunnen bepalen. RS kan ook met satellieten gebeuren maar evengoed met vliegtuigen, hierbij wordt informatie, zoals hoogte of vegetatie, over het aardoppervlak verzameld. GPS en RS voorzien in de input data voor GIS en GIS voorziet in de opslag en manipulatie van deze data.

Ook definiëren welke functionaliteit een GIS moet bieden gaat niet zonder problemen. Zo hebben verschillende organisaties verschillende definities gemaakt, bijvoorbeeld:

Federal InterAgency Coordinating Committee on Digital Cartography (1988) (nu het Federal Geographic Data Committee) [20]: *System of computer hardware, software and procedures designed to support the capture, management, manipulation, analysis, modeling and display of spatially referenced data for solving complex planning and management problems.*

Wetenschappelijke Commissie GIS Vlaanderen (1993) [21]: *Een GIS is een coherent communicatie- en beheers-systeem voor het verwerken en de interpretatie van gegevens over de ruimtelijke aspecten van de wereld.*

Hendriks en Ottens (1997) [6]: *Een geografisch informatiesysteem is een computersysteem, dat hulpmiddelen biedt om aan elkaar gekoppelde ruimtelijke en niet-ruimtelijke gegevens te structureren, op te slaan, te bewerken, te beheren, op te vragen, te analyseren en weer te geven, zodanig dat die gegevens nuttige informatie opleveren voor het beantwoorden van een gegeven beleids- of onderzoeksvraag.*

De definitie die in deze verhandeling gebruikt wordt, is de volgende van het Decreet GIS-Vlaanderen (17 juli 2000, B.S. 2 sept. 2000) [21]: *Informatiesysteem voor het opbouwen, beheren, verwerken, presenteren, integreren en communiceren van geografische informatie. Waarbij met geografische informatie: "alle ruimtelijk gerefereerde informatie" bedoeld wordt.*

Concreet betekenen deze definities allemaal ongeveer hetzelfde, namelijk dat GIS voor een gebruiker een **hulpmiddel** is dat hem toelaat de verwerking van ruimtelijke informatie te vereenvoudigen en/of te optimaliseren.

Aan de hand van bovenstaande definities kunnen we de hoofdtaken van een

GIS onderscheiden:

- het integreren, ordenen en beheren van grote hoeveelheden gegevens met een ruimtelijke component
- het structureren van deze gegevens tot basisinformatie en het beschikbaar stellen ervan
- het analyseren en verder verwerken van deze informatie
- het ([carto-]grafisch)[1] presenteren van de informatie

## 2.2 Componenten van een GIS

In de volgende paragraaf worden de onderdelen van een GIS toegelicht. Iedere component is noodzakelijk opdat een GIS zijn taken kan uitvoeren. Deze componenten kunnen gegroepeerd worden als volgt: technische componenten [3], spatial referentiesysteem (hoofdstuk 8 van [7]), gegevens ([3], [7] en hoofdstuk 2 van [17]) en modellen (hoofdstuk 2 van [25]).

### 2.2.1 Technische componenten

In deze paragraaf gaan we dieper in op de technische onderdelen van een GIS. Dit zijn de hardware, software, humanware en de orgware.

**Hardware** of Computeruitrusting: het basisplatform is geëvolueerd van *mainframes* in de jaren 70, via *mini computers* in de jaren 80, tot *micro computers* vanaf de jaren 90 tot nu. De huidige *personal computers* (PC's) volstaan in de meeste gevallen om een GIS applicatie te draaien. Voor data-opslag en communicatie volstaat ook de conventionele randapparatuur, desnoods kan een aanvullende en gedeelde opslagcapaciteit voorzien worden op servers. Extra randapparatuur zoals een digitaliseertablet, scanner of plotter zijn wel nodig om de ruimtelijke gegevens in te voeren of te tonen.

**Software** of Programmatuur: deze is net zoals de hardware geëvolueerd: het is begonnen met *DataBase Management Systemen* (DBMS) van de eerste generatie samen met *Computer Aided Design* (CAD) van de eerste generatie, matrixkaarten en beeldverwerking. De eerste twee systemen evolueerden dan via een tweede generatie naar een *vector GIS* en de laatste twee naar een

[1] Cartografisch: geprint op een landkaart. Grafisch: weergegeven op een monitor.

*raster GIS*. Daarna werden raster GIS en vector GIS gecombineerd tot een *GIS* van de tweede generatie. Daarna werden er nog netwerkmogelijkheden en kennis- en expertsystemen aan toegevoegd samen met nog vele andere technologieën die de laatste jaren ontwikkeld zijn om zo aan een *GIS* van *the Next generation* te komen.

**Humanware** of programmeurs: adequaat opgeleid personeel is nodig omdat GIS de laatste jaren dusdanig geëvolueerd is dat het binnen heel wat disciplines door toepassingsdeskundigen, zonder fundamentele kennis over geografische informatie, wordt gebruikt. Dit neemt niet weg dat het uitwerken van een conceptueel en logisch datamodel, het topologisch opbouwen van een coherente gegevensset, het correct refereren in verschillende systemen, de keuze van de gepaste ruimtelijke algoritmes en het correct visualiseren specifieke kennis en ervaring vergt. Dit behoort tot de opleiding en de taken van de geografen en geo-informatici.

**Orgware** of organisatie: hierin kan men verschillende dimensies onderscheiden. Er is enerzijds de eigen organisatie, zowel de structuur als het personeel, maar ook de technologische uitrusting en integratie. Anderzijds is er de externe organisatie, met andere woorden hoe wordt de beschikbaarheid van de ruimtelijke gegevens georganiseerd regionaal, nationaal en zelfs internationaal. Dit omvat naast technische en wetenschappelijke aspecten ook verschillende administratieve, legale, commerciële en politieke punten.

### 2.2.2 Spatial referentiesysteem

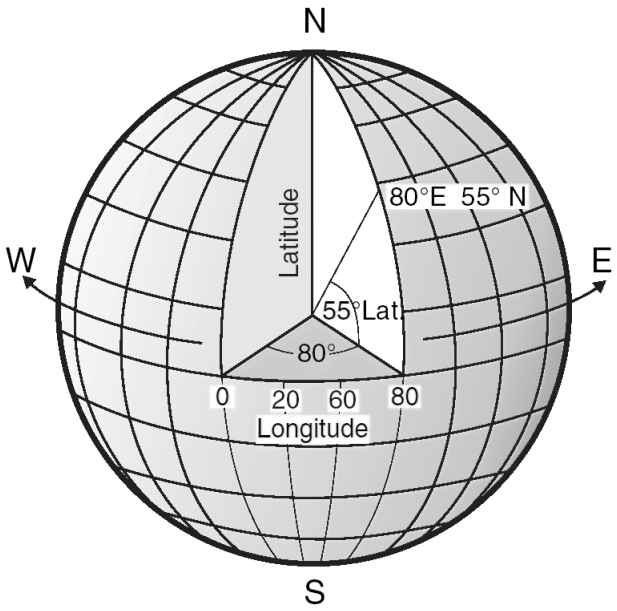
Ruimtelijke gegevens worden meestal voorgesteld door middel van coördinaten. Omdat coördinaten, bijvoorbeeld van landgrenzen, alleen maar een zinvolle betekenis hebben als ze correct geplaatst kunnen worden in de reële wereld, moet er een referentie zijn die de coördinaten hun positie in de reële wereld geeft. Die referentie gebeurt door een spatial referentiesysteem, dit bestaat uit een coördinaten systeem, een ID, het maximale bereik van de coördinaten en nog andere elementen voor interne representatie. Het belangrijkste element hiervan is het **coördinaten systeem**. Dit is een raamwerk om de relatieve positie van 'dingen' in een gegeven gebied te bepalen, bijvoorbeeld om een object of een gebeurtenis, zoals een botsing, te plaatsten in een gebied op het aardoppervlak of zelfs op de aarde in zijn geheel. De meest voorkomende systemen zijn: het *geographic coordinate system* dat een drie-dimensionaal bolvormig oppervlak gebruikt om posities op aarde voor te stellen en het *projected coordinate system* dat een platte, twee-dimensionale voorstelling van de aarde gebruikt.

#### 2.2.2.1 Geographic coordinate system

Een *geographic coordinate system* of geografisch coördinaten systeem is een referentiesysteem dat een drie-dimensionaal bolvormig oppervlak gebruikt om locaties op aarde vast te leggen. Naar iedere locatie op aarde kan verwezen worden met een punt, dit punt bestaat uit een *longitude* en een *latitude* component. Deze kunnen uitgedrukt worden in de volgende eenheden:

- graden (latitude [-90°, 90°], longitude [-180°, 180°])
- gradialen (latitude [-100gon[2], 100gon], longitude [-200gon, 200gon])
- radialen (latitude [$-\pi/2$, $\pi/2$], longitude [$-\pi$, $\pi$])

We leggen dit referentiesysteem uit aan de hand van een voorbeeld. Figuur 2.1 toont een geografisch coördinaten systeem waar een locatie voorgesteld wordt door de coördinaten 80° oosterlengte (longitude) en 55° noorderbreedte (latitude). (Ter informatie, deze locatie bevindt zich midden in Novosibirsk, Rusland.)

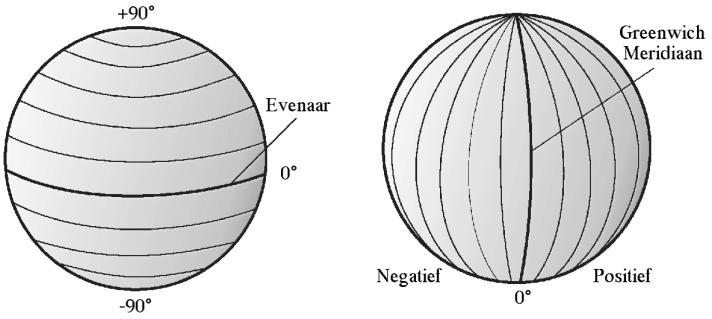



Figuur 2.1: Geographic coordinate system (bron [7]).

Elke lijn die van oost naar west loopt, heeft een constante latitude en wordt een *breedtecirkel* genoemd. Deze lijnen zijn equidistant en parallel aan elkaar

[2]Eén gon is $1/400^{ste}$ van een cirkel, 100gon komt overeen met 90°.

en vormen concentrische cirkels rond de aarde. De grootste hiervan is de *evenaar* met een latitude van 0° deze verdeelt de aarde in twee. Locaties ten noorden van de evenaar hebben een latitude met een bereik van 0° tot +90° terwijl locaties ten zuiden ervan een latitude hebben van 0° tot -90°. Zie figuur 2.2(a).




(a) Latitude (b) Longitude

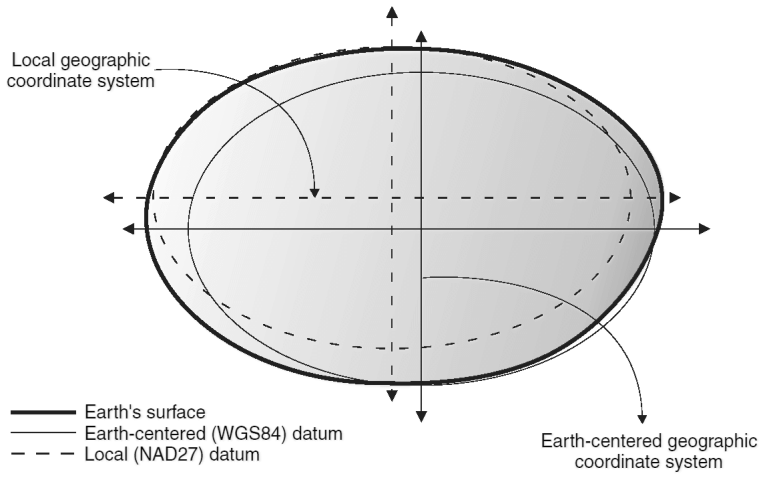
Figuur 2.2: Componenten geographic coordinate system (bron [7]).

Elke lijn die van noord naar zuid loopt, heeft een constante longitude en wordt een *meridiaan* genoemd. Deze lijnen vormen cirkels van dezelfde grootte rond de aarde en snijden elkaar op de polen. De *nulmeridiaan* heeft een longitude van 0° en bepaalt zo de oorsprong voor de andere meridianen. De meest gebruikte meridiaan is die door Greenwich, Engeland. Er kunnen echter ook andere meridianen gebruikt worden om de oorsprong aan te geven, bijvoorbeeld de meridiaan door Parijs, Rome of Brussel. Locaties ten oosten van de nulmeridiaan tot aan zijn antipodische meridiaan (het vervolg van de nulmeridiaan aan de andere kant van de aarde) hebben een longitude met een bereik van 0° tot +180°, terwijl locaties ten westen ervan een longitude hebben van 0° tot -180°. Zie figuur 2.2(b).

De meridianen en breedtecirkels vormen een grid dat de hele aarde bedekt. De oorsprong van dit raster, positie (0, 0), is waar de evenaar en de nulmeridiaan elkaar snijden. De evenaar is de enige plaats op het raster waar de lineaire afstand overeenkomstig met 1° latitude ongeveer gelijk is aan de afstand van 1° longitude. Omdat de meridianen samenkomen aan de polen, is de afstand tussen twee meridianen verschillend aan iedere breedtecirkel. Aan de evenaar komt 1° latitude overeen met een afstand van ongeveer 111,321km terwijl dit aan de polen slechts een punt is zonder lengte. Omdat er geen uniforme lengte is voor 1° op een meridiaan en breedtecirkel, kan de lengte tussen twee willekeurige punten niet accuraat gemeten worden.

Een coördinaten systeem kan gedefinieerd worden door een bol ofwel door een afgeplatte bol. Omdat de aarde niet prefect bolvormig is, representeert een afgeplatte bol de aarde beter. De geometrie die de afgeplatte bol voorstelt is een ellipsoïde, deze verkrijgt men door een ellips te roteren rond één van haar assen. Maar dit levert in vele gevallen nog altijd niet de gewenste accuraatheid op, daarom kan de ellipsoïde in zijn geheel nog verplaatst worden ten opzichte van de aarde zodat deze **voor een lokaal deel** van de aarde samenvalt. Deze relatieve verplaatsing wordt gerealiseerd door een *datum*.

Een **datum** is een set van waardes die de positie van de ellipsoïde definieert relatief aan het middelpunt van de aarde. Zo definieert het de oorsprong van het raster. Er bestaan datums voor globaal gebruik die een goede gemiddelde accuraatheid hebben voor de hele wereld. Maar er bestaan ook lokale datums die alleen maar voor een bepaald gebied accuraat zijn, dit heeft als gevolg dat wanneer een verkeerde lokale datum gebruikt wordt de coördinaten niet meer overeenkomen met posities in de reële wereld.




Figuur 2.3: Datum (bron [7]).

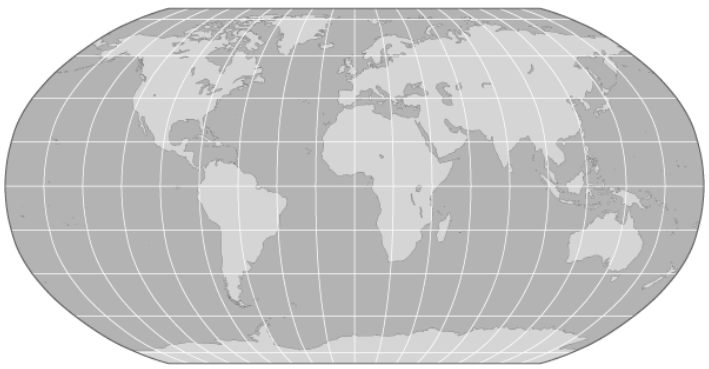
Figuur 2.3 toont een voorbeeld hoe een datum de ellipsoïde lokaal laat samenvallen met het aardoppervlak. Merk op dat wanneer de datum veranderd wordt, de coördinaten van een locatie op de aarde ook mee veranderen.

#### 2.2.2.2 Projected coordinate system

Een geprojecteerd coördinaten systeem is een platte, twee-dimensionale voorstelling van de aarde. Het is gebaseerd op een (bijna) bolvormig geografisch coördinaten systeem, maar het gebruikt lineaire eenheidsmaten. Dit heeft als gevolg dat berekeningen van afstand en oppervlakte gemakkelijk gedaan kunnen worden in dezelfde eenheden.

Latitude en longitude coördinaten worden geconverteerd naar $x$ en $y$ coördinaten op de platte projectie. De $X$ as valt meestal samen met de evenaar, terwijl de $Y$ as samenvalt met de nulmeridiaan. Hierbij is het snijpunt van de $X$ en $Y$ as de oorsprong en zijn de waardes boven de $X$ as en rechts van de $Y$ as positief. De waardes die er onder en links van liggen zijn negatief. Zowel de lijnen parallel aan de $X$ as als aan de $Y$ as zijn equidistant.

Om te converteren van een drie-dimensionaal geografish coördinaten systeem naar een twee-dimensionaal plat geprojecteerd coördinaten systeem worden er wiskundige transformaties gebruikt. Deze transformaties, ook wel map projecties genoemd, worden meestal ingedeeld volgens het gebruikte projectie-oppervlak, zoals kegelvormig, cilindrisch of vlak. Afhankelijk van de gebruikte projectie, zullen sommige ruimtelijke eigenschappen verstoord worden. Meestal probeert een projectie de vervorming van één of meerdere eigenschappen te beperken. Dit houdt in dat de andere eigenschappen, zoals afstand, oppervlakte, vorm, richting of een combinatie hiervan, niet accuraat kunnen zijn. Net zoals bij de datum het geval is, zijn er projecties die ofwel lokaal ofwel globaal een minimale verstoring van de eigenschappen proberen te bekomen. Waarschijnlijk het meest bekende voorbeeld van zo een globale projectie is de *Robinson* projectie (zie figuur 2.4).



Figuur 2.4: Voorbeeld van de Robinson projectie (bron [26]).

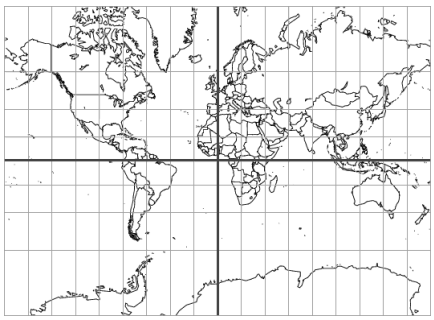
Andere veelgebruikte map projecties zijn:

*Equal area projections*: Deze projecties behouden de oppervlakte, maar verstoren de vorm, hoek en schaal van de objecten. Een voorbeeld hiervan is de *Albers Equal Area* [27] projectie.

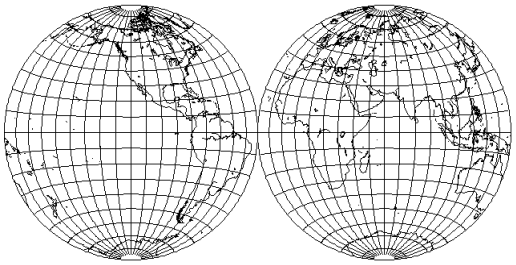
*Conformal projections*: Deze projecties bewaren de lokale vorm voor kleine gebieden evenals alle hoeken, de oppervlakte daarentegen wordt vervormd. De *Mercator* [27] (zie figuur 2.5(a)) en de *Lambert Conformal Conic* [27] projecties zijn hier voorbeelden van.

*Equidistant projections*: Hier worden de afstanden tussen punten bewaard door het behouden van de schaal in een bepaalde data set. Buiten deze set zal de schaal meer en meer vervormd geraken. Voorbeelden hiervan zijn de *Sinusoidal* [27] en de *Equidistant Conic* [27] projectie.

*True-direction or azimuthal projections*: Deze laatste groep projecties behoudt de richting van één punt naar alle andere punten. Ze geven de richting of azimut van alle punten correct ten opzichte van het middelpunt. Deze projecties kunnen ook gecombineerd worden met *equal area*, *conformal* of *equidistant* projecties. De *Lambert Equal Area Azimuthal* [27] (zie figuur 2.5(b)) projectie en de *Azimuthal Equidistant* [27] projectie zijn hier voorbeelden van.



(a) Mercator



(b) Lambert Equal Area Azimuthal

Figuur 2.5: Voorbeelden van map projecties (bron [27]).

### 2.2.3 Gegevens

De kern van ieder GIS bestaat uit twee aan elkaar gekoppelde beheersystemen voor data: één voor *ruimtelijke gegevens* en één voor *niet-ruimtelijke gegevens*, ook wel attributen genoemd. Rond deze kern kunnen een aantal systemen gebouwd zijn om de gegevens te bewerken of te analyseren, zoals: een datauitwisselingssysteem, een digitaliseersysteem voor ruimtelijke (cartografische) gegevens, systemen voor analyse, een beeldverwerkingsysteem en een weergavesysteem (ook cartografisch).

De rest van deze paragraaf gaat over ruimtelijke en niet-ruimtelijke gegevens: waar ze vandaan komen, wat ze voorstellen en hoe ze voor te stellen. De analyse-systemen komen in paragraaf 2.3 aan bod. In paragraaf 2.4 is het de beurt aan de datauitwisselings-, digitaliseer- en presentatie-systemen.

Niet-ruimtelijke gegevens (attributen) zijn in de meeste gevallen gewoon alfanumerieke data: records over werknemers, producten, adressen, belastingen, enzovoorts. Deze gegevens zijn één-dimensionaal, met andere woorden ze kunnen geordend worden op één enkele waarde. Het aanbrengen van deze data kan handmatig, bijvoorbeeld door de gegevens van een werknemer in te vullen of automatisch, bijvoorbeeld door per product metingen te registreren. De attributen worden meestal in een relationele database opgeslagen omdat dit hierin efficiënt kan gebeuren. Ook verdere analyse en querying van één-dimensionale gegevens wordt in relationele database systemen optimaal ondersteund.

Ruimtelijke gegevens kunnen niet zomaar in relationele databases opgeslagen worden, dit komt omdat deze data uit meerdere dimensies bestaan en de relationele databases alleen maar simpele types voorzien voor één-dimensionale gegevens. De $x$ en $y$ coördinaat van ruimtelijke gegevens kunnen per tuppel natuurlijk wel in aparte kolommen opgeslagen worden, maar dit brengt vele nadelen met zich mee. Zie paragraaf 3.2 op pagina 40 voor meer uitleg hierover. In een GIS is het aantal dimensies typisch gelijk aan twee, wegens het geografisch aspect van de gegevens (in het bijzonder bij cartografische data). Soms kan ook een derde of vierde dimensie nodig zijn om de hoogte/diepte van een object op te slaan en/of een tijdstip of een ander meetbare waarde bij te houden.

Ruimtelijke gegevens stellen informatie voor over een (*geografische*) *feature*. Een feature kan om het even wat zijn waarvan de locatie kan bepaald worden in de reële wereld of waarvan men de locatie zou kunnen voorstellen.

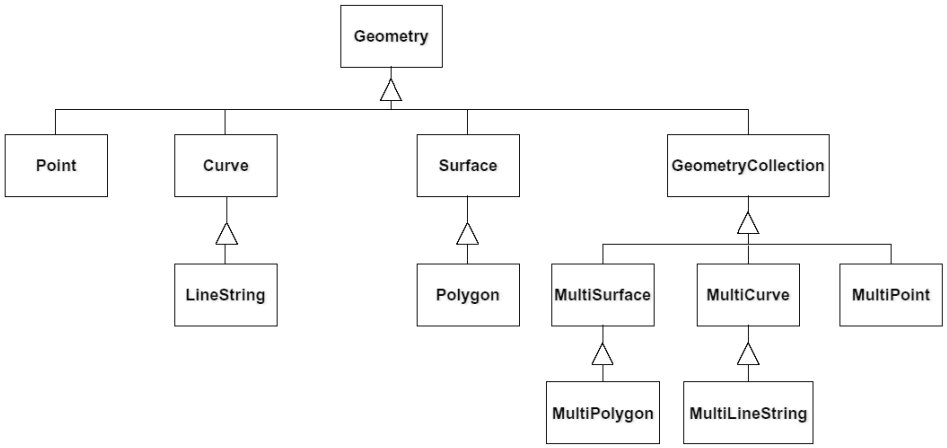
Voorbeelden hiervan zijn:

- een object/concrete entiteit: een stad, een rivier, een bos, enzovoorts
- een ruimte: een veiligheidszone rond een radioactieve plaats, een afzetmarkt van een winkel, enzovoorts
- een gebeurtenis op een bepaalde locatie: een auto-ongeval, een transactie in een bepaalde winkel, enzovoorts

De informatie over features die wordt opgeslagen zijn alle details over de locatie ervan. Informatie over andere dan locatiegebonden feiten van features wordt bijgehouden in het systeem voor niet-ruimtelijke gegevens om zo het plaatje compleet te maken.

Het simpelste *ruimtelijk data element* bestaat uit één enkel coördinatenpaar dat de locatie van één enkele geografische positie definieert. Een iets complexer ruimtelijk data element bestaat uit verschillende coördinaten die samen een lineair pad definiëren, zoals bijvoorbeeld een straat of een rivier. Een derde soort ruimtelijk data element bestaat uit coördinaten die de grens van een gebied definiëren, de grens van een land is hier een voorbeeld van.

Naar deze en andere ruimtelijke data elementen kan collectief verwezen worden met de term *Geometry*. De geometrieën die door het OpenGIS Consortium gedefinieerd zijn in [17] kunnen hiërarchisch voorgesteld worden, zoals in figuur 2.6.




Figuur 2.6: SQL Geometry Type Hiërarchie (bron [17]).

Het basistype *Geometry* heeft subtypes voor *Point*, *Curve*, *Surface* en *GeometryCollection*. Een GeometryCollection is een verzameling van mogelijk heterogene Geometrieën. *MultiPoint*, *MultiCurve* en *MultiSurface* zijn specifieke subtypes van GeometryCollection die gebruikt worden om homogene verzamelingen van Points, Curves en Surfaces te beheren. De types Point en MultiPoint zijn nul-dimensionaal. De types Curve en MultiCurve samen met zijn subklassen zijn één-dimensionaal. En de types Surface en MultiSurface samen met zijn subklassen zijn twee-dimensionaal.

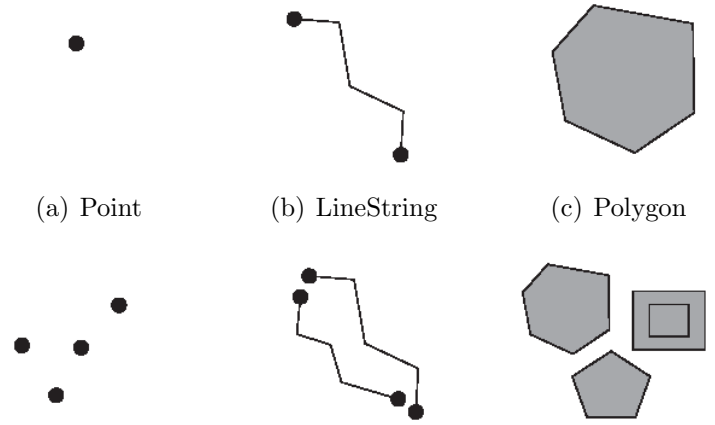
Het OpenGIS Consortium definieert in [17] ook dat een implementatie hiervan nieuwe types mag introduceren, zolang de subtype relaties maar behouden blijven. Bijvoorbeeld, als een type $A$ de rechtstreekse ouder is van een type $B$ mag er bijvoorbeeld een nieuw type $C$ tussen $A$ en $B$ voorkomen, zolang $B$ maar een subtype van $A$ blijft. Een andere resrictie die opgelegd wordt, is dat de types Geometry, Curve, Surface, MultiCurve en MultiSurface niet-instantieerbaar mogen zijn, er mogen dus geen constructors voor deze types gedefinieerd worden.

De overige types mogen wel geïnstantieerd worden en kunnen ingedeeld worden in twee categorieën: de *basis geometrieën* en de *homogene collecties*. De homogene collecties bevatten naast verzamelingen van basis geometrieën ook nog enkele extra eigenschappen, het duidelijkste voorbeeld hiervan is het aantal basis geometrieën in de verzamelingen.

*De basis geometrieën zijn*:
**Points**: Eén enkel punt. Punten stellen discrete features voor die liggen op een snijpunt van de oost-west en de noord-zuid coördinaat lijnen. Een punt kan bijvoorbeeld een stad voorstellen.
**LineStrings**: Een lijn tussen twee punten. Deze lijn moet niet recht zijn, maar is meestal wel een serie van rechte lijnstukken. Een lijn stelt een lineaire feature voor, zoals bijvoorbeeld een straat of kanaal.
**Polygons**: Een veelhoek of de oppervlakte van een veelhoek. Veelhoeken stellen veelzijdige features voor, zoals een gemeente of een bos.

*De homogene collecties zijn*:
**Multipoints**: Een verzameling van punten. Multipoints stellen features voor die uit meerdere punten bestaan, zoals een eilandengroep.
**Multilinestrings**: Een verzameling van 'Linestrings'. Ze stellen features voor die uit meerdere Linestrings bestaan, zoals een netwerk van rivieren.
**Multipolygons**: Een verzameling van veelhoeken. Ze stellen features voor die uit meerdere veelhoeken bestaan, zoals een land dat uit meerder delen

bestaat.

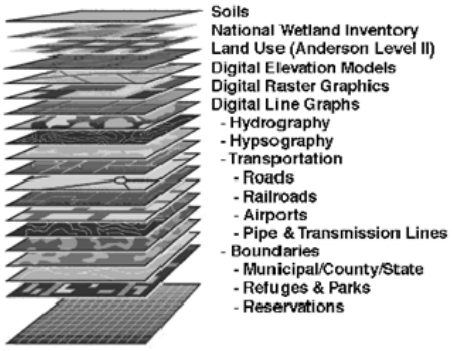


(a) Point (b) LineString (c) Polygon

(d) MultiPoint (e) MultiLineString (f) MultiPolygon

Figuur 2.7: Voorbeelden van geometrieën (bron [7]).

Al deze geometrieën worden natuurlijk niet allemaal samen opgeslagen (bijvoorbeeld in één bestand), maar zijn georganiseerd in verschillende *layers*, ook wel *themes* of *coverages* genoemd (zie figuur 2.8). In een layer zitten de gemeenschappelijke features/geometrieën **per thema**, zo kunnen er dus verschillende layers zijn voor wegen en rivieren. De manier waarop de geometrieën in een layer worden voorgesteld, hangt af van het gebruikte model (zie paragraaf 2.2.4).




Figuur 2.8: Voorbeelden van layers (bron [1]).

#### 2.2.3.1 Eigenschappen van geometrieën

Iedere geometry bezit een aantal eigenschappen [7]. Een implementatie van een geometry kan hier al dan niet rekening mee houden. Van een polygon kan bijvoorbeeld geëist worden dat het een gesloten linestring is. Of het kan zijn dat een implementatie alleen maar simpele geometrieën aankan. Hieronder worden deze en andere eigenschappen van geometrieën toegelicht:

*Type*: Het type van de geometry. (Multi)Point, (Multi)LineString, (Multi)Polygon of GeometryCollection.

*Coördinaten*: Iedere geometry bevat minimum één $X$ coördinaat en één $Y$ coördinaat, tenzij het de lege geometry is. Verder kunnen er ook nog één of meerdere $Z$ of $M$ coördinaten zijn. Een $Z$ coördinaat wordt meestal gebruikt voor de hoogte boven zeeniveau aan te geven en een $M$ coördinaat voor programmaspecifieke informatie, zoals de afstand tot de dichtstbijzijnde telefoonpaal op een autostrade.

*Inwendige, grens, uitwendige*: De positie van iedere geometry in de ruimte is gedefinieerd door hun inwendige, grens en uitwendige. Het uitwendige is de ruimte die niet ingenomen is door de geometry, het inwendige de ruimte die wel ingenomen wordt en de grens vormt de interface daartussen.

*Simpel of niet-simpel*: Een punt of lege geometry is altijd simpel, een multipoint is simpel als geen twee punten samenvallen, een (multi)linestring of een (multi)surface is simpel als hij zichzelf niet snijdt. In alle andere gevallen is de geometry niet-simpel.

*Gesloten*: Een curve is gesloten als zijn start- en eindpunt samenvallen. In het geval van een multicurve moet elke curve gesloten zijn.

*Convex*: Een polygon is convex als, voor iedere twee punten $a$ en $b$ in de polygon, het lijnstuk $ab$ volledig in de polygon ligt. In het geval van een convexe multipolygon moet iedere polygon convex zijn.

*Empty*: Een geometry is leeg als het geen coördinaten heeft. Het inwendige, grens en uitwendige zijn dan niet gedefinieerd (*null*) en in het geval van een (multi)polygon is zijn oppervlakte gelijk aan 0.

*Minimum Bounding Rectangle/Box (MBR/MBB)*: Dit is de kleinste rechthoek rond de geometry die de geometry volledig omvat. Deze rechthoek

wordt gevormd door de minimum en maximum coördinaten ervan en de zijden zijn evenwijdig aan assen van het coördinaten systeem.

*Dimensie*: Een geometry kan -1, 0, 1 of 2 dimensies hebben. Hierbij wordt '-1' gebruikt om aan te duiden dat het over de *lege geometry* gaat. Bij 0 dimensies heeft de geometry een lengte en oppervlakte gelijk aan 0, bij 1 dimensie is de lengte groter dan 0 maar is de oppervlakte nog altijd gelijk aan 0 en bij 2 dimensies is de oppervlakte van de geometry groter dan 0.

*Spatial referentiesysteem*: Dit is de verwijzing naar het gebruikte spatial referentiesysteem om de geometry voor te stellen.

### 2.2.4 Modellen

Het modelleren van geografische en topologische informatie in GIS heeft wiskundige grondvesten die een grondige aanpak vergen. Deze tekst beperkt zich echter tot enkele basisconcepten die volstaan voor de hier besproken modellen. De ruimte waarin we geïnteresseerd zijn is de Euclidische ruimte $\mathbb{R}^d$ samen met de Euclidische afstand. Tenzij anders vermeld is $d$ gelijk aan 2. Verder wordt ook alleen het geprojecteerd coördinaten systeem beschouwd waarvan de coördinaten in de Euclidische ruimte liggen. Tenslotte wordt aangenomen dat het maximale bereik van de coördinaten voldoende groot is zodat alle objecten waarin we geïnteresseerd zijn hierin liggen.

Er zijn verschillende manieren om de geografische ruimte te benaderen, de twee belangrijkste manieren zijn *entity-based* en *field-based*. Een *entity-based* benadering zorgt voor een abstract model van de geografische data dat alle punten in de ruimte met gelijke eigenschappen verzamelt in een *feature*, ook wel *object* of *entiteit* genoemd. Hier worden punten (nul-dimensionale objecten), curves (één-dimensionale objecten) en vlakken (twee-dimensionale objecten) voor gebruikt. (Deze benadering hebben we dus in het voorgaande gedeelte gebruikt.)

In de *field-based* benadering, ook wel *space-based* genoemd, wordt aan ieder punt in de ruimte één of meerdere waardes toegekend. Een voorbeeld hiervan is de hoogte boven zeeniveau, per $x$ en $y$ waarde wordt hier een derde waarde voor de hoogte mee geassocieerd. Andere metingen die gedaan kunnen worden zijn neerslag, temperatuur en vervuiling. Deze benadering van de ruimte als een continu veld staat in contrast met die van de object-gebaseerde die een set van punten (krommes, vlakken) als een aparte feature

beschouwt. Het concept van een object is hier niet relevant.

De entity-based benadering van geografische data wordt meestal gebruikt door het vector model (zie paragraaf 2.2.4.2) en aanverwante modellen. De field-based benadering daarentegen wordt meestal door het raster model (zie paragraaf 2.2.4.1) gebruikt.

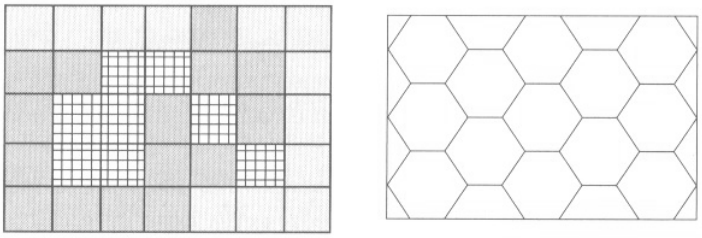
Opmerking bij het voorstellen van curves en vlakken met een entity-based model: hiervoor worden zeer vaak rechte lijnsegmenten gebruikt, niettegenstaande het gebruik van hogere-orde veeltermen een meer accurate voorstelling kan opleveren. Deze benadering vereenvoudigt de data opslag en zorgt voor het efficiënt modelleren en analyseren van de spatial data. Het nadeel hiervan is dat voor een getrouwe representatie van een kromme een hoge sampling rate nodig is, met als gevolg dat er veel segmenten en dus ook geheugencapaciteit nodig is.

Tot nu toe werden de ruimtelijke gegevens op een abstract niveau voorgesteld door punten, lijnen en vlakken. In deze paragraaf is het tijd voor een meer praktische implementatie hiervan. De moeilijkheid zit erin dat een oneindige set punten op een eindige wijze in een computer moet opgeslagen worden. Hiervoor bekijken we eerst het *raster model* en daarna het *vector model* en het *half-vlak model*. Alhoewel vector data in (bijna) alle commerciële GIS applicaties ondersteund wordt, wint het raster model de laatste jaren aan belangstelling omdat er steeds meer pixel-data beschikbaar is via satellieten en omdat field-based data ook meer en meer gebruikt wordt.

Maar het vector model is nog niet afgeschreven. Zo bestaan er variaties op het vector model, dat in zijn basisvorm relatief eenvoudig is. Zo laat het vector model niet toe om relaties weer te geven tussen verzamelingen van geometrieën. Daarom zijn er andere modellen gecreëerd die deze relaties wel voorstellen, deze zijn het *spaghetti model*, het *netwerk model* en het *topologische model*. Het verschil tussen deze modellen is de manier waarop ze *topologische relaties* tussen de geometrieën voorstellen. Topologische relaties zijn die relaties die invariant zijn onder topologische transformaties zoals translatie, rotatie of scalering in de Euclidische ruimte. Voorbeelden van deze relaties zijn aangrenzendheid, overlapping en disjunctie (zie paragraaf 2.3.3.1). Het expliciet opslaan van deze relaties in het model zorgt voor betere kennis over de geometrieën en snellere evaluatie van verschillende queries.

#### 2.2.4.1 Raster model

Bij het *raster* of *tessellation* (mozaïek) model wordt de ruimte opgesplitst in een eindig aantal cellen om een geometry voor te stellen. Het opdelen van de ruimte in cellen definieert een *discreet model*, ook wel een *spatial resolution model* of betegeling genoemd of *meshes* bij computer graphics. Dit model kan verder ingedeeld worden afhankelijk of de cellen *vast* of *variabel* zijn. Een vast raster model gebruikt een regelmatig grid om de ruimte op te delen dat bestaat uit een verzameling van veelhoeken met dezelfde grootte. Een variabel raster model echter maakt gebruikt van cellen van verschillende vorm en grootte, de grootte van de cellen kan bijvoorbeeld afhangen van de resolutie van de data. Figuren 2.9(a) en 2.9(b) zijn voorbeelden van vaste rasters, één met vierkante cellen en één met zeshoekige cellen. Figuur 2.10 toont twee voorbeelden van variabele rasters, figuur 2.10(a) stelt een kadastrale onderverdeling voor, terwijl figuur 2.10(b) een verdeling in Thiessen[3] veelhoeken toont.

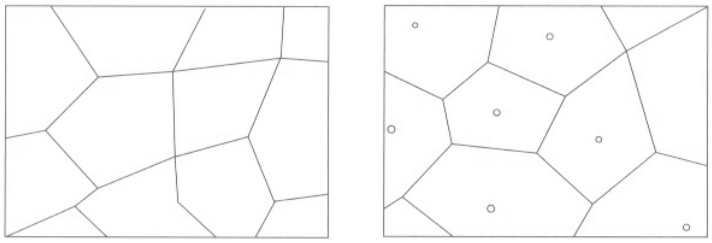


(a) Vierkante cellen (b) Hexagonale cellen

Figuur 2.9: Soorten vaste rasters (bron [25]).

Meestal wordt een regulier model gebruikt zoals in figuur 2.9(a), waarbij de zijden van de cellen evenwijdig zijn aan de assen van het coördinatenstelsel. Deze cellen worden *pixels* genoemd. Naar iedere cel kan verwezen worden met een adres in de vorm van een $(x, y)$ koppel, hiermee wordt naar de $x^e$ rij en de $y^e$ kolom verwezen. In de praktijk komt men dit raster model tegen om field-based data op te slaan bij toepassingen die afbeeldingen verwerken die door remote sensing verzameld zijn (satellietfoto's), hierbij wordt er gemapt van de oneindige set punten op het aardoppervlak naar een eindige set pixels.

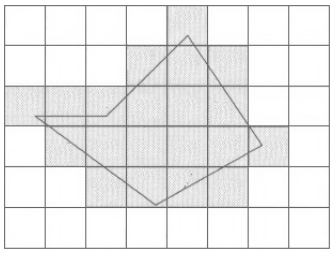
[3]Gegeven een set van punten $P$, een verdeling in Thiessen veelhoeken associeert met elk punt $p \in P$ een veelhoek, deze omvat de punten in de ruimte die het dichtste bij $p$ liggen. Deze indeling wordt ook een Voronoi diagram genoemd.



(a) Kadastrale zones (b) Thiessen veelhoeken

Figuur 2.10: Soorten variabele rasters (bron [25]).

Het is ook mogelijk om entity-based data met een raster model op te slaan, hiervoor wordt het raster gewoon over de features gelegd en wordt bijgehouden welke cellen overlappen met welke feature (zie figuur 2.11). Om een nauwkeurige representatie van de features te hebben, moet de resolutie van het raster hoog genoeg zijn (dus kleinere cellen). Dit vergt meer geheugencapaciteit, wat op zijn beurt zorgt voor de langzamere uitvoering van operaties op deze data.



Figuur 2.11: Raster voorstelling van een polygon (bron [25]).

#### 2.2.4.2 Vector model

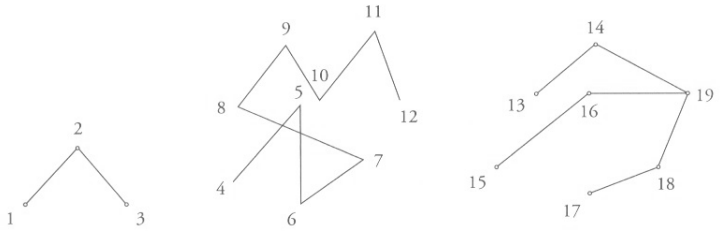
Het vector model gebruikt punten en segmenten als primitieven om objecten voor te stellen. Een punt wordt voorgesteld door zijn coördinaten, terwijl complexere objecten voorgesteld worden door lijsten of sets van punten. Deze voorstelling gebruikt, in tegenstelling tot het raster model, zeer weinig geheugen, een veelhoek wordt bijvoorbeeld voorgesteld door zijn hoekpunten.

Er bestaan verschillende methoden om een linestring, polygon of multipolygon voor te stellen. Hier wordt de volgende representatie gebruikt: Een

linestring is een *lijst* van punten $< p_1, ...p_n >$ waarbij opeenvolgende punten een segment van de linestring voorstellen. Een polygon is ook een lijst van punten, maar nu moet deze gesloten zijn, dat wil zeggen dat het koppel $(p_n, p_1)$ ook een segment van de polygon is. Een multipolygon is simpelweg een *set* van polygonen. Als we tuppels voorstellen door [ ], lijsten door $< >$ en sets door { }, dan kunnen we hun structuur als volgt samenvatten:

- punt: $[x : real, y : real]$
- linestring: $< punt >$
- polygon: $< punt >$
- multipolygon: $\{polygon\}$

Deze representatie laat veel vrijheid toe bij het instantieëren. Zo moet er bij een polygon de keuze gemaakt worden met welk hoekpunt er begonnen wordt en in welke volgorde dan verder gegaan wordt, met de klok mee of er tegen in. Er is ook geen zichtbaar onderscheid in de structuur van een linestring en een polygon, het is aan de onderliggende software om de geometry te interpreteren en deze te valideren. Hetzelfde geldt voor andere constraints op de geometrieën, zoals convexiteit of simpelheid. Figuur 2.12 illustreert drie gevallen van een linestring. Figuur 2.12(a) toont een linestring bestaande uit twee segmenten. Figuur 2.12(b) toont een niet-simpele linestring. En figuur 2.12(c) is volgens onze definitie geen linestring omdat er drie segmenten aankomen in één punt. Deze laatste geometry kan wel voorgesteld worden als we de multilinestring invoeren als een *set* van linestrings.

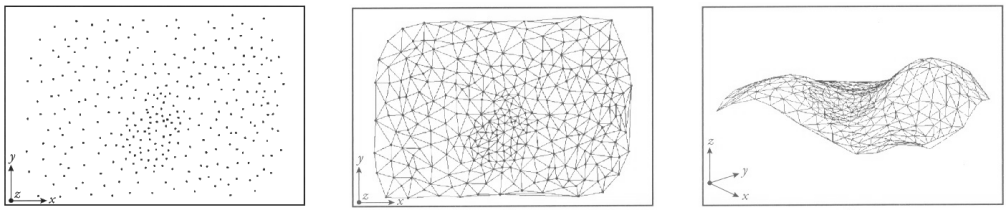



(a) (b) (c)

Figuur 2.12: Voorbeelden van LineStrings (bron [25]).

Het vector model wordt meestal gebruikt voor entity-based data, maar het is ook mogelijk om field-based data op te slaan. Een voorbeeld hiervan is het *Digital Elevation Model* (DEM) dat oorspronkelijk diende om de hoogte boven zeeniveau voor te stellen. Maar het kan ook iedere andere continue

twee-dimensionale ruimte voorstellen, zoals temperatuur of vochtigheid. Omdat de geografische ruimte door een eindig aantal punten wordt voorgesteld, moet er tussen de overige punten geïnterpoleerd worden. In het veelgebruikte *Triangulated Irregular Network* (TIN), een specifiek DEM, wordt lineair geïnterpoleerd door een netwerk van driehoeken. De verdeling en locatie van de punten is willekeurig, maar vaak worden er meer punten voorzien voor ongelijkmatige gebieden, zoals bergen, dan voor vlakke gebieden. In figuur 2.13(a) is de verdeling van de punten te zien, in figuur 2.13(b) is er geïnterpoleerd door willekeurige driehoeken en in figuur 2.13(c) tenslotte is de hoogte van ieder punt samen met de interpollatie gevisualiseerd.



(a) Puntenwolk (b) Indeling in driehoeken (c) TIN

Figuur 2.13: Voorbeeld van een Triangulated Irregular Network (bron [25]).

#### 2.2.4.3 Half-Vlak model

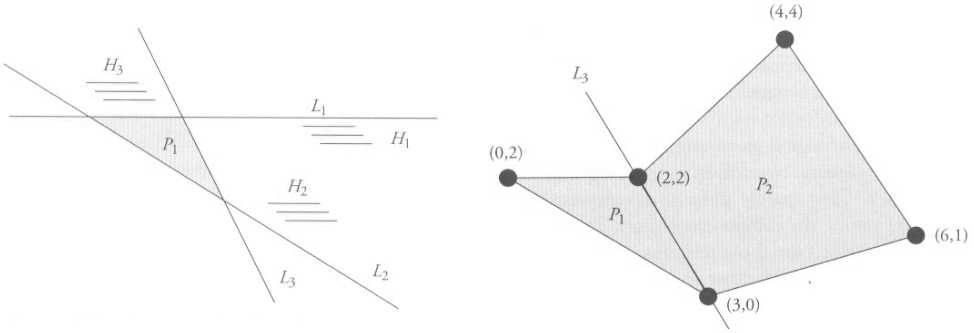
In tegenstelling tot het implementatie-afhankelijke vector model berust het half-plane model op stevige grondvesten. Alle geometrieën worden voorgesteld door één enkele primitieve, het half-vlak. Niettegenstaande dit model al een hele tijd geleden voorgesteld werd, is de interesse hierin pas de laatste jaren toegenomen. Het gevolg hiervan is dat dit model, net zoals het raster model, nog niet courant in commerciële database systemen terug te vinden is.

Een *half-ruimte* $H$ kan gedefinieerd worden in de ruimte $\mathbb{R}^d$ door een verzameling punten $P(x_1, ...x_d)$ die voldoen aan een vergelijking van de vorm:

$$a_1x_1 + ... + a_dx_d + a_{d+1} \leq 0$$

Een half-ruimte kan dus voorgesteld worden door de vector $[a_1, ...a_d, a_{d+1}]$. Een *convex d-dimensionaal veelvlak* kan dan voorgesteld worden door de doorsnede te nemen van een *aantal* gesloten half-ruimtes. Een willekeurig veelvlak kan bekomen worden door de unie te nemen van een eindig aantal convexe veelvlakken.

Een convexe polygon met $n$ hoekpunten (of zijdes) kan dus voorgesteld worden door de doorsnede van $n$ half-vlakken en een multipolygon door de unie van convexe polygonen. Deze definitie laat toe om niet-simpele polygonen, multipolygonen met gaten of simpele niet-convexe polygonen te definiëren. Een lijnsegment is een convex één-dimensionaal veelvlak, voorgesteld door twee half-lijnen die de eindpunten van het segment definiëren. Een linestring wordt voorgesteld door de unie van lijnsegmenten, deze kunnen eventueel wel niet-verbonden zijn. Een punt tenslotte is een nul-dimensionaal veelvlak en een multipoint is de unie van een verzameling punten. Figuur 2.14 toont twee voorbeelden van een polygon. Figuur 2.14(a) toont een convexe driehoek en figuur 2.14(b) toont een niet-convexe vijfhoek.




(a) Definitie van polygon $P_1$ (b) Niet-convexe polygon

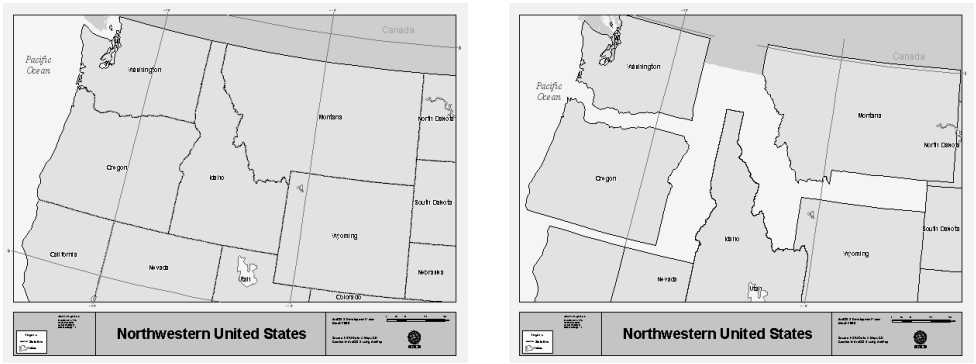
Figuur 2.14: Voorbeelden Half-Plane (bron [25]).

*Opmerking*: Een niet-convexe veelhoek kan op verschillende manieren in convexe deel-veelhoeken gesplitst worden, zo kon de vijfhoek in figuur 2.14(b) onder andere ook volgens het half-vlak door de punten (0,2) en (2,2) gesplitst worden. Een andere opmerking is dat dit model niet uitgebreid hoeft te worden als data met een hogere dimensie dan twee opgeslagen moet worden.

#### 2.2.4.4 Spaghetti model

Dit model stelt iedere geometry voor als een op zichzelf staand object. Er wordt dus geen informatie opgeslagen over topologische relaties tussen objecten, met gevolg dat deze telkens opnieuw volledig berekend moet worden. Dit model is te vergelijken met het gewone vector model in paragraaf 2.2.4.2, maar dan voor verzamelingen van features. Deze voorstelling introduceert redundante data, zo wordt bijvoorbeeld de grens tussen twee aangrenzende

gebieden twee keer opgeslagen, één keer per gebied. Het voordeel van dit model is zijn eenvoud en de mogelijkheid om punten, linestrings en polygonen samen en zonder restricties, zoals het verplicht opslaan van een intersectiepunt van twee linestrings, op te slaan. Het nadeel is het ontbreken van topologische informatie in het model en de redundantie van data, zodat er gevaar dreigt op inconsistentie bij het enkel updaten van één linestring van een gemeenschappelijke grens van twee gebieden. Een voorbeeld van de op zichzelf staande features is te zien in figuur 2.15.

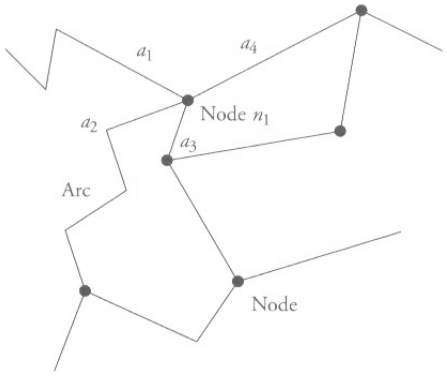



(a) Aangrenzende objecten (b) Niet-aangrenzende objecten

Figuur 2.15: Voorbeeld Spaghetti model.

#### 2.2.4.5 Netwerk model

Dit model is ontwikkeld om netwerken voor te stellen in netwerk-gebaseerde applicaties, zoals het beheer van wegen en elektriciteit. Er wordt alleen informatie opgeslagen over topologische relaties tussen punten en linestrings. Hiervoor moeten twee nieuwe concepten geïntroduceerd worden, namelijk *nodes* en *arcs*. Een node is een speciaal soort punt dat een lijst van arcs met elkaar verbindt. Een arc is een linestring die begint en eindigt in een node. Figuur 2.16 toont een voorbeeld van een netwerk, hierbij verbindt de node $n_1$ de vier arcs $a_1$, $a_2$, $a_3$ en $a_4$. Met deze informatie is het mogelijk om te *navigeren* door het netwerk door bij iedere node een uitgaande arc te kiezen, hiermee kan bijvoorbeeld het kortste pad tussen twee nodes berekend worden. De toevoeging van nodes aan het model betekent niet dat er geen gewone punten meer zijn, de gewone punten dienen nog altijd om punten op linestrings en hoekpunten van polygonen op te slaan. Een node is een eindpunt van een arc ofwel, wanneer er geen arcs mee verbonden zijn, een geïsoleerd punt in de ruimte. Afhankelijk van de implementatie kan het netwerk




Figuur 2.16: Voorbeeld van een Netwerk (bron [25]).

*planar* (in één vlak liggende) zijn of *nonplanar* (niet in één vlak liggende). Een netwerk is planar als iedere intersectie van linestrings een node vormt en nonplanar als dit niet het geval is. Dit laatste kan bijvoorbeeld in een wegennetwerk duiden op een tunnel of een brug. De structuur van dit model lijkt op die van het spaghetti model, maar hier is de linestring vervangen door de volgende twee structuren: *node*: $[punt, < arc >]$ en *arc*: $[node\text{-}start, node\text{-}end, < punt >]$. Een laatste opmerking over het netwerk model is dat er geen informatie over topologische relaties tussen *twee-dimensionale* objecten wordt opgeslagen, hiervoor is het topologisch model nodig.

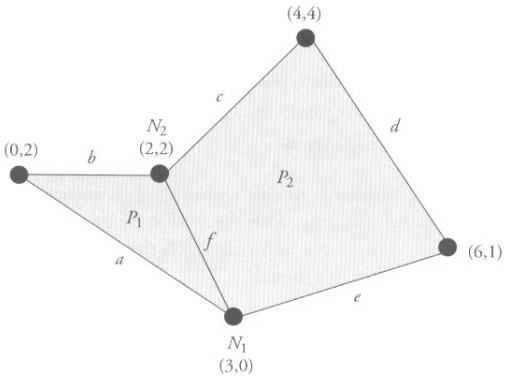
#### 2.2.4.6 Topologisch model

Dit model bouwt verder op het planar netwerk model, het deelt de ruimte op in aangrenzende polygonen. Het kan wel voorkomen dat sommige van deze polygonen geen geografische objecten voorstellen in de reële wereld. Omdat het netwerk planar is, zorgt iedere intersectie van linestrings voor een node en wordt de rand van iedere polygon door een aantal arcs bepaald. De structuur van het topologische model kan als volgt samengevat worden:

- punt: $[x : real, y : real]$
- node: $[punt, < arc >]$
- arc: $[start\text{-}node, end\text{-}node, left\text{-}poly, right\text{-}poly, < punt >]$
- polygon: $< arc >$
- multipolygon: $\{polygon\}$

Een arc wordt uitgebreid met een referentie naar zijn linker en rechter polygon en een polygon wordt nu voorgesteld door een lijst van arcs. Dit model

bevat nog altijd redundantie, maar in een beperkte mate. Een polygon kan zowel aangesproken worden via de polygonen of via de arcs. Er is echter geen redundantie bij de opgeslagen geometry, iedere punt en linestring komt maar één keer voor. Een voorbeeld van het topologisch model is te zien in figuur 2.17.




Figuur 2.17: Voorbeeld van een Topologie (bron [25]).

De polygon $P_1$ kan voorgesteld worden door de lijst <a, b, f>, polygon $P_2$ door <c, d, e, f>, arc $f$ door het tuppel $[N_1, N_2, P_1, P_2, <>]$ en node $N_1$ door $[[3, 0], < a, f, e >]$. De reden dat de lijst in arc $f$ leeg is, is omdat $N_1$ en $N_2$ door een rechte linestring verbonden zijn. Een voordeel van het topologisch model is dat topologische queries efficiënt berekend kunnen worden, zo kunnen aangrenzende polygonen van een gegeven polygon $P$ simpelweg gevonden worden door de arcs van $P$ te scannen en de polygonen uit te lezen. Een ander voordeel is dat updates van de geometrieën gemakkelijker gaan en er minder kans is op inconsistentie. Er zijn echter ook nadelen aan dit model verbonden. Zo hebben sommige polygonen geen equivalent in de reële wereld. Of kan het zijn dat de complexe structuur sommige operaties vertraagt, zoals bij het weergeven van een subset van de data. Hierbij moet er door de complexe structuur genavigeerd worden, wat leidt tot flink wat overhead. Een laatste punt is dat bij het invoegen van nieuwe geometrieën een deel van de reeds bestaande data geladen moet worden en er voor eventuele nieuwe intersecties extra nodes toegevoegd moeten worden.

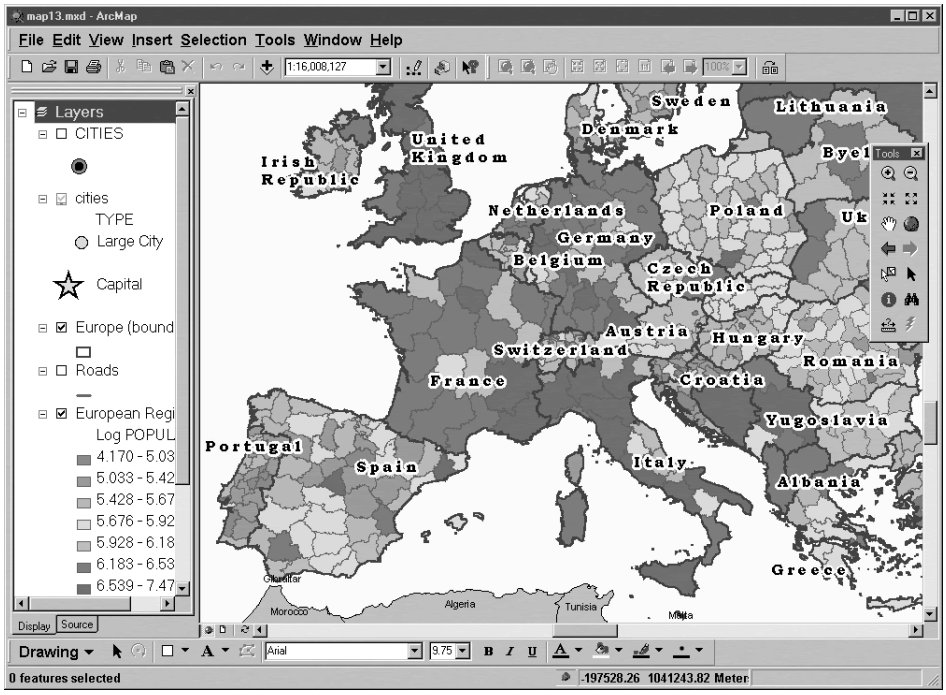
## 2.3 Analyse

Nu we weten hoe ruimtelijke gegevens voorgesteld worden, gaan we kijken naar hoe we deze kunnen analyseren. Dit is één van de belangrijkste taken

van GIS, zonder deze functionaliteit zou een GIS alleen maar dienen om spatial data op te slaan en om ze te visualiseren! Eerst kijken we naar de mogelijke interfaces in GIS om gegevens te analyseren [3]. Dan worden de soorten analyses vermeld en tenslotte worden deze types besproken en geïllustreerd (hoofdstukken 1 en 3 van [25]).

### 2.3.1 Interface

In een GIS zijn er twee manieren om de spatial data te ondervragen. Bij de eerste manier kunnen de attributen van één locatie of entiteit opgevraagd worden. Meestal gebeurt dit in de weergave-modus door er met de muis op te klikken. Bij een raster-GIS kan dan de waarde van de aangeklikte pixel opgevraagd worden, evenals positie in het raster $(rij, kolom)$ en/of zijn coördinaten $(x, y)$. Ook de identificatiecode van de pixel kan getoond worden samen met andere aanwezige attributen. In een vector-GIS wordt de aangeklikte geometry geselecteerd en worden zijn attributen gevisualiseerd. In figuur 2.18 is een stuk van de kaart van Europa te zien in *ArcMap* (een onderdeel van ArcGIS), een van de taken van deze module is het weergeven en bevragen van spatial data.

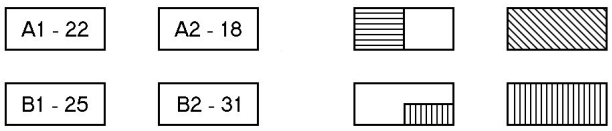



Figuur 2.18: Analyse in ArcGIS met ArcMap (bron [31]).

Bij de tweede manier van bevraging worden alle objecten of plaatsen geselecteerd die aan een bepaald criterium voldoen. Dit gebeurt meestal door eerst de layer(s) op te geven waarin men geïnteresseerd is, vervolgens één of meerdere attributen ervan te selecteren en tenslotte deze door een predikaat te beperken. Een predikaat kan bijvoorbeeld eisen dat een attribuut, zoals een ID, gelijk is aan of kleiner is dan een bepaalde waarde. Of er kan geëist worden dat een object niet verder dan $x$ km van een bepaald punt mag liggen.

De bevraging van data kan in drie groepen onderverdeeld worden. Queries met een *alfanumeriek* criterium, queries met een *ruimtelijk* criterium en *interactieve* queries. Tot de eerste groep horen queries zoals: "Uit hoeveel provincies bestaat België?". In de tweede groep zitten queries zoals: "Toon de buurlanden van België.". En tot de derde groep: "Op welk land werd er geklikt?". In de rest van deze paragraaf worden enkele queries uit de drie groepen toegelicht en geïllustreerd op een voorbeeld.

*Het voorbeeld* waarmee gewerkt wordt is de voorstelling van een campus van een universiteit met verschillende afdelingen. Hiervoor zijn twee tabellen nodig, één voor de gebouwen op de campus en één voor de afdelingen binnen de gebouwen. Een tuppel in 'Gebouwen' bestaat uit een string (met een unieke naam), een getal (met het aantal bureaus per gebouw) en een spatial type *polygon* met de naam *geo* (voor de geografische voorstelling). Een tuppel in 'Afdelingen' heeft een string (met een unieke naam) en een multipolygon (voor de geografische voorstelling). De schema's van deze tabellen kunnen als volgt samengevat worden: *Gebouwen*(`naam, aantal_bureaus, geo:polygon`) en *Afdelingen*(`naam, geo:multipolygon`).




(a) Gebouwen (b) Afdelingen

Figuur 2.19: Campus-schema.

Een instantie van deze twee relaties wordt getoond in figuur 2.19. Hierbij stellen de witte en de gearceerde delen in figuur 2.19(b) de verschillende afdelingen voor. Nu zijn we klaar om de drie analyse-groepen te bespreken.

### 2.3.2 Metrische queries

Deze groep van queries is niet zo interessant voor ons, dit komt omdat deze queries ook uitgevoerd kunnen worden op relationele databases. Het enige verschil dat er hier is, is dat de tabellen waarop de bevraging gedaan wordt toevallig ook nog een spatial kolom hebben, maar deze wordt niet gebruikt in de query. Een voorbeeld van zo een query is: "Geef de namen van alle gebouwen op de campus" of "Hoeveel bureaus zijn er in gebouw $A_1$?".

### 2.3.3 Spatial queries

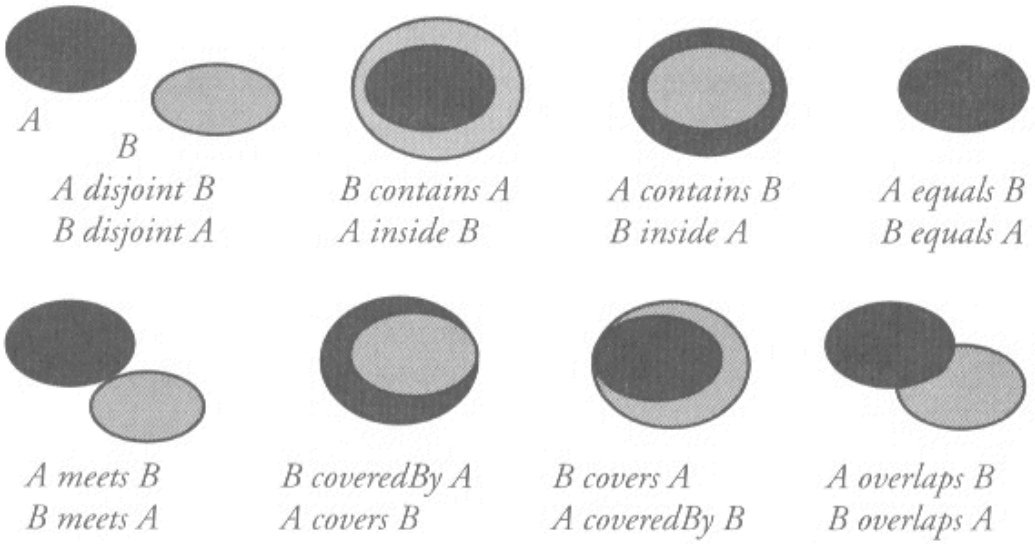
Deze groep van queries is wel interessant voor ons, omdat in deze en de volgende groep de geometrieën in queries gebruikt worden. In deze queries kunnen simpele operaties gebruikt worden, zoals: "*Toon* alle afdelingen in gebouw $A_1$.". Maar ook meer complexe operaties kunnen uitgevoerd worden, zoals: "Wat is de gemiddelde oppervlakte van een bureau in een bepaalde afdeling?" Hierbij moeten de twee tabellen of layers samen gebruikt worden. De afdelingen-relatie om de oppervlakte te bepalen van de geometry van de afdeling en de gebouwen-relatie om het aantal bureaus per oppervlakte van ieder gebouw te bepalen. Als een afdeling volledig in één gebouw ligt zijn we nu klaar, maar als een afdeling over meerdere gebouwen verspreid is, dan moet nog het gewogen gemiddelde van de oppervlaktes in de verschillende gebouwen berekend worden.

De enige beperking die op deze queries ligt, zijn de door het systeem voorziene operaties op geometrieën (en de beperkingen van Structured Query Language (SQL)). Andere voorbeelden van veelgebruikte operaties zijn: geef de lengte van een linestring, toon alle linestrings die door een bepaalde polygon lopen en toon alle polygonen die aan een bepaalde polygon grenzen. Om deze operaties uit te kunnen voeren moet het systeem de volgende functionaliteit bieden: de afstand tussen twee punten kunnen bepalen, de doorsnede van twee (verschillende) geometrieën kunnen berekenen en nagaan of aan de topologische relaties voldaan wordt. De meeste lezers zullen wel vertrouwd zijn met de termen afstand en doorsnede, maar omdat waarschijnlijk niet iedereen weet wat topologische relaties inhouden, gaan we hier dieper op in.

#### 2.3.3.1 Topologische relaties

Soms is het niet voldoende om te weten of de doorsnede van twee geometrieën leeg is of niet. Stel dat men wil weten welke gemeenten in een provincie aan de grens liggen van die provincie. Dan kan men niet zomaar die gemeenten

nemen die in de doorsnede met de provincie liggen en ook niet de gemeenten die in de doorsnede zitten met de grens van de gemeente. De oplossing hier is om de doorsnede van de twee vorige oplossingen te nemen. Om dit soort relaties gemakkelijker te bepalen heeft men de topologische relaties ingevoerd. Deze relaties zijn gedefinieerd met behulp van het inwendige en de grens van de geometrieën. (*Nota*: Sommige modellen gebruiken ook het uitwendige van de geometrieën, zie [12].) De topologische relatie tussen twee geometrieën kan formeel voorgesteld worden door te kijken naar de doorsnedes van het innerlijke en de grens van de ene geometry met het innerlijke en de grens van de andere geometry. Er zijn dus vier doorsnedes en ofwel is deze leeg ($\emptyset$) ofwel is deze niet leeg ($\neg\emptyset$). In theorie zijn er dus $2^4 = 16$ relaties mogelijk, maar omdat de helft hiervan niet mogelijk is, blijven er nog acht over. Deze zijn te zien in figuur 2.20.




Figuur 2.20: Topologische relaties (bron [25]).

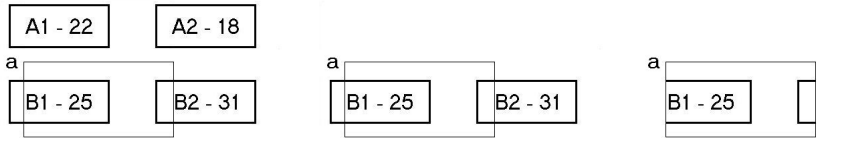
Er zijn echter twee uitzonderlijke gevallen waarbij een onmogelijke relatie toch mogelijk is, zie figuur 2.21. Dit komt voor als we punten definiëren als zijnde alleen een rand (dus geen inwendige) en als we de twee eindpunten van een linestring definiëren als zijnde de rand en de segmenten hiertussen als het inwendige. Zo kan de doorsnede van het inwendige van één geometry met het inwendige en de grens van de andere geometry $\neg\emptyset$ zijn, zonder dat de twee grenzen van de geometrieën elkaar snijden. Dit komt onder andere voor als een linestring buiten een polygon begint en in de polygon eindigt.


(a) A on B (b) A overlap_bdy_disjoint B

Figuur 2.21: Uitzonderingen Topologische relaties (bron [19]).

### 2.3.4 Interactieve queries

Bij deze laatste groep van queries moet de gebruiker zelf input geven, meestal gebeurt dit door op een object te klikken of door met de muis een selectie-rechthoek te trekken op het scherm. Deze queries vindt men ook terug onder de naam *geometric* of meetkundige selectie. De drie operaties die hierbij het meest voorkomen zijn: punt query (stabbing), windowing en clipping. Bij de eerste moet de gebruiker een punt opgeven, terwijl bij de laatste twee de gebruiker een rechthoekige polygon moet voorzien. Bij een *punt query* worden alle tuppels teruggegeven waarvan de geografische voorstelling het gegeven punt bevat. Dit is vergelijkbaar met het steken van een duimspijker op een kaart en te kijken welk(e) gebied(en) er geraakt zijn, vandaar de naam *stabbing*. Bij *Windowing* worden alle tuppels teruggegeven waarvan de geografische voorstelling (al dan niet gedeeltelijk) in de doorsnede met de gegeven polygon ligt. Omdat deze polygon gewoonlijk een rechthoek is, noemt men het daarom ook een *window*. *Clipping* tenslotte extraheert het stuk van de layer dat in een gegeven polygon ligt. Het verschil met windowing is dat bij clipping alleen maar dat deel van de geometry wordt teruggegeven dat in de doorsnede ligt, terwijl bij windowing altijd de volledig geometry wordt teruggegeven. In figuur 2.22 worden windowing en clipping aan de hand van een polygon $a$ geïllustreerd.




(a) Gebouwen & polygon $a$ (b) Windowing (c) Clipping

Figuur 2.22: Windowing versus Clipping.

## 2.4 Data input en output

In de vorige paragrafen hebben we besproken wat GIS is, waaruit het opgebouwd is en hoe de ruimtelijke gegevens geanalyseerd kunnen worden. Nu is het tijd om te bespreken hoe we die gegevens in de eerste plaats invoeren en hoe we de gegevens er, eventueel na analyse, weer uit krijgen.

### 2.4.1 Input

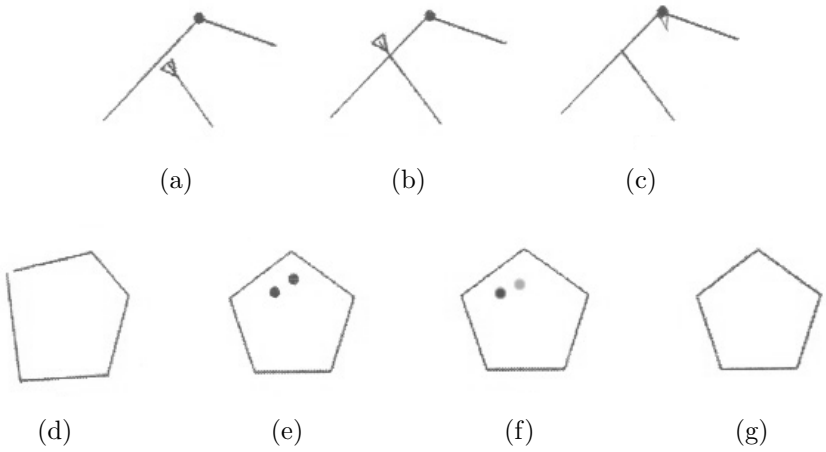
De *bottleneck* van GIS is de invoer van data. Het invoeren van ruimtelijke gegevens kan op verschillende manieren gebeuren. Het kan door deze zelf te verzamelen en te *digitaliseren* [3] of we kunnen niet-ruimtelijke gegevens, zoals adressen, omzetten naar ruimtelijke gegevens met *geocoding* ([3] en [19]). Nog een andere manier is door *spatial functies*, die nieuwe ruimtelijke gegevens produceren, op bestaande data in het GIS los te laten (hoofdstuk 22 van [7]). Of we kunnen ruimtelijke gegevens die al door anderen verzameld zijn gewoon *importeren* (hoofdstuk 10 van [7]).

#### 2.4.1.1 Digitalisatie

Alvorens bruikbaar te zijn in een GIS moeten (analoge) gegevens omgezet worden in een digitaal formaat. Dit geldt zowel voor tekst, cijfers, kaarten als beelden. Alfanumerieke gegevens worden meestal ingetikt, soms ingescand en daarna geïnterpreteerd met een Optical Character Recognition (OCR) pakket. Deze data wordt dan aan een constructor functie meegegeven die de gepaste geometry, zoals een punt, linestring of polygon, genereert met de opgegeven coördinaten. Beelden worden meestal ingescand en eventueel nabewerkt. Kaarten kunnen ofwel op een digitaliseertafel of -tablet worden gevectoriseerd ofwel kan men ze inscannen en achteraf op scherm vectoriseren. Merk op dat gescande informatie steeds ongestructureerd is en dus geen topologie bevat. Ze is bovendien schaalgebonden.

Het vectoriseren van data kan door met de pen van een digitaliseertablet de features op een kaart na te trekken of met de muis de features op het scherm na te gaan. Problemen die zich hier bij kunnen voordoen zijn weergegeven in figuur 2.23.

Figuren 2.23(a) en 2.23(b) tonen een *undershoot* respectievelijk een *overshoot*, dit betekent dat de lijn te kort respectievelijk te lang is om met zijn eindpunt op een andere lijn uit te komen. Met het gebruik van een *snap*-functie, die automatisch de lijnen verbindt als de fout kleiner is dan een




Figuur 2.23: Digitaliseerfouten (bron [3]).

bepaalde tolerantie-waarde, kan deze fout verholpen worden. Figuur 2.23(c) toont een lijn die niet aansluit bij een node, maar die wel met een lijn verbonden is. Als de fout kleiner is dan een bepaalde tolerantiewaarde, is ze corrigeerbaar bij het opbouwen van een topologie in een later stadium. Figuur 2.23(d) toont een gelijkaardige fout als $a$ en $b$, maar nu met een polygon die niet gesloten is. Door het gebruik van de snap-functie of bij de topologische opbouw kan deze fout verbeterd worden. Figuren 2.23(e), 2.23(f) en 2.23(g) gaan allemaal over fouten met labelpunten. Bij $e$ zijn de twee labels dubbel, bij $f$ zijn er twee verschillende labels en bij $g$ ontbreekt het label. De fout in $e$ is corrigeerbaar bij de opbouw van de topologie, terwijl de twee andere fouten dan alleen maar detecteerbaar zijn.

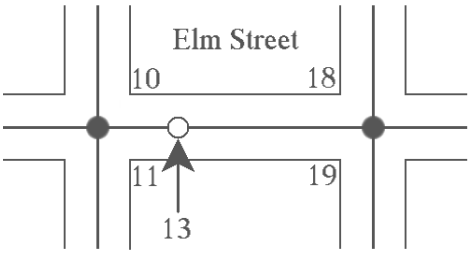
Naast de manuele vectorisatie die hierboven beschreven is, bestaan er ook programma's die deze taak voor hun rekening nemen. Deze programma's werken meestal in twee modes, een automatische mode waarbij er geen interactie met de gebruiker nodig is. En een half-automatische mode, hierbij kan de gebruiker, na iedere interpretatie van een feature door het programma, wijzigingen in deze interpretatie aanbrengen. Voor complexere zones kan zelfs terug naar manuele vectorisatie overgestapt worden. Essentieel bij het automatisch vectoriseren is de kwaliteit van het gescande document. De algoritmes die door deze programma's gebruikt worden, steunen op een proces van *skeleting*, hierbij worden lijnen op het gescande document omgezet naar een raster van pixels met één-pixel lijnen. Deze lijnen worden dan nog bijgewerkt om onnauwkeurigheden te verwijderen en worden daarna omgezet

naar vector geometrieën.

#### 2.4.1.2 Geocoding

Een tweede manier om ruimtelijke gegevens te verkrijgen is door het gebruik van een *geocoder*, een programma dat de geocoding uitvoert. Hierbij vertrekt men van informatie in een indirect referentiesysteem. Dit is een referentiesysteem waar geen gebruik gemaakt wordt van een coördinaten systeem, maar voor de lokalisatie van objecten verwijst naar andere objecten of entiteiten waarvan men de lokalisatie wel kent.

Het meest voorkomende geval is het lokaliseren aan de hand van adressen. Hiervoor wordt eerst het adres ontleed, dit gebeurt in de velden: straat, huisnummer, postcode, gemeente en eventueel nog andere. Dan gaat de geocoder in zijn referentiedata kijken of dit adres overeenkomt met zijn gegevens. Deze match hoeft niet 100 procent overeen te komen met het input adres, maar moet toch boven een bepaald percentage liggen. Als er meerdere adressen matchen kan de gebruiker gevraagd worden welk adres het juiste is. Dan wordt de geometry van het punt bepaald dat met het adres overeen komt. Hiervoor kijkt de geocoder naar zijn referentiedata waarin alleen maar straat segmenten zitten met huisnummers voor de uiteinden. Het punt wordt geïnterpoleerd tussen deze uiteinden aan de hand van het input huisnummer en de veronderstelling dat de huisnummers evenredig gespreid zijn over de straat segmenten. Figuur 2.24 illustreert de interpollatie voor "Elm Street, nummer 13".



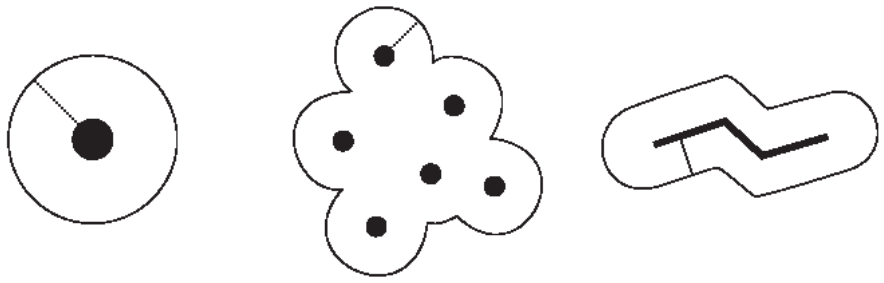

Figuur 2.24: Voorbeeld geocoding (bron [19]).

Als het input adres niet volledig is, wordt er geïnterpoleerd op een hoger niveau waarvoor er wel gegevens zijn. Als bijvoorbeeld alleen maar "Elm Street" opgegeven werd als adres, dan zou het middelpunt van de straat teruggegeven worden. Als er zelfs geen straat opgegeven werd dan zou het

middelpunt van de gemeente teruggegeven worden (bijvoorbeeld voor het opvragen van een weersvoorspelling), enzovoorts.

#### 2.4.1.3 Spatial functies

Spatial functies zijn functies die één of meerder geometrieën als input nemen en één of meerdere geometrieën als resultaat hebben. Voorbeelden hiervan zijn de set operaties *doorsnede*, *verschil*, *unie* en *symmetrisch verschil* (XOR), maar ook de functies *buffer*, *convex hull* en *aggregatie* functies creëren nieuwe geometrieën. Het resultaat van deze functies wordt meestal maar tijdelijk opgeslagen en dient alleen maar voor analyse. Om te kijken welke features allemaal in een straal van $x$ km rond een winkel liggen, kan een buffer van $x$ km rond de winkel gecreëerd worden. Hierna hoeft men alleen nog maar te controleren of er features met deze buffer overlappen. Voorbeelden van de buffer functie zijn te zien in figuur 2.25.



(a) Rond een punt (b) Rond een multipoint (c) Rond een linestring

Figuur 2.25: Buffers rond geometrieën (bron [7]).

Andere spatial functies kunnen gebruikt worden om reeds bestaande geometrieën aan te passen. Voorbeelden hiervan zijn functies om een punt toe te voegen, te wijzigen of te verwijderen. Veel van deze functies hangen af van de implementatie van de geometrieën en de methoden die erop gedefinieerd zijn.

#### 2.4.1.4 Import

Een vierde manier om ruimtelijke gegevens in een GIS te krijgen is door ze te importeren via bestanden die door externe bronnen geleverd worden. Deze bestanden bevatten vaak kaart gerelateerde gegevens, zoals wegennetwerken, grondgebruik, enzovoorts. Door deze externe gegevens te combineren met zelf geproduceerde gegevens, zoals risicogebieden voor overstromingen, kan

men bijvoorbeeld gemakkelijk beslissen waar een wachtbekken voor water aangelegd moet worden.

Het importeren van data wordt gedaan als men, zoals in het bovenstaand voorbeeld, gewoon meer informatie wil om beslissingen te nemen. Een andere situatie kan zijn om te migreren van één database naar een andere, hiervoor moeten de gegevens wel eerst geëxporteerd worden uit de eerste database. Een derde situatie is om de gegevens te visualiseren in een externe toepassing terwijl men niet met de database verbonden is. Hiervoor moet de data ook eerst geëxporteerd worden om daarna te importeren in de externe applicatie.

### 2.4.2 Output

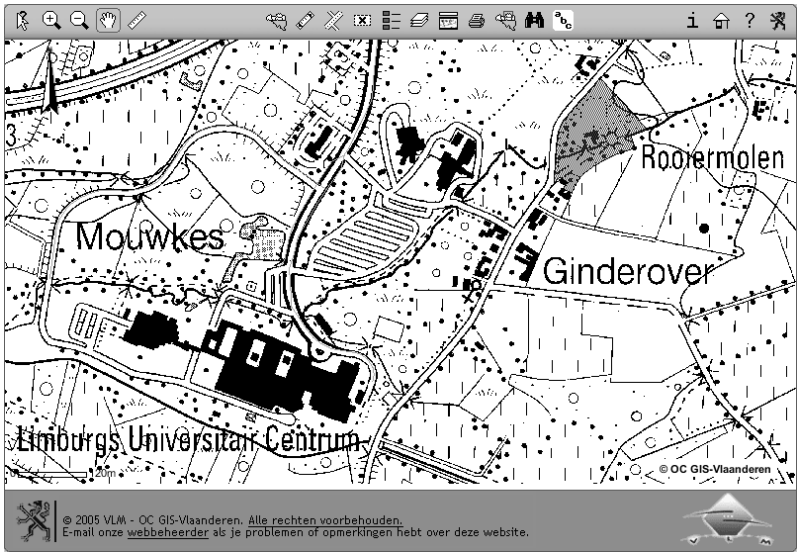
Het presenteren van ruimtelijke gegevens kan ook op verschillende manieren gebeuren. Een eerste manier is het *visualiseren* van de data op het scherm of op een kaart [3]. Een tweede mogelijkheid is het *exporteren* van de data zodat deze in een ander GIS opnieuw geïmporteerd kan worden (hoofdstuk 10 van [7]).

#### 2.4.2.1 Visualisatie van informatie

Bij het visualiseren van ruimtelijke gegevens zijn er twee mappings die moeten gebeuren. De eerste mapping zorgt voor de weergave van punten en lijnen van geometrieën. De tweede mapping zorgt ervoor dat de alfanumerieke gegevens van de geometrieën weergegeven worden. Zo kan de inkleuring van een polygon het landgebruik voorstellen of kan een icoon in de plaats van een gewoon punt een luchthaven voorstellen.

De eerste mapping moet de geometrieën inlezen en vervolgens gewoon zijn coördinaten tonen. Voor de tweede mapping zijn de volgende methoden beschikbaar: een andere *Grijswaarde of tint*, een andere *Rasterstructuur* of een andere *Kleur* voor de opvulling van polygonen, en verschillen in de *Grootte*, een andere *Textuur*, verschillen in *Richting* of een andere *Vorm* van iconen. De eigenschappen die op een kaart kunnen worden uitgedrukt verschillen per methode. Om *groepering* van geometrieën aan te geven kan men ze eenzelfde rasterstructuur, kleur, textuur, richting of vorm geven. Alle methoden, behalve de vorm, zorgen voor *beeldvorming*, dat wil zeggen dat de kaartlezer in één oogopslag een overzicht krijgt van het afgebeelde gebied. Het bekomen van een *rangorde* kan door verschillende groottes, grijswaardes of rasterstructuren te gebruiken. Om een *kwantiteit* aan te geven kan alleen de grootte

van een icoon gebruikt worden. En tenslotte kan ook nog van labels gebruik gemaakt worden om attributen weer te geven die niet door de voorgaande methoden getoond kunnen worden.




Figuur 2.26: Voorbeeld van visualisatie (bron [21]).

Nu de ruimtelijke en een aantal alfanumerieke gegevens weergegeven kunnen worden, rest alleen nog de keuze *waarop* dat we ze gaan weergeven. Dit kan op een papieren kaart of een *scherm-kaart*. Hierbij moeten voor- en nadelen van de keuze tegen elkaar afgewogen worden. De voordelen van een schermkaart zijn: actualisering van de gegevens, interactiviteit (men kiest zelf welke layer men wil zien) en gemakkelijk te verspreiden (bijvoorbeeld via internet). De nadelen hiervan zijn: de beperkte beeldresolutie van een scherm ten opzichte van een kaart (er kan wel ingezoomd worden), beperkt aantal informatiepunten dat in één keer kan bekeken worden (beeldvorming kan hierdoor moeilijker zijn), moeilijkheden met de verspreiding van copyright materiaal en onpraktisch in gebruik in terreinomstandigheden.

#### 2.4.2.2 Export

Een andere mogelijkheid om gegevens uit een GIS te krijgen is door ze te exporteren naar bestanden. Deze bestanden kunnen dan door anderen geimporteerd worden om te analyseren of gewoon om ze te visualiseren. De meeste databases ondersteunen export-mogelijkheden naar verschillende bestandstypes, maar hebben zelf ook vaak een eigen bestandsformaat hiervoor ontwikkeld.

# Hoofdstuk 3

# GIS & Relationele DB

Er zijn, zoals vermeld in het vorige hoofdstuk, twee soorten gegevens in een GIS, geografische en alfanumerieke data. Een GIS moet deze gegevens kunnen invoeren en opslaan, opvragen en analyseren en kunnen presenteren. In de begindagen van GIS werd dit allemaal gedaan door de applicatie zelf die zijn gegevens gewoon in bestanden opsloeg. Het gevolg hiervan was dat een GIS zelf de interne data-structuur moest kennen om de gegevens op te slaan en te analyseren. Dit is strijdig met het data-onafhankelijkheids principe en zorgt voor problemen met data robuustheid en bij gelijktijdige toegang. Hieronder volgen enkele voorstellen om dit probleem op te lossen.

## 3.1 "Loosely coupled" benadering

Vele huidige GIS scheiden het beheer van de ruimtelijke gegevens van het beheer van de attributen (hoofdstuk 1 van [25]). Met een architectuur als deze zijn er dan ook twee verschillende systemen nodig: Eén (relationeel) DataBase Management System (DBMS) voor het beheren van de attributen (alfa-numerieke gegevens). En één speciale module voor het beheer van de ruimtelijke gegevens. Deze worden meestal als gewone bestanden opgeslagen in het file-systeem van het besturingssysteem.

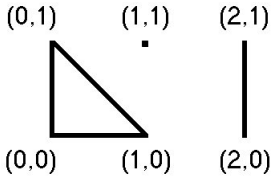
Bij deze benadering wordt de structuur van zowel de attributen als de ruimtelijke data netjes verborgen achter de twee systemen. Dit houdt natuurlijk wel in dat de GIS applicatie de twee interface-talen van deze systemen moet 'kennen'. Omdat er verschillende bestandsformaten zijn voor ruimtelijke gegevens, zoals bijvoorbeeld coverages en shape files, moet de module elk datamodel kunnen converteren naar een gemeenschappelijk interface-model dat de GIS applicatie begrijpt. Alhoewel deze aanpak zijn diensten reeds bewe-

zen heeft, zijn er toch enkele tekortkomingen aan verbonden, de belangrijkste hiervan zijn:

- Het naast elkaar bestaan van de heterogene data modellen in de bestanden, dit houdt in dat de spatial module hier rekening mee moet houden waardoor hij complexer wordt in implementatie en gebruik
- Het gemis van basis DBMS functionaliteit, zoals querying, optimalisaties en recovery

## 3.2 Een relationele DBMS gebruiken

Omdat, bij het gebruik van een file-systeem, alle spatial functionaliteit zelf geïmplementeerd moet worden en er toch al een DBMS gebruikt wordt voor de alfanumerieke data, was men op het idee gekomen om de ruimtelijke gegevens gewoon in de relationele database op te slaan samen met de alfanumerieke gegevens (hoofdstuk 1 van [25] en hoofdstuk 14 van [5]). Hierbij worden de gegevens per layer in een aparte relatie/tabel opgeslagen. Een geometry kan voorgesteld worden door één of meerdere rij(en) met coördinaten in een relatie. Maar hiervoor kan enkel gebruik gemaakt worden van reeds bestaande alfanumerieke types zoals *string* en *real*. Het analyseren van de data kan hier wel, zowel voor de attributen als de spatial gegevens, met behulp van SQL. Deze benadering wordt geïllustreerd in figuur 3.1.




| ID | $X$ | $Y$ |
|---|---|---|
| driehoek_1 | 0 | 0 |
| driehoek_1 | 1 | 0 |
| driehoek_1 | 0 | 1 |
| punt_a | 1 | 1 |
| lijn_A | 2 | 0 |
| lijn_A | 2 | 1 |

Figuur 3.1: Features in een Relationele Database.

Figuur 3.1 laat zien dat het mogelijk is om zowel punten, lijnen als veelhoeken op te slaan in relationele databases. Maar het toont ook dat de gebruiker zelf onderscheid moet maken tussen de verschillende types, bijvoorbeeld door ze in verschillende tabellen te plaatsten. Er kan ook niet gemakkelijk gecontroleerd worden of de geometrieën aan hun eigenschappen voldoen, bijvoorbeeld

of een veelhoek gesloten is. En het opslaan van complexe geometrieën, zoals veelhoeken met gaten en/of gebogen zijdes, vergt enige creativiteit van de gebruiker. Er zijn nog andere moeilijkheden bij het gebruik van relationele DBMS, zoals de volgorde van hoekpunten in een veelhoek bepalen, maar we houden het bij de hier vermelde.

Deze aanpak kan wel profiteren van de DBMS functionaliteit, maar heeft toch nog enkele niet te verwaarlozen nadelen:

- Schending van het data-onafhankelijks principe. Om de data via SQL te analyseren is dus een grondige kennis van de interne representatie nodig. En bij een wijziging van de data-structuur moeten er mogelijk vele reeds ingevoerde tuppels en queries aangepast worden.
- De slechte performantie in verband met data opslag, per geometry zijn er vaak meerdere tuppels in een relatie nodig.
- De twee vorige nadelen zorgen voor een slechte gebruiksvriendelijkheid.
- Het definiëren van nieuwe (ruimtelijke) types wordt daardoor ook bemoeilijkt.
- Het onvermogen om meetkundige queries, zoals punt queries of aangrenzendheid, efficiënt uit te voeren (dat wil zeggen zonder de hele relatie te moeten doorzoeken)

De laatste tekortkoming zou verholpen kunnen worden door middel van indexen, ware het niet dat deze in een relationele database weinig of niet helpen op meer-dimensionale gegevens. Dit komt omdat men de tuppels niet tegelijk op twee waardes, de $x$ en de $y$ coördinaat, *gesorteerd* kan bijhouden in het secundair geheugen (hard disk). Als er dus, via de index van de niet-gesorteerde coördinaat, een reeks tuppels opgehaald moet worden, zullen de tuppels verspreid liggen op de hard disk en zal het ophalen hiervan niet efficiënt gaan.

## 3.3 Een relationele DBMS uitbreiden

Het gebruik van een relationele DBMS was al een stap in de goede richting, alleen moesten de beperkingen van de huidige types nog overwonnen worden (hoofdstuk 1 van [25]). Daarom was men tot de conclusie gekomen dat er nieuwe (spatial) types moesten komen die alle (of toch de belangrijkste) nadelen wegwerken. De nieuwe types moeten in staat zijn om een geometry

in één tuppel op te slaan, tegelijkertijd het data-onafhankelijkheids principe respecteren en een efficiënte fysische implementatie hebben. Dit kan worden bereikt door middel van Abstracte Data Types (ADT's) (zie paragraaf 4.1 en 4.2). Omdat er nieuwe types zijn moet de ondervragingstaal SQL ook uitgebreid worden, dit houdt in dat er nieuwe operaties op types en ook nieuwe query-optimalisaties voorzien moeten worden. Omdat deze types nu meer-dimensionale gegevens moeten ondersteunen, kunnen sommige reeds bestaande technieken, zoals B-trees voor de index, niet hergebruikt worden en zullen dus nieuwe structuren uitgevonden moeten worden. Ook operaties die sterk afhankelijk zijn van indexen moeten herwerkt worden, de belangrijkste hiervan is de *join* operatie. Naar dit soort DBMS wordt verwezen als Object-Relationele DBMS.

Deze object-relationele DBMS met een spatial extensie wordt besproken in hoofdstuk 4.

## 3.4 Object-georiënteerde DBMS

Een andere oplossing voor de problemen van relationele DBMS is ervoor te zorgen dat het DBMS met willekeurige *objecten* overweg kan (hoofdstuk 3 van [25]). In deze objecten kunnen dan de geometrieën in hun geheel opgeslagen worden, maar worden ook gebruikt om de attributen in op te slaan. Dit gebeurt door object-georiënteerde programmeertalen en relationele DBMS samen te voegen. Het resultaat hiervan is extra modelleer-kracht, uitbreidbaarheid van de systemen, hergebruik van code en gemakkelijk onderhoud van programma's.

De volgende verschillen zijn er met relationele DBMS: Een object heeft tijdens zijn ganse leven een unieke *object identifier* (OID) in plaats van een *primary key* waarmee naar hem verwezen kan worden. Deze OID wordt door het systeem toegekend en hoeft dus niet apart bijgehouden te worden. Een ander verschil is het gebruik van *classes* om geometrieën op te slaan in plaats van de ad hoc implementatie in relationele databases. Het type van een object kan nog altijd een basis- of *atomic* type zijn, zoals string of integer, maar kan ook een bepaalde klasse of zelfs een referentie/pointer naar een ander object zijn. Deze referentie gebeurt aan de hand van de OID van een object. Een klasse komt overeen met de implementatie van een ADT in object-relationele DBMS. Het zorgt voor de *encapsulation* van de geometrieen, dit betekent dat de interne datastructuur ervan verborgen blijft en dat het object alleen ondervraagd kan worden via de methoden van de klasse.

Het gevolg hiervan is dat de interne structuur van de klasse of zijn methodes gewijzigd kan worden zonder dat de gebruiker daar iets van merkt. Een ander voordeel van klassen is (*multiple*) *inheritance*, hierbij kan een nieuwe klasse de componenten en methoden overerven van bestaande klassen, waarbij het eventueel deze methoden opnieuw kan definiëren door middel van *overriding*.

Omdat er slechts weinig object-georiënteerde DBMS zijn met spatial ondersteuning en omdat de meeste grote commerciële systemen object-relationele DBMS zijn, worden deze DBMS verder niet besproken in deze verhandeling. Voorbeelden van (niet spatial) object-georiënteerde DBMS zijn te vinden op [2], enkele hiervan zijn: Object Store [14], $O_2$ [13] en Ozone [15].

## 3.5 Overzichtstabel

In de onderstaande tabel worden de belangrijkste eigenschappen gegeven van de vier voorgestelde oplossingen om geografische data te verwerken. Hierbij staat LC voor de "Loosely coupled" benadering van paragraaf 3.1, RDBMS voor het gebruik van een relationele DBMS van paragraaf 3.2, ORDBMS voor de uitgebreide relationele DBMS van paragraaf 3.3 en OODBMS voor de Object-georiënteerde DBMS van paragraaf 3.4.

| Eigenschap | LC | RDBMS | ORDBMS | OODBMS |
|---|---|---|---|---|
| Eenvoudig 'logisch data model' | nee | nvt.[4] | ja | ja |
| Data onafhankelijks principe | nee | nee | ja | ja |
| DBMS functionaliteit[5] | nee | ja | ja | ja |
| Eenvoudig te bevragen | ja | nee | ja | ja[6] |
| Efficiënte topologische queries[7] | ja | nee | ja | ja |
| Efficiënte fysische representatie | ja | nee | ja | ja |

[4]Geografische objecten moeten door de gebruiker zelf gedefinieerd worden
[5]Querying, optimalisatie, recovery
[6]Dit moet wel met de programmeertaal via de API
[7]Als ondersteund door het fysisch model

# Hoofdstuk 4

# Spatial Databases

## 4.1 Abstracte DataTypes (ADT's)

Wegens het onvermogen van de relationele database types om geometrieën efficiënt op te slaan, heeft men ADT's geïntroduceerd (hoofdstuk 3 van [25]). Een ADT is te vergelijken met een *class*, *struct* of *record* in een programmeertaal en geeft een abstracte maar functionele definitie van een object. Dit wil zeggen dat het de interne structuur van het object verbergt voor de gebruiker en dat de enige manier om toegang te krijgen via een set van operaties kan. Deze operaties horen bij het ADT en vormen de interface tussen het object en de buitenwereld.

Het splitsen van het gebruik en de implementatie van een object wordt *encapsulation* genoemd, dit laat ons toe om een ondervragingstaal uit te breiden met geometrische functionaliteit onafhankelijk van de gebruikte implementatie. Het enige wat dus vast ligt is de interface van de ADT, hoe deze geïmplementeerd wordt hangt volledig af van de ontwikkelaar. Het ADT blijft hierdoor transparant voor de gebruiker.

## 4.2 Spatial ADT's

In deze paragraaf wordt ingegaan op de keuze van de ADT's om geometrieën voor te stellen. Wat de gevolgen zijn van de definities van de geometrieën en hun onderliggend model. Dan wordt een overzicht gegeven van mogelijke operaties van de ADT's. En tenslotte lichten we de semantiek hiervan toe met een voorbeeld. (Hoofdstuk 3 van [25]).

Het is niet gemakkelijk om te beslissen welke geometrieën we gaan voor-

stellen met de ADT's. Aan de ene kant is het aantrekkelijk om de types veel vrijheid te geven zodat ze in veel verschillende situaties gemodelleerd kunnen worden. Aan de andere kant staan dan weer de *constraints* op de types waaraan voldaan moet worden, anders kan men evengoed één type maken waarin alle soorten geometrieën passen. Voorbeelden van mogelijke types zijn weergegeven in figuren 2.7 en 2.12 (op pagina's 16 en 22). Als we bijvoorbeeld een linestring definiëren als twee met segmenten verbonden punten, dan kan de geometry in figuur 2.12(c) hiermee niet voorgesteld worden. Als we ze daarentegen definiëren als een set van linestrings, dan kunnen er ook niet verbonden linestrings mee voorgesteld worden en is dan equivalent aan een multilinestring.

Eenmaal de keuze van de types vast ligt, moet er nog beslist worden welk model gebruikt wordt om ze te implementeren. De keuze van het model heeft geen invloed op de functionaliteit van de ADT's, maar wel op de efficiëntie van de operaties ervan. Bijvoorbeeld als voor een polygon het vector model gebruikt wordt en een gebruiker wil weten welke andere polygonen er aan grenzen, dan moet voor iedere polygon in de relatie nagegaan worden of deze er aan grenst. Als het topologisch model echter gebruikt wordt, kunnen de aangrenzende polygonen onmiddellijk gevonden worden. Anderzijds gaat het toevoegen van polygonen aan de relatie wel sneller als het vector model gebruikt wordt.

De interface van een ADT wordt gevormd door zijn methodes, in het stuk dat nu volgt worden een aantal functies gegeven. Deze lijst van functies is niet uitputtend of minimaal bedoeld, maar om de uitgebreidheid ervan te illustreren. Hierbij groeperen we de operaties op het aantal en de types van hun parameters, er wordt ook aangenomen dat ten minste één van deze parameters een spatial type heeft.

*Unaire operaties met een Booleaans resultaat*: Hiermee wordt getest of een geometry aan een bepaalde eigenschap voldoet, bijvoorbeeld of een polygon convex of gesloten is.

*Unaire operaties met een Scalair resultaat*: Hiermee kan een bepaalde waarde gemeten worden, zoals de lengte, oppervlakte of omtrek van een geometry.

*Unaire operaties met een Ruimtelijk resultaat*: Deze groep bestaat uit drie onderverdelingen.

- De eerste zijn de *topologische transformaties* zoals rotaties, translaties,

scalering en symmetrie.

- Een andere categorie is het *transformeren* van een $d$-dimensionaal object in een object van hogere of lagere dimensie, een voorbeeld hiervan is de grens bepaling van een polygon.

- De laatste subgroep is *object extractie*, zoals het bepalen van de minimal bounding box/rectangle (*mbb*/*mbr*) van een object of een buffer rond een object.

*Binaire operaties met een Booleaans resultaat*: Deze operaties worden ook wel *Spatial predikaten* genoemd en vormen de basis voor ruimtelijke queries, in het bijzonder voor de *spatial selectie* en de *spatial join*. Meestal wordt de volgende onderverdeling gemaakt:

- *Topologische predikaten*, deze gaan na of er een topologische relatie (zie paragraaf 2.3.3.1 pagina 30) bestaat tussen twee geometrieën en worden ook gebruikt bij interactieve queries (zie paragraaf 2.3.4 pagina 32).

- *Richting predikaten*, zoals ligt een geometry boven of ten noorden van een andere geometry.

- En ten slotte *Metrische predikaten*, hierbij wordt meestal getest of de output van een binaire operatie met scalair resultaat kleiner, groter of gelijk is aan een bepaalde waarde.

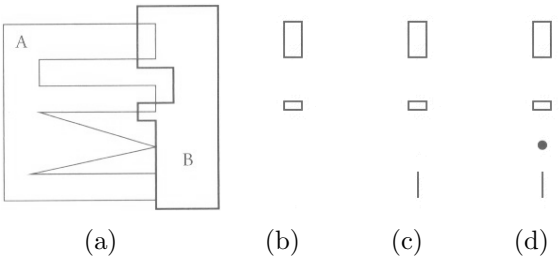
*Binaire operaties met een Scalair resultaat*: Het berekenen van de afstand tussen twee geometrieën is een typische operatie in deze categorie.

*Binaire operaties met een Ruimtelijk resultaat*: De meest gebruikte operaties in deze groep zijn de *set* operaties doorsnede, unie, verschil en symmetrisch verschil (XOR). De doorsnede is de basis voor het bepalen van de *overlay* van twee layers en de unie wordt gebruikt bij het bepalen van de *merge* van twee geometrieën in een layer.

*N-aire operaties met een Ruimtelijk resultaat*: In deze laatste groep zitten de overige operaties met een ruimtelijk resultaat, een voorbeeld hiervan is de verdeling in Thiessen veelhoeken, ook wel het opstellen van een Voronoi diagram genoemd (zie paragraaf 2.2.4.1 pagina 20).

Omdat een operatie een andere betekenis kan hebben afhankelijk van de geometry waarop hij toegepast wordt, gaan we nu ook even kijken naar de *semantiek* van operaties. Als voorbeeld nemen we de doorsnede operatie,

het resultaat hiervan kan een andere dimensie zijn als de dimensie van zijn argumenten. Dit wordt geïllustreerd in figuur 4.1.




Figuur 4.1: Semantiek van de doorsnede operatie (bron [25]).

Het normale geval, de geometrische doorsnede, wordt getoond in figuur 4.1(d). Zoals daar te zien is kan het resultaat een heterogene verzameling zijn van alle geometrische types. Omdat dit moeilijker is om te implementeren, heeft men de *genormaliseerde* doorsnede gedefinieerd. Hierbij is het type van het resultaat gelijk aan het type van de input. Voor dit voorbeeld wordt het resultaat van de genormaliseerde doorsnede getoond in figuur 4.1(b). Er zijn ook andere normalisaties mogelijk, zo kan bijvoorbeeld het resultaat van de doorsnede van een linestring en een polygon alleen maar een linestring als resultaat hebben of alleen maar een multipoint als resultaat. Iedere operatie kan zo genormaliseerd worden. Als de taal waarin de ADT's geprogrammeerd worden *overloading* van functies toelaat, vormt dit geen probleem om te implementeren.

## 4.3 Spatial indexen

Om efficiënt ruimtelijke queries uit te voeren, zijn er specifieke datastructuren nodig die snel en efficiënt toegang geven tot de geometrieën (hoofdstuk 6 van [25]). Deze datastructuren worden indexen genoemd en versnellen de toegang door niet alle geometrieën te onderzoeken, maar alleen degene die in het gebied liggen dat ons interesseert. Voor alfanumerieke gegevens kan een B-tree gebruikt worden, maar voor spatial data is deze niet toereikend.

Operaties die profiteren van een index zijn punt en window queries (zie paragraaf 2.3.4 pagina 32). Een ander voorbeeld is de spatial join, hierbij moeten objecten aan een spatial predikaat, zoals aangrenzendheid of omvatting, voldoen opdat ze samengevoegd zouden worden. (Zie paragraaf 4.4.2 voor een

bespreking van de spatial join.)

De algoritmes, die de spatial indexen gebruiken, om ruimtelijke gegevens op te vragen worden ook wel *spatial access methods* (SAMs) genoemd. Van deze algoritmes wordt verwacht dat ze logaritmisch —of beter— werken in verhouding met het aantal tuppels in de relatie en dat de indexen niet veel meer ruimte innemen dan het aantal *entries* die in de index zitten. Omdat SAMs voor alle geometrische types gebruikt moeten kunnen worden en deze geometrieën mogelijk complex kunnen zijn, wordt per geometry zijn omvattende *minimal bounding box* (*mbb*) bijgehouden. Een *mbb* kan gemakkelijk worden voorgesteld met behulp van een vierhoek, hiervoor zijn slechts twee coördinaat-paren nodig, één voor het punt links-onder en één voor het punt rechts-boven. In de index wordt dan de *mbb* van een geometry als *key* gebruikt, zo kan de kost van het uitvoeren van predikaten beperkt worden aangezien het evalueren van een predikaat op een vierhoek voor de meeste predikaten in constante tijd kan. In de index wordt dan bij iedere *mbb* verwezen naar zijn werkelijke geometry.

De SAMs werken in twee fasen, de *filter* stap en de *refinement* stap. In de eerste stap wordt de index gebruikt met de *mbb* van de geometrieën om het aantal mogelijke kandidaten dat aan het predikaat voldoet te beperken. In de tweede stap, de verfijnings fase, worden alle kandidaat geometrieën uit het resultaat van de eerste stap sequentieel gescand en geëvalueerd volgens het opgegeven predikaat. Omdat de evaluatie in de tweede stap met de geometrieën zelf gebeurt is deze meestal zeer kostbaar, maar wordt slechts op een beperkt aantal tuppels uitgevoerd. Het is in de verfijnings fase dat er echt beslist wordt welke objecten voldoen aan het predikaat en welke –ten onrechte– door de filter stap geraakt zijn. Omdat deze stap toch altijd hetzelfde is, onafhankelijk van de gebruikte index, wordt in de rest van de paragraaf alleen de filter stap besproken.

Zoals vermeld in het begin van deze paragraaf voldoet een B-tree of een B+tree (een geoptimaliseerde variant van de B-tree) niet voor spatial data. Een B+tree is een gebalanceerde boom met in iedere node een aantal (key, pointer)-entries. Voor interne nodes wijst de pointer naar een andere node waarvan de waardes van de key kleiner zijn (of groter) dan zijn eigen key. Bij de eind nodes wijst de pointer naar de locatie op de harde schijf van het tuppel dat met de key overeen komt. Een B+tree maakt dus gebruik van het feit dat er een *totale orde* ligt op de *key*, meestal is dit de lexicografische orde. Deze orde laat toe om efficiënt waardes op te vragen en om interval queries te beantwoorden. Voor punten met twee-dimensionale coördinaten

werkt deze aanpak niet omdat hiervoor geen 'goede' totale orde bestaat. Van een goede totale orde wordt verwacht dat twee punten die dicht bij elkaar liggen in het vlak, ook dicht bij elkaar liggen nadat ze geordend zijn. Een voorbeeld van een 'slechte' orde is de opeenvolging van de $x$ en $y$ coördinaat. Hierbij is het mogelijk dat de punten (1,1) en (2,1), die naast elkaar liggen in het vlak, toch ver van elkaar verwijderd zijn in de orde omdat er nog vele andere punten zijn van de vorm (1,$y$) waarbij $y$ groter is dan 1. Deze limitatie in verband met totale ordening geldt ook voor *hashing*, zodat ook deze methode niet gebruikt kan worden. In de rest van de paragraaf worden twee groepen van ruimtelijke indexen besproken die dit probleem oplossen, namelijk de *grid* en de *R-tree*.

Omdat er bij deze indexen geen gebruik kan gemaakt worden van de orde van coördinaten, moet er iets anders gezocht worden. Bij *Space-driven indexen* wordt de ruimte in cellen verdeeld, onafhankelijk van de verdeling van de data en worden de geometrieën in deze cellen gemapt. Tot deze aanpak horen de *grid files* en *lineaire structuren*. Bij *Data-driven indexen* gaat men juist indelen volgens de geometriën. Tot deze aanpak horen de *R-Trees*. Beide aanpakken presteren redelijk tot goed bij een gelijkmatige verdeling van de data, maar presteren slecht als dit niet het geval is. Dit in tegenstelling tot een B+tree voor alfanumerieke data die ook in *worst-case* scenario optimaal presteert.

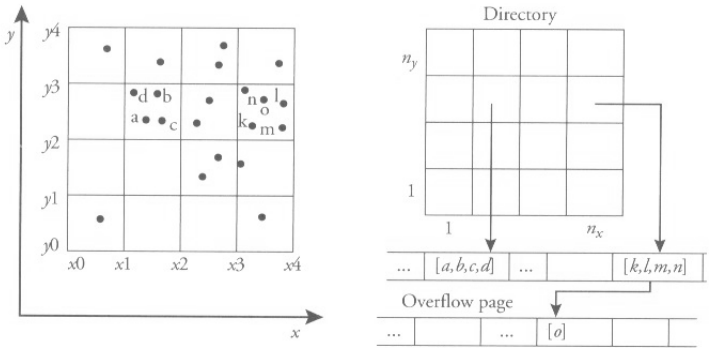
### 4.3.1 Space-driven indexen

Bij deze aanpak wordt de ruimte in cellen gesplitst onafhankelijk van de verdeling van de data. Eerst kijken we naar de *fixed grid* en de *grid file* voor het indexeren van punten, hierna beschouwen we deze twee indexen voor rechthoeken. Dan bestuderen we de *lineaire quadtree* en tenslotte kijken we naar de *z-ordening tree*.

#### 4.3.1.1 Fixed grid & Grid file

De eenvoudigste variant van space-driven indexen is de *fixed grid*, hierbij wordt de ruimte in cellen van gelijke grootte gedeeld waarbij iedere cel naar een *page* op de harde schijf verwijst. Om te bepalen in welke cel een punt zit of moet komen, wordt door middel van gehele deling zijn $x$ waarde gedeeld door de breedte en zijn $y$ waarde door de hoogte van een cel. Als deze waarden nog met één vermeerderd worden bekomt men de gewenste celcoordinaten. Als, bij het toevoegen van een punt, de *page* waar de cel naar verwijst vol is, wordt een *overflow page* toegevoegd zodat deze cel nu naar

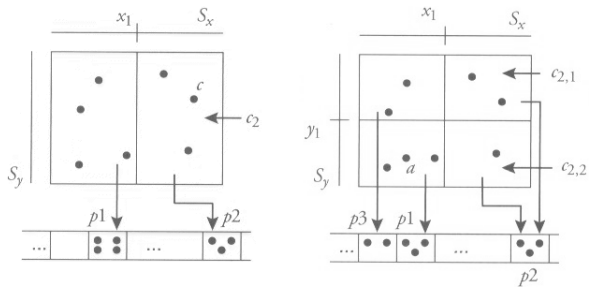
een lijst van *pages* verwijst. Dit wordt geïllustreerd in figuur 4.2, hierbij wordt punt *o* toegevoegd en passen er maximaal vier entries per *page*.




Figuur 4.2: Overflow in een Fixed grid (bron [25]).

Over het algemeen presteert deze index redelijk goed voor het toevoegen en opvragen van punten, maar zijn er veel lege cellen en cellen met veel overflow *pages*, in het *worst-case* scenario liggen alle punten in één cel.

Een iets betere index is de *grid file*, net zoals bij fixed grid verwijst iedere cel naar een *page* op de harde schijf. Maar als hier een cel vol is, krijgt hij geen overflow *page* maar wordt de cel in twee gelijke delen gesplitst. De keuze, volgens welke as er gesplitst wordt, is willekeurig. Om te bepalen waar een punt ligt of moet worden ingevoegd, moeten wel de afstanden bijgehouden worden waarop er gesplitst werd. Een andere verbetering is dat verschillende cellen nu naar één *page* kunnen wijzen, zo worden er geen *pages* op de harde schijf verspild als er cellen leeg zijn. Figuur 4.3 illustreert het

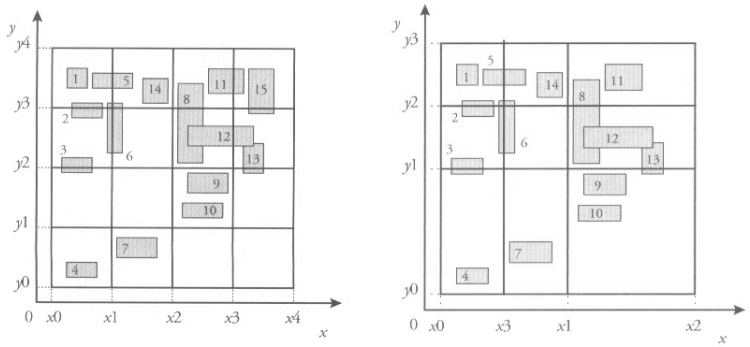



(a) Beginsituatie (b) Na toevoeging

Figuur 4.3: Toevoeging van punt *a* in een Grid file (bron [25]).

toevoegen van punten in een grid file. Deze index werkt de belangrijkste beperkingen van fixed grid weg. Een punt kan nu altijd na het lezen van twee *pages* gevonden worden (één voor de index en één voor het punt). Hierbij komt dat de structuur dynamisch is en dat de ruimte beter ingedeeld wordt.

Het indexeren van rechthoeken, in het bijzonder *mbb*'s, verloopt op analoge wijze, alleen kunnen de rechthoeken nu in meerdere cellen voorkomen. Dit heeft als gevolg dat de cellen sneller gesplitst moeten worden en dat de efficiëntie dus afneemt. Een voorbeeld van geïndexeerde rechthoeken is te zien in figuur 4.4.




(a) Fixed grid (b) Grid file

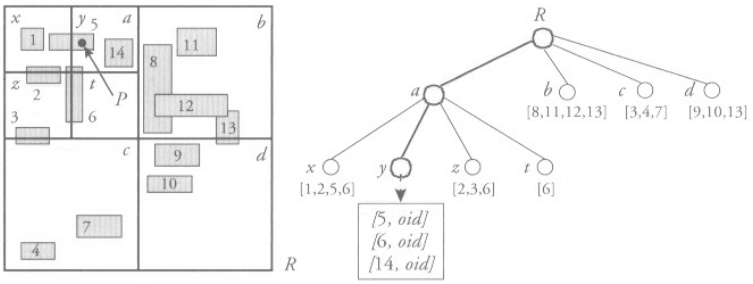
Figuur 4.4: Fixed grid versus Grid file (bron [25]).

Omdat rechthoeken in meerdere cellen kunnen voorkomen, zullen er duplicaten in het resultaat van de filter stap zitten. Als er veel cellen gesplitst moesten worden, waardoor de grootte van de cellen afneemt, ontstaan er alleen maar meer duplicaten. De kost om deze te verwijderen kan, als het resultaat groot is, hoog oplopen. Als laatste punt merken we op dat, voor zeer grote data sets, de index niet meer in het RAM past. Als dit gebeurt kan een deel ervan op harde schijf opgeslagen worden, maar dit bemoeilijkt het beheer en vertraagt het gebruikt van de index.

#### 4.3.1.2 Lineaire quadtree

Deze en volgende indexen maken gebruik van een *quadtree*. Alvorens de indexen toe te lichten, leggen we uit hoe een quadtree werkt. Hierbij wordt de ruimte recursief in vier kwadranten opgesplitst, dit gebeurt totdat het aantal rechthoeken dat in een kwadrant ligt kleiner is dan de capaciteit van een *page*. Dan wordt een boomstructuur opgebouwd waarbij iedere interne node

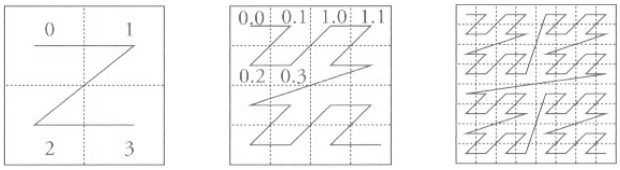
vier kinderen heeft en waarbij iedere node naar een *page* verwijst. Net zoals bij een grid kunnen rechthoeken in meerdere *pages* voorkomen en komen er dus duplicaten voor in de quadtree. Om de locatie van een punt te weten te komen, moet men in de root beginnen. Van daar moet men een pad door de boom volgen door in iedere node het kwadrant te kiezen waarin het punt ligt. Uiteindelijk komt men dan bij een eind node met de verwijzing naar het punt. Een voorbeeld van een punt-query in een quadtree is te zien in figuur 4.5.




Figuur 4.5: Punt query in een Quadtree (bron [25]).

Een quadtree kan als index gebruikt worden, maar alleen als deze volledig in het RAM wordt bijgehouden. De reden hiervoor is dat iedere node een *page* op de harde schijf voorstelt en dat het aantal entries per node slechts vier is. Hiermee is een *page* lang niet gevuld en wordt er dus geheugen verspild. Een gevolg hiervan is dat de boom veel te diep is en dat bij het navigeren door de boom, in het slechtste geval, alle *pages* van de harde schijf geladen moeten worden. Een index die deze nadelen wegwerkt is de lineaire quadtree.

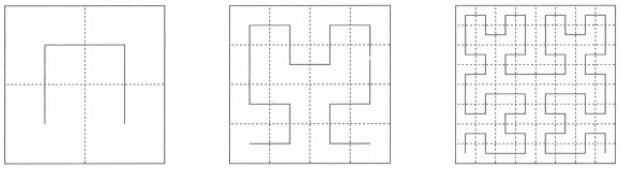
Om een lineaire quadtree te maken moeten de kwadranten van de quadtree gelabeld zijn met een nummer. Deze nummering moet bovendien totaal geordend zijn en gedeeltelijk de *nabijheid* bewaren, dat wil zeggen dat twee cellen die dicht bij elkaar liggen *meestal* ook een label hebben dat dicht bijeen ligt. We kunnen zo een nummering bekomen door middel van *space-filling curves*. Een eerste variant hiervan is *z-order*, ook wel *z-ordering* of *Morton code* genoemd. Hierbij krijgt iedere node een label bestaande uit het alfabet {0,1,2,3}. De root krijgt als label de lege string, een niveau lager wordt met één nummer gelabeld, het niveau daaronder met twee nummers, enzovoorts. Welk getal in het label komt hangt af van de positie van het kwadrant, het NW-kwadrant (links-boven) krijgt een '0', NE een '1', SW een '2' en SE een '3'. Een voorbeeld van deze nummering is te zien in figuur 4.6.




(a) Niveau 1 (b) Niveau 2 (c) Niveau 3

Figuur 4.6: z-ordering (bron [25]).

Een andere variant van de space-filling curves is de *Hilbert curve*, deze curve vormt nu geen Z, maar een Π, zie figuur 4.7. In tegenstelling tot z-order bestaat de Hilbert curve uit segmenten van gelijke lengte zodat nooit ver *gesprongen* moet worden naar een volgende cel. Toch zijn er ook hier nog gevallen waar twee objecten dicht bij elkaar liggen maar toch ver vandaan volgens hun label.

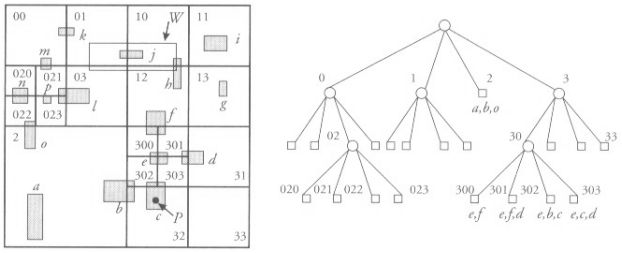


(a) Niveau 1 (b) Niveau 2 (c) Niveau 3

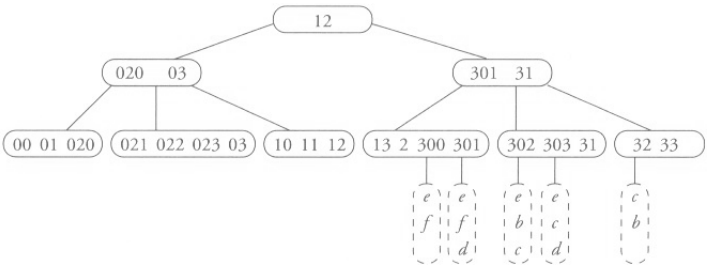
Figuur 4.7: Hilbert curve (bron [25]).

Nu de quadtree gelabeld is, is de enige stap die we van een lineaire quadtree verwijderd zijn, het indexeren van de (label, mbb_ID) koppels in een B+tree. Hierdoor wordt de originele quadtree dynamisch gecomprimeerd, dat wil zeggen dat de B+tree consistent blijft tijdens het toevoegen en verwijderen van rechthoeken. Een voorbeeld van deze compressie is te zien in figuur 4.8.

Door het gebruik van een B+tree kan het opvragen van een punt altijd gebeuren door maximaal $d + 1$ *pages* in te lezen. Hierbij is $d$ gelijk aan de diepte van de B+tree en bevindt het punt zich op de extra *page*. Voor een window query moet er twee keer een punt query gebeuren voor de eindpunten, dit gaat door maximaal $2d$ *pages* in te lezen. Daarna moeten er nog zoveel *pages* als nodig gelezen worden om het volledige interval hiertussen in te lezen. Omdat sommige objecten die dicht bij elkaar liggen toch labels kunnen hebben die ver van elkaar liggen, kan het zijn dat er overbodige ob-




(a) Quadtree




(b) Lineaire quadtree

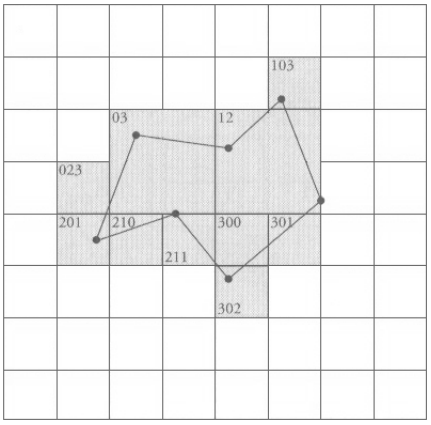
Figuur 4.8: Compressie van een Quadtree (bron [25]).

jecten ingelezen worden. Er bestaan algoritmen die dit vermijden, maar de prijs die hiervoor betaald moet worden is een zeer complexe implementatie.

#### 4.3.1.3 z-Ordening tree

Deze laatste space-driven index verschilt met de voorgaande indexen omdat hier niet met de *mbb* van de geometrieën gewerkt wordt, maar rechtstreeks met de geometry zelf. Deze wordt ontbonden in een quadtree met een maximale diepte $d$. Het algoritme gaat als volgt: Gegeven een geometry $g$ en een kwadrant $q$, kijk of $g$ overlapt met *alle* vier deelkwadranten van $q$. Als dit zo is, associeer het label van $q$ dan met $g$. Zo niet, herhaal de vorige stap dan voor de deelkwadranten waarmee $g$ overlapt. De recursie stopt als de maximale diepte $d$ bereikt is, het kwadrant is dan *minimaal*. In figuur 4.9 wordt de ontbinding van een geometry geïllustreerd, hierbij zijn alle grijze kwadranten minimaal, behalve die met de labels '03' en '12'.

Nadat alle geometrieën ontbonden zijn, moeten de (label, geo_ID) koppels in een B+tree geïndexeerd worden. Hierbij zijn vele duplicaten mogelijk, er kunnen meerdere geometrieën met hetzelfde label zijn en er kunnen geometrieën zijn die op verschillende niveaus met dezelfde cel overlappen. Voor



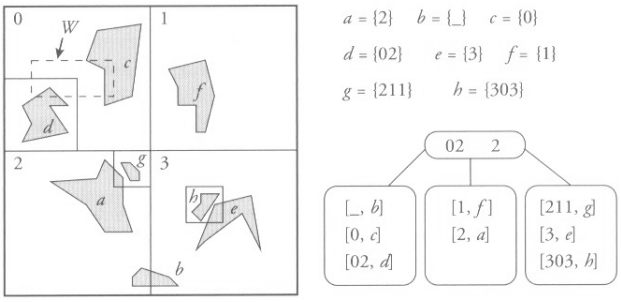

Figuur 4.9: Ontbinding volgens z-Ordering (bron [25]).

dezelfde data zitten er meestal meer entries in de B+tree van z-ordenig dan bij een lineaire quadtree. Hiertegenover staat dat er bij de lineaire quadtree nog een extra *page* opgehaald moet worden om aan het $ID$ van de geometry te komen, terwijl dit bij z-ordening rechtstreeks in de B+tree zit.

Een raster variant op deze index gaat onmiddellijk op de maximale diepte $d$ kijken welke geometrieën overlappen met de kwadranten. Hierdoor komen er nog meer entries in de B+tree, maar zijn de algoritmes voor de punt en window query veel eenvoudiger.

Een laatste variant op de z-ordening tree doet juist het tegenovergestelde, een geometry krijgt het label toegekend van het kleinste kwadrant dat de geometry volledig omvat. Hierdoor zijn er geen duplicaten in de B+tree en is deze ook veel minder diep. Een nadeel hiervan is dat de verhouding van de grootte van een geometry niet altijd overeen komt met de grootte van zijn kwadrant. Zo zal de geometry van $b$ in figuur 4.10 bij iedere query in het resultaat van de filter stap zitten en zal er in de *refinement* stap nog uitgehaald moeten worden. Een optimalisatie hierbij is om de *mbb* van iedere geometry toch bij te houden zodat het predikaat in de *refinement* stap eerst op de *mbb* getest kan worden.

Nog enkele opmerkingen over de space-driven indexen: Het gebruik van B+trees zorgt voor een uitstekende performantie, zelfs in worst-case. De mapping van twee-dimensionale data naar één-dimensionale data gooit echter roet in het eten. Niet alle objecten, maar wel de meeste, die dicht bij elkaar liggen voor de mapping, liggen erna ook nog dicht bij elkaar. Dit zorgt




Figuur 4.10: z-Ordering zonder duplicaten (bron [25]).

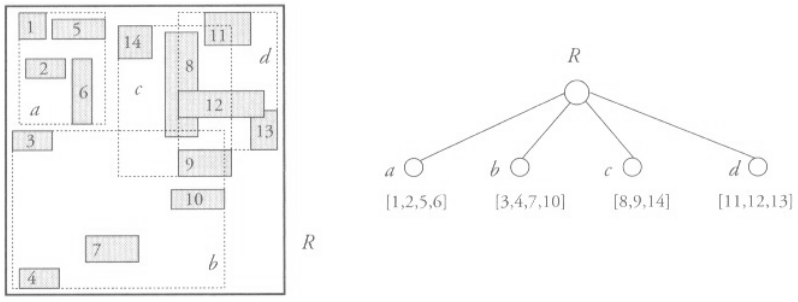
voor onefficiënte uitvoering van window queries. Maar wat erger is, door de mapping ontstaan, in bijna alle gevallen, veel duplicaten. Hierdoor neemt de grootte van de index toe en moeten deze duplicaten achteraf weer verwijderd worden wat mogelijk een zeer kostbare operatie is.

### 4.3.2 Data-driven indexen

Bij deze aanpak wordt de ruimte in cellen gesplitst volgens de verdeling van de data. Aan de basis van deze groep indexen ligt de *R-tree*; net zoals bij de B-tree wordt hier een gebalanceerde boomstructuur gebruikt. Eerst beschouwen we de originele R-tree waarna we enkele variaties beschouwen, enkele hiervan zijn de *R*tree*, *R-tree Packing* en de *R+tree*.

#### 4.3.2.1 R-tree

Bij de R-tree wordt, in tegenstelling tot bij de B-tree, niet met een één-dimensionale key gewerkt, maar met een twee-dimensionale *directory rectangle* (*dr*), dit is de *mbb* rond al de geometrieën in zijn kind nodes. De koppels die in de node van de R-tree gebruikt worden hebben de vorm (*mbb*,$ID$). Voor interne nodes is de *mbb* gelijk aan de *dr* en het verwijst het $ID$ naar een kind node. In eind nodes zit de *mbb* van de geometry en zijn $ID$. De overige voorwaarden zijn hetzelfde als bij een B-tree: Alle nodes, behalve de root, moeten minstens voor de helft vol zitten. En alle eind nodes moeten op dezelfde diepte liggen. In figuur 4.11 wordt een R-tree geïllustreerd. Merk hierbij op dat ieder object maar in één eind node voorkomt en dat verschillende *dr*'s mogen overlappen. Voor het bepalen van een geometry wordt het pad (of de paden) doorheen de boom gevolgd waarvoor de *dr* overlapt met de geometry. De lengte van een pad is gelijk aan de diepte van de boom, deze is logaritmisch in het aantal geïndexeerde objecten. Maar omdat *dr*'s




Figuur 4.11: R-tree (bron [25]).

kunnen overlappen moeten, in het slechtste geval waarbij een geometry met alle *dr*'s overlapt, alle paden doorlopen worden. Dit staat in contrast met de space-driven indexen waarbij ieder punt gevonden kan worden door één pad.

Omdat de R-tree data-driven is kan het zijn dat er, bij het toevoegen of verwijderen van (de *mbb* van) geometrieën, *dr*'s veranderd, gesplitst of samengevoegd moeten worden. Als er meerdere *dr*'s geschikt zijn voor de toevoeging van een object, dan wordt die *dr* genomen die het minste vergroot moet worden. Deze aanpassing kan in het slechtste geval omhoog gepropageerd worden tot de root. In het geval van een splitsing probeert men zowel zo klein mogelijke *dr*'s te krijgen als een zo klein mogelijke overlap tussen de *dr*'s. Omdat dit tegenstrijdige criteria zijn, minimaliseert men bij een R-tree meestal de oppervlakte van de *dr*'s.

Omdat de optimale splitsing kan niet snel genoeg bepaald worden via een *brute-force* methode (exponentieel in het aantal koppels in een node), wordt met heuristieken gewerkt. Het *quadratic split* algoritme is hier een voorbeeld van, het initialiseert eerst twee groepen met de twee entries die het verste van elkaar liggen en voegt dan rest toe aan deze groepen. Omdat het toevoegen van de overige entries in willekeurige volgorde gebeurt, kan het zijn dat de verdeling ervan niet optimaal is. Varianten op de R-tree proberen een betere verdeling van de geometrieën te bekomen.

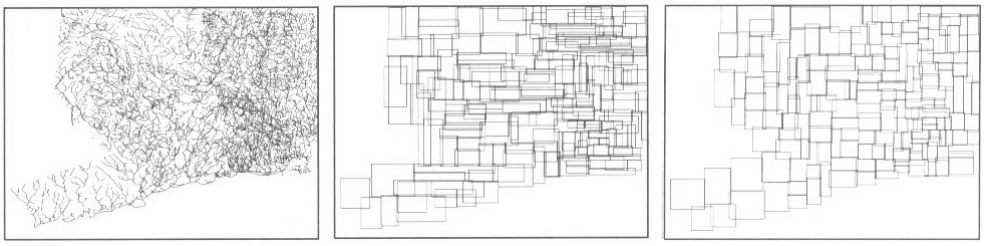
#### 4.3.2.2 R*tree

Bij deze variant op de R-tree ligt de nadruk niet alleen op het minimaliseren van de oppervlakte. Deze index heeft als doel het beperken van: de overlap tussen nodes, de oppervlakte van een node en de omtrek van de *dr* van een node. Ook deze criteria zijn tegenstrijdig, daarom probeert men bewerkin-

gen, zoals splitsen en toevoegen, te verbeteren met nieuwe heuristieken.

De heuristiek voor het splitsen van een node is het kiezen van de beste verdeling bij een splitsing volgens een *as*. Door de *mbb*'s te sorteren op hun minimum en maximum coördinaten kan de beste as bepaald worden. De beste verdeling is die wat voor een minimale overlap zorgt. Als er zo meerdere verdelingen zijn, dan wordt die gekozen met de kleinste oppervlakte.

Een andere heuristiek van de R*tree is die voor het toevoegen van een object. Hierbij wordt het splitsen van een node, wanneer deze overloopt, vermeden. Dit kan bereikt worden door een object, dat al aanwezig is maar dat voor een slechte ordening in de *dr* zorgt, eerst te verwijderen en dan opnieuw toe te voegen. Hierna wordt het nieuwe object toegevoegd. Indien het verwijderde object in een andere node dan zijn oorspronkelijke node terecht gekomen is, kan kan het nieuwe object daarin toegevoegd worden zonder een splitsing. Indien het verwijderde object terug in dezelfde node zit zal deze bij de toevoeging van het nieuwe object overlopen en moet deze node toch gesplitst worden. In figuur 4.12 is een vergelijking te zien van een R-tree met een R*tree voor het rivierennetwerk van Connecticut.



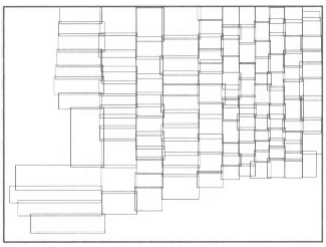
(a) Data set (b) R-tree (c) R*tree

Figuur 4.12: Voorbeeld van R-tree versus R*tree (bron [25]).

#### 4.3.2.3 R-tree packing

Tot nu toe hebben we de R-tree voor *dynamische* datasets gebruikt, dit houdt in dat we de index moeten kunnen updaten. Deze variant van de R-tree dient voor *statische* datasets, hierbij wordt eerst alle data ingeladen voordat de index hierop geconstrueerd wordt. Voor het aanmaken van de index wordt het *Sort-Tile-Recursive* (STR) algoritme gebruikt. Het idee hierachter is om de data te sorteren en dan in te delen volgens een dambord-patroon. Eerst worden de eind nodes ingedeeld, daarna worden deze recursief verder

ingedeeld totdat er nog maar één node overblijft. Bij deze index zijn (bijna) alle nodes volledig gevuld met entries om maximaal gebruik te maken van het geheugen. Het nadeel hiervan is wel dat er al bij het toevoegen van één object gesplitst moet worden. In figuur 4.13 is een *packed* R-tree te zien over dezelfde dataset als in figuur 4.12(a).



Figuur 4.13: Voorbeeld van R-tree Packing (bron [25]).

#### 4.3.2.4 R+tree

De laatste variant op de R-tree zorgt voor een indeling van nodes zodat hun *dr*'s niet overlappen. Hierdoor is ieder punt nu het volgen van één pad door de boom te vinden. Het nadeel hiervan is wel dat er, net zoals bij de space-driven indexen, opnieuw duplicaten mogelijk zijn omdat de objecten in verschillende nodes kunnen voorkomen. Het gevolg hiervan, samen met het feit dat niet alle nodes voor de helft vol moeten zijn, is dat een R+tree aanzienlijk meer entries bevat en dus dieper is dan een gewone R-tree. Ook het opstellen en beheren van deze index is moeilijker, zo moet een rechthoek die met meerdere *dr*'s overlapt, voor iedere *dr* gesplitst en toegevoegd worden. In het geval dat een node vol is en gesplitst moet worden, zal deze splitsing ook naar *onder* gepropageerd moeten worden.

Hoewel deze index heel efficiënt punt queries kan uitvoeren, is dit voordeel bij window queries minder uitgesproken. Hierbij komt dat de duplicaten zorgen voor een grotere index en dat het verwijderen van de duplicaten mogelijk zeer kostbaar is.

## 4.4 Spatial algoritmes

Deze paragraaf bespreekt algoritmes op ruimtelijke gegevens in spatial databases (hoofdstuk 7 van [25]). Net zoals bij relationele databases het geval is, worden deze operaties meestal op zeer grote datasets uitgevoerd, zo groot

zelfs dat ze niet in het RAM-geheugen passen. We beginnen daarom met een voorbeeld van zo een algoritme, de *External Sort/Merge*. Dit algoritme is niet specifiek voor spatial data, maar het wordt wel gebruikt bij de *Spatial Join*, die we daarna bespreken. Dan geven we enkele optimalisaties voor de spatial join. Ten slotte bespreken we het *Query Execution Plan*, hierbij worden algoritmes en de toegang tot de data op een harde schijf samen geintegreerd.

In de rest van deze paragraaf veronderstellen we dat voor het uitvoeren van de algoritmes er gebruik gemaakt wordt van een computer met maar één processor en één harde schijf. Alle operaties moeten dus sequentieel gedaan worden en kunnen niet parallel werken. Een andere aanname is dat zowel het RAM als de harde schijf bestaan uit *pages* van dezelfde grootte. De grootte van de dataset wordt aangeduid met $n$ en de grootte van het beschikbare RAM met $m$ (beide uitgedrukt in *pages*). Een laatste veronderstelling is dat iedere toegang tot een *page* op de harde schijf een kost heeft van één of dit nu sequentieel gebeurt of random.

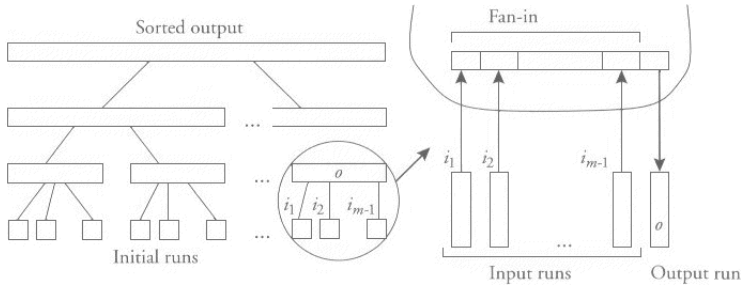
### 4.4.1 External sort & merge

Het sorteren van data is een vaak voorkomende operatie in queries, het wordt vaak gedaan als voorbereidende stap van een andere operatie. Een bekend sorteringsalgoritme is het *sort/merge join* algoritme, gebruikt om twee niet-geïndexeerde relaties te joinen. Omdat een join in lineaire tijd [tijdscomplexiteit O($n$)] gedaan kan worden als de relaties gesorteerd zijn, loont het de moeite om de relaties eerst te sorteren. Het sorteringsalgoritme dat hier gegeven wordt heeft een tijdscomplexiteit van O($n\log_m n$). Hierdoor bekomt men een join met tijdscomplexiteit O($n\log_m n$), dit is beter dan het alternatief, een *nested-loop join* die een tijdscomplexiteit heeft van O($n^2$).

Het algoritme dat hier besproken wordt is een variant op het sort/merge algoritme voor *main memory*. Hierbij worden de gegevens recursief opgedeeld totdat de elementen triviaal gesorteerd kunnen worden. In de tweede fase worden deze gegevens recursief en bottom-up samengevoegd. In het geval van *extern* geheugen werkt deze aanpak niet omdat de binaire boomstructuur niet gemakkelijk gemapt kan worden op *pages*. Daarom werd het algoritme zo veranderd dat in iedere node van de boom met één *page* gewerkt wordt in plaats van één tuppel.

In de eerste fase worden gesorteerde subsets gemaakt van de input dataset. Dit gebeurt door $m$ *pages* van de dataset in het RAM te laden, deze te

sorteren met het sorteringsalgoritme voor *main memory* en de gesorteerde subset terug naar de harde schijf te schrijven. Deze fase maakt, met mogelijke uitzondering van de laatste subset, optimaal gebruik van het geheugen. Nadat de ganse dataset per $m$ *pages* gesorteerd is, voegt de volgende fase ze weer samen. Om het geheugen optimaal te gebruiken worden deze sets niet per twee *pages* samengevoegd, zoals bij het *main memory* algoritme, maar per $m-1$ *pages*, waarbij de resterende *page* gebruikt wordt om het resultaat bij te houden. Als een *page* leeg raakt wordt de volgende *page* van de overeenkomstige subset geladen en als de *page* voor de output vol is wordt deze weggeschreven naar de harde schijf. Deze merge-fase wordt geïllustreerd in het rechterdeel van figuur 4.14.




Figuur 4.14: Mergen van gesorteerde subsets (bron [25]).

Omdat de input dataset groter kan zijn dan $m-1$ keer $m$ *pages*, moet de merge-fase net zolang herhaald worden met de output van de vorige uitvoering ervan, totdat alle data gesorteerd is. Dit wordt getoond in het linkerdeel van figuur 4.14. Het mergen op één niveau kan in lineaire tijd gebeuren, alle *pages* worden hier maar één keer gelezen en weggeschreven. Omdat er per keer $m-1$ *pages* samengevoegd worden, zijn er maar $O(\log_m n)$ niveaus nodig om de ganse dataset te sorteren. De totale tijdscomplexiteit is dus, zoals reeds vermeld, $O(n\log_m n)$, wat optimaal is (hoofdstukken 5 en 7 van [25]).

### 4.4.2 Spatial join

Om op twee relaties de spatial join uit te voeren, worden die tuppels gebruikt waarvan hun ruimtelijk attribuut aan een bepaald spatial predikaat voldoet. Voorbeelden van predikaten zijn, *overlap*, *contain* en andere *topologische*, *metrische* of *richting* predikaten (zie paragraaf 4.2). Wij concentreren ons hier op de *overlap* joins. We nemen hierbij ook nog aan dat voor iedere geometry ook zijn *mbb* bekend is. Deze benadering van de geometry maakt het mogelijk om snel te controleren of een geometry aan het predikaat voldoet of

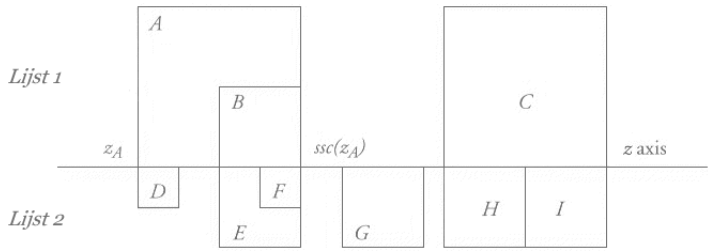
niet, deze eerste stap *filtert* de dataset. Hierna moet nog een *refinement* stap gebeuren om te controleren of de geometry zelf voldoet aan het predikaat. In deze paragraaf wordt alleen de filter stap besproken, de refinement stap komt in paragraaf 4.5 aan bod.

Eerst bespreken we de spatial join met behulp van een space-driven index, meer specifiek de z-Ordering tree. Daarna wordt gebruik gemaakt van een data-driven index, de R-tree. Ten slotte bespreken we het geval dat er geen index ligt op de relaties, hierbij gebruiken we een hash functie op één of beide relaties.

#### 4.4.2.1 z-Ordering join

Bij een z-ordering index wordt de geometry ontbonden volgens een quadtree waarvan de cellen hiërarchisch gelabeld zijn. We bespreken hier het normale geval waarbij de cellen van variabele grootte kunnen zijn, dit om het aantal duplicaten te beperken. We nemen ook aan dat er op de beide relaties een index ligt.

Als de eind nodes van een index sequentieel gescand worden, bekomt men een lijst $L$ van de vorm (label,$ID$). Deze lijst is gesorteerd op de label-waarde van de kwadranten. De rest van het join algoritme kan als volgt beschreven worden, merge de lijsten bekomen door de twee indexen te scannen. Een opmerking die hierbij nog gemaakt kan worden, is dat als twee cellen in elkaar vervat zijn, zoals $A$ en $B$ in het voorbeeld in figuur 4.15 en de kleinste joint met een object uit de andere lijst, dan joint de grootste ook met dit object. Twee cellen zijn in elkaar vervat, als het label van de ene een prefix is van de andere.




Figuur 4.15: Voorbeeld van z-ordering join (bron [25]).

Deze eigenschap wordt geïmplementeerd door twee stacks te gebruiken, één voor iedere lijst. Als een object tegengekomen wordt bij het doorlopen van de

twee lijsten moet het op de stack, horende bij de juiste lijst, gepusht worden. Het doorlopen van de lijsten kan gezien worden als het van links naar rechts bewegen van een lijn, loodrecht op de z-as. Hierbij horen de objecten boven de z-as tot de eerste lijst en die eronder tot de tweede lijst. Het tegenkomen van een object komt dan overeen met het bereiken van de linkerkant van een cel. Als de rechterkant van een cel bereikt wordt, moet het object terug van de stack gepopt worden en moet dit object met de objecten van de andere stack gejoind worden. De rechterkant van een cel wordt bereikt als een object met een groter label tegengekomen wordt, maar waarvan het geen prefix is. Zo is de output van de join van het voorbeeld in figuur 4.15 gelijk aan: (A,D), (A,E), (A,F), (B,E), (B,F), (C,H) en (C,I).

Als we het aantal *pages*, waarbij één *page* overeen komt met één eind node, van iedere index noteren met $n_1$ en $n_2$ en het aantal *pages* van het resultaat met $r$, dan is de totaal kost gelijk aan $n_1 + n_2 + r$. De z-ordering join leest namelijk iedere eind node van elke index één keer en schrijft iedere *page* van het resultaat terug naar de harde schijf. Voorwaarde hiervoor is wel dat de beide stacks in het RAM passen. Er is in alle situaties wel nog een extra kost van $O(r \log r)$ nodig voor het verwijderen van de duplicaten, geïntroduceerd door de z-ordering tree.

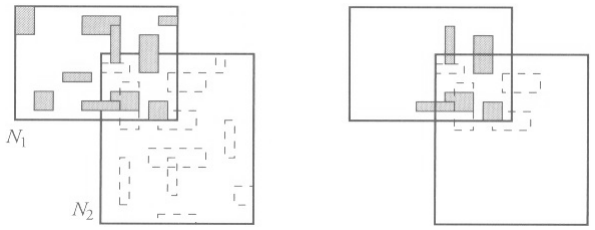
#### 4.4.2.2 R-tree join

Bij het joinen van twee relaties met behulp van R-trees, zorgt de index niet voor een orde op de objecten. De objecten komen er wel telkens maar één keer in voor. Ook hier wordt aangenomen dat er een index op de twee relaties ligt, dit mag een R-tree zijn, maar ook een variant hiervan. Omdat dit algoritme zeer veel objecten (*mbb*'s) met elkaar vergelijkt, wordt er ook nog een optimalisatie gegeven.

Het algoritme dat voor deze join gebruikt wordt heet het *synchronized tree traversel* (STT) algoritme. Het kernidee hierbij is, het *depth-first* doorlopen van de eind nodes van beide indexen en deze node per node te vergelijken. Dit wordt bereikt door te starten met de root van elke R-tree en in iedere stap een node $N_1$ van de ene index te vergelijken met een node $N_2$ van de andere index. Hierbij berekenen we koppels van overlappende entries ($e_1$,$e_2$) waarbij $e_1 \in N_1$ en $e_2 \in N_2$. Hierna voeren we recursief dezelfde stap uit op de deelbomen gevormd door $e_1$ en $e_2$. Als we bij de eind nodes uitkomen, beide op hetzelfde niveau, dan hebben we de $ID$'s van de objecten gevonden die met elkaar overlappen. Als de bomen niet even diep zijn, dus als we in de ene boom in een eind node $e$ zitten en de andere in een interne node $n$,

dan moet er een window query uitgevoerd worden op deelboom van de node $n$ met de *mbb* van node $e$ als argument.

Ondanks het feit dat dit algoritme goed scoort op het gebruik van de harde schijf, is het toch niet toereikend, dit komt omdat de CPU kost heel hoog is. De zwakke plek in het algoritme is de *geneste loop* die gebruikt wordt om de overlappende entries te bepalen, hierdoor worden soms entries gecontroleerd die totaal geen overlap hebben. Een optimalisatie die daarom voorgesteld werd is het *beperken van de zoek-ruimte*. Hierbij worden alleen die entries in $N_1$ en $N_2$ met elkaar vergeleken die in de doorsnede liggen van de *dr* van $N_1$ en de *dr* van $N_2$. Hiervoor worden beide nodes eerst gescand en worden de entries die in de doorsnede liggen gemarkeerd, daarna wordt het algoritme uitgevoerd op de gemarkeerde entries. Dit wordt geïllustreerd in figuur 4.16.



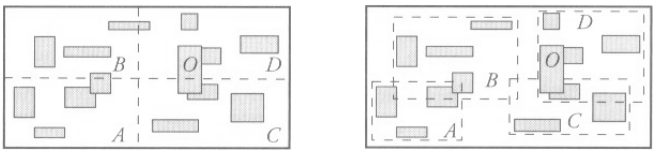

Figuur 4.16: Beperken van de zoek-ruimte (bron [25]).

#### 4.4.2.3 Spatial hash join

We frissen eerst even het algoritme van een gewone hash-join van twee relaties $R$ en $S$ op. Als één van de twee relaties, bijvoorbeeld $R$, in het RAM past, wordt hiervoor in één scan een hash tabel gemaakt. Dan wordt de andere relatie, $S$, gescand en vergeleken met de hash tabel. Als geen van beide relaties in het RAM past, moeten er een aantal buckets op de harde schijf bewaard worden. Hierbij moet er wel voor gezorgd worden dat één bucket wel nog in het RAM past.

Het hashen van de twee relaties met eenzelfde functie zorgt voor een één-op-één mapping tussen de relaties. Bij spatial relaties is zo een mapping echter moeilijk te bekomen zonder overlapping en duplicaten. Dit probleem kwam al bij de *space-driven* en de *data-driven* indexen aan bod in paragraaf 4.3. De eerste groep introduceert duplicaten in verschillende buckets terwijl

de andere groep voor overlapping tussen de buckets zorgt, dit wordt geïllustreerd in figuur 4.17.




(a) Duplicatie (b) Overlapping

Figuur 4.17: Alternatieven voor spatial hashing (bron [25]).

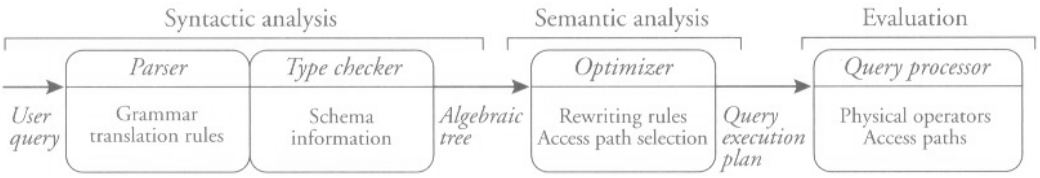
We geven hier een hash join algoritme voor de relaties $R$ en $S$ in het geval van overlapping van de buckets. De eerste stap doet een *initiële indeling* van $R$, ieder object wordt in één bucket gemapt, waarbij het domein van de bucket gelijk is aan de *mbb* van alle objecten die erin zitten. Deze indeling zorgt voor een situatie zoals te zien is in figuur 4.17(b). De voorwaarden waaraan deze indeling moet voldoen zijn: in iedere bucket moeten ongeveer evenveel objecten zitten, iedere bucket moet in het RAM passen en de *mbb*'s van de buckets moet minimaal zijn. In gebieden waar de objecten dicht bij elkaar liggen, zijn er dus meer buckets nodig dan in gebieden waar ze meer verspreid liggen.

De tweede stap doet een *tweede indeling*, maar nu van $S$. Hierbij wordt ieder object van $S$ gemapt op de initiële indeling van $R$. Dit kan aanleiding geven tot duplicaten en een ongelijkmatige verdeling over de buckets omdat de verdeling van $S$ totaal kan verschillen van die van $R$. Deze situatie is gelijkaardig aan die in figuur 4.17(a). Merk op dat objecten van $S$ die niet overeen komen met een bucket van $R$ al in deze stap gefilterd worden.

In de derde fase wordt de *join* van de relaties uitgevoerd. Dit wordt gedaan door twee overeenkomstige buckets van $R$ en $S$ te joinen. Omdat de buckets van $R$ in het RAM passen, kan dit simpelweg gedaan worden door de bucket van $S$ te scannen. Net zoals bij de R-tree join het geval is, is de laatste stap zeer CPU intensief en bestaan ook hiervoor optimalisaties, zoals het beperken van de zoek-ruimte. Merk ook op dat, omdat er maar voor één relatie duplicaten in de hash tabel zijn, de output van dit algoritme ook geen duplicaten bevat.

## 4.5 Query execution plan

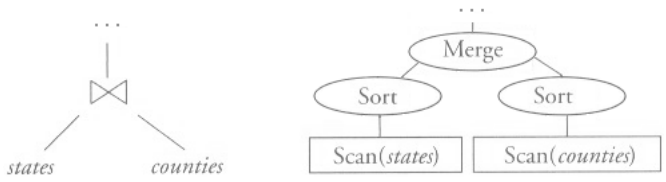
Tot nu toe hebben we ons maar met één spatial operatie bezig gehouden, namelijk de spatial join. In complexe queries kunnen natuurlijk meerdere spatial operaties voorkomen die samen verwerkt moeten worden. Het query execution plan (QEP) zorgt, onder andere, voor deze combinatie. Andere situaties waarmee het QEP rekening mee moet houden, zijn de refinement stap na het filteren en het uitvoeren van verschillende joins na elkaar, ook wel *multiway joins* genoemd. (Hoofdstuk 7 van [25]).




Figuur 4.18: Evaluatie van queries (bron [25]).

Wanneer een query uitgevoerd wordt door een DBMS, moet het de query omzetten naar een programma dat de data inleest en verwerkt tot het gewenste resultaat. De typische volgorde waarin dit gebeurt is te zien in figuur 4.18. Eerst wordt de query door de *parser* omgezet naar een equivalente relationele algebra expressie bestaande uit logische operatoren. De *type checker* controleert dan of de gebruikte relaties, attribuut namen en types wel bestaan en overeenkomen met het database schema. In de derde stap herschrijft de *optimizer* deze expressie om zo tot het query execution plan te komen. Hierbij worden de logische operatoren vervangen door fysische operatoren. De belangrijkste verschillen hiertussen zijn dat één logische operator meestal overeen komt met meerdere fysische operatoren, zoals te zien is in figuur 4.19 voor de join. En dat er voor sommige fysische operatoren, zoals het sorteren van een relatie, zelfs geen logische versie bestaat in de standaard relationele algebra. De *optimizer* moet dus kiezen welke fysische operatoren en in welke volgorde hij ze gebruikt in het QEP.

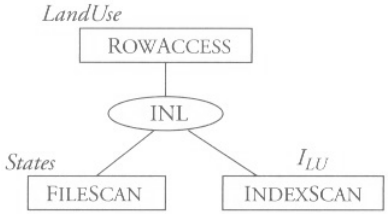
De laatste stap is het uitvoeren van het QEP door de *Query processor*. De rest van deze paragraaf zal over deze laatste stap gaan. Het QEP kan meestal gezien worden als een binaire boom die *bottom-up* wordt uitgevoerd, een voorbeeld hiervan is te zien in figuur 4.19. Een conventie die hierbij meestal gebruikt wordt, is dat toegang tot de data in rechthoeken weergegeven wordt en operaties in ovalen. In dit voorbeeld wordt een *scan* van de relatie gedaan,




(a) Algebra (b) QEP

Figuur 4.19: Logische versus fysische operatoren (bron [25]).

maar er kan evengoed gebruik gemaakt worden van een index op een relatie, zoals in figuur 4.20.




Figuur 4.20: QEP voor een Indexed Nested Loop Join (bron [25]).

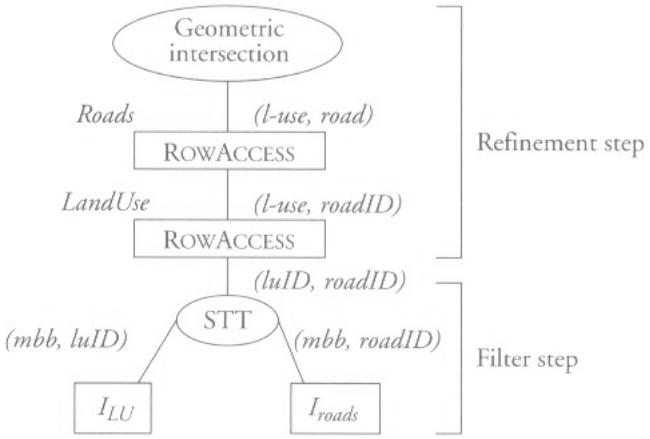
Hierbij stelt *INL* een *indexed nested loop* join voor. Merk hierbij ook op dat er nog een *rowacces* in de laatste stap moet gebeuren. De reden hiervoor is dat het resultaat van de join bestaat uit tuppels waarin een *record* zit van de relatie *States* en een $ID$ van de relatie *LandUse*. Dit $ID$ komt van $I_{LU}$, de index op *LandUse*, waarvan de geometrie nog opgezocht moet worden in de relatie *LandUse*. Een lijn wijst op de *dataflow* tussen twee operatoren, een naïeve aanpak hiervoor zou zijn om eerst het resultaat van één operatie te berekenen, dit resultaat tijdelijk op te slaan en tenslotte als input gebruiken voor de volgende operatie. Omdat het tijdelijke resultaat meestal te groot is om in het RAM opgeslagen te worden, wordt het dan op de harde schijf bijgehouden. Dit zorgt voor aanzienlijke overhead en neemt veel tijd in beslag.

Een betere aanpak is om niet te wachten totdat het volledige tussenresultaat klaar is, maar onmiddellijk bij het beschikbaar komen van het eerste tuppel ervan dit al door te geven aan de volgende operator. Deze methode kan geïmplementeerd worden met *iterators*. Hierbij heeft iedere operator een iterator tot zijn beschikking, deze kan onmiddellijk data leveren als hierom door een andere operatie gevraagd wordt en er data beschikbaar is. Dit zorgt

voor een *pipelined* uitvoering van meerdere operatoren. Deze methode is in de meeste relationele databases geïmplementeerd en kan relatief eenvoudig toegepast worden op ruimtelijke gegevens. Er zijn echter wel twee punten waarmee rekening moet gehouden worden: de refinement stap en de kost van spatial operaties.

### 4.5.1 Refinement stap

Veronderstel nu de *overlap join* tussen twee relaties *LandUse* en *Roads*, waarop in beide gevallen een index ligt, bijvoorbeeld een R-tree. Wanneer we deze join implementeren als een iterator met het *synchronized tree traversel* (STT) algoritme, dat in paragraaf 4.4.2.2 beschreven wordt, dan moet er nog twee keer een *rowacces* gedaan worden om de geometry te weten te komen. Dit wordt geïllustreerd in figuur 4.21.




Figuur 4.21: De filter en refinement stap (bron [25]).

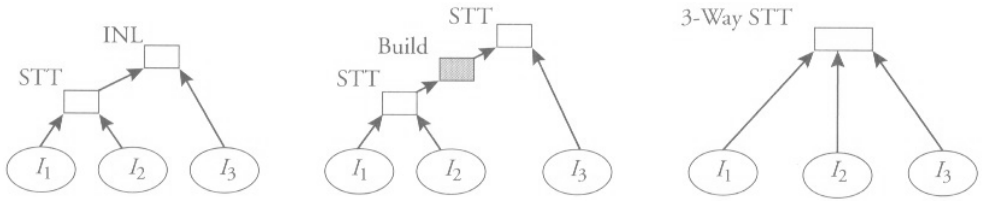
De eerste *rowacces* in de refinement stap haalt de *page* op waarop het record met *luID* staat en de tweede *rowacces* de *page* van *roadID*. Als beide ingeladen zijn in het RAM kan gecontroleerd worden of de geometrieën overlappen. Dit simpel algoritme heeft echter een belangrijke tekortkoming: het vraagt de tuppels random op, in *worst-case* moet hiervoor iedere keer een nieuwe *page* van de harde schijf gelezen worden. Hierbij komt ook nog dat bij een join meestal dezelfde tuppels meerdere keren worden opgevraagd. Door deze naïeve aanpak te gebruiken, wordt de refinement stap de duurste stap van ganse operatie.

Een verbetering hiervoor wordt geboden door de *segmented sort*. Hierbij

wordt er een buffer in het RAM gebruikt om een aantal output tuppels op te slaan. De tuppels worden dan gesorteerd volgens de volgorde op de harde schijf en worden dan in deze volgorde geladen. Tenslotte worden de tuppels verwerkt en wordt de buffer leeg gemaakt zodat een nieuwe iteratie kan beginnen. Een *page* wordt nu nog maar één keer per iteratie ingelezen, alhoewel deze wel nog in andere iteraties kan ingelezen worden, is deze oplossing toch al een serieuze verbetering. Een andere, meer drastische oplossing is om het volledige tussenresultaat op te slaan en in zijn geheel te sorteren. De kost hiervan hangt grotendeels af van de grootte van het resultaat, maar hierdoor wordt de *pipeline* wel onderbroken. Dit heeft als gevolg dat de *response tijd* van de query toeneemt en dat de gebruiker langer op de eerste resultaten moet wachten.

### 4.5.2 Multiway join

Het laatste punt dat we bij het QEP willen toelichten is de evaluatie van een sequentie van spatial joins. We beschouwen hier het geval: $R_1 \bowtie R_2 \bowtie R_3$, waarbij op iedere relatie een R-tree index ligt. Figuur 4.22 toont drie mogelijke QEP's voor deze *3-way join*.




(a) Left-deep tree (b) Build-and-match (c) 3-Way

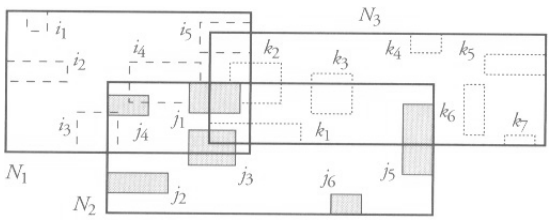
Figuur 4.22: Mogelijkheden voor een multi-way join (bron [25]).

Het eerste geval toont een *left-deep tree*. Hier kan voor iedere join de index van de rechtse relatie gebruikt worden, een uitzondering hierop is de join uiterst links waar beide relaties over een index beschikken. Omdat dit een volledig *pipelined* QEP is, heeft het een korte response tijd. Bovendien biedt het ook een maximale flexibiliteit omdat de refinement stap op ieder moment kan gedaan worden. Als er bijvoorbeeld verwacht wordt dat veel tuppels van het resultaat van de filter stap niet door de refinement stap raken, kan deze onmiddellijk na de filter stap uitgevoerd worden om zo het aantal tuppels te verminderen. Het gevolg hiervan is wel dat de tuppels groter zijn omdat ze dan een geometry bevatten in plaats van alleen maar een $ID$. Een nadeel

van deze aanpak is dat sommige tuppels en hun *pages* meerdere keren geladen moeten worden omdat ze door meerdere joins gebruikt worden. Een oplossing hiervoor wordt getoond in figuur 4.22(b).

De tweede mogelijkheid is het volgen van een *build-and-match* strategie. Hierbij wordt (een deel van) het resultaat van de eerste join tijdelijk opgeslagen en wordt hierop een index gecreëerd. Dit vertraagt natuurlijk de response tijd, maar omdat de volgende join sneller en efficiënter uitgevoerd kan worden, is de totale performantie toch beter dan bij de eerste methode. Een manier om de index te voorzien is door deze dynamisch aan te maken, hierbij moet tuppel voor tuppel toegevoegd worden, wat niet efficiënt is. Een betere manier is door de index op het tussenresultaat in zijn geheel te creëren, bijvoorbeeld door middel van R-tree packing (zie paragraaf 4.3.2.3).

Het laatste geval dat in figuur 4.22(c) te zien is, houdt in dat de drie indexen tegelijkertijd doorlopen worden. Het voordeel dat we hier bij hebben is dat de zoek-ruimte sterk beperkt kan worden. Wanneer we een *n-way* join doen, met $n > 3$, zal dit voordeel nog meer uitgesproken zijn. Een voorbeeld hiervan voor een *3-way* join is te zien in figuur 4.23.




Figuur 4.23: Beperken van de zoek-ruimte bij een 3-way join (bron [25]).

Omdat in de refinement stap voor een *multi-way* join er net zoveel keer een *rowacces* gedaan moet worden als er relaties zijn, kan de kost hiervan enorme proporties aannemen. Dit komt omdat eenzelfde $ID$ in meerdere tuppels kan voorkomen, waardoor zijn *page* dus meerdere keren geladen moet worden. Maar erger is, dat het *overlap predikaat* meerdere keren voor dezelfde object paren berekend moet worden. Stel dat de tuppels $(r_1, r_2, r_3)$ en $(r_1, r_2, r_3')$ door de filter stap geraakt zijn, dan moet hiervoor de overlap van $r_1$ en $r_2$ twee keer berekend worden tijdens de refinement stap. Bij de eerste twee gevallen kan deze situatie vermeden worden omdat daar, na iedere (*2-way*) join, de refinement stap al uitgevoerd kan worden. Het wel of niet uitvoeren hiervan kan afhangen van de grootte van het tussenresultaat, de kost van het spatial predikaat, de benadering van een geometry door zijn *mbb* en eventueel

nog andere parameters. Het voordeel dat deze twee methodes dan weer niet hebben is de efficiënte filter stap, zoals bij de derde methode. Het probleem bij *multi-way* joins is dus dat men nog geen methode gevonden heeft die in zowel de filter als de refinement stap efficiënt werkt.

# Hoofdstuk 5

# Commerciële Systemen

In de voorgaande hoofdstukken hebben we het gehad over: wat een GIS is, wat ermee gedaan kan worden en wat de rol van spatial database systemen hierin is. Hierbij hebben we verschillende modellen, structuren en algoritmen toegelicht die hierbij gebruikt *kunnen* worden. In de nu volgende hoofdstukken bekijken we drie commerciële toepassingen die ruimtelijke gegevens kunnen manipuleren. Per systeem worden de *gebruikte* modellen, structuren en algoritmen toegelicht. Ook wordt door middel van een *case study* nagegaan welke queries er per systeem mogelijk zijn.

Hieruit zal blijken dat niet alle software ontwikkelaars de 'beste' datastructuur of het meest efficiënte algoritme implementeren. De reden hiervoor is dat GIS applicaties en databases op lange-termijn ontwikkeld worden en gebonden zijn aan in het verleden gemaakte technische keuzes, hierdoor is het introduceren van nieuwe features meestal een kostbare zaak. Anderzijds ligt de nadruk bij DBMS nog altijd meer op alfanumerieke gegevens voor de bedrijfswereld, alhoewel er toch meer en meer aandacht is voor ruimtelijke gegevens. Meestal wordt er een compromis gevonden tussen de bestaande features en de in de vorige hoofdstukken voorgestelde technieken.

Dit hoofdstuk bespreekt de drie commerciële systemen *DB2 Spatial Extender* van $IBM$, *Oracle Spatial* van *Oracle* en *ArcGIS* van *ESRI*. De systemen van DB2 en Oracle zijn 'Object-Relationele DBMS' en het systeem van ESRI is een GIS toepassing. Van de twee spatial database systemen worden de datatypes, de index mogelijkheden en de functies besproken. Van de GIS toepassing wordt ook zijn datatype besproken maar daarna richten we onze aandacht op de drie *tools* waaruit dit systeem is samengesteld. Na deze bespreking wordt in het volgende hoofdstuk een case study uitgevoerd met behulp van deze drie commerciële systemen.

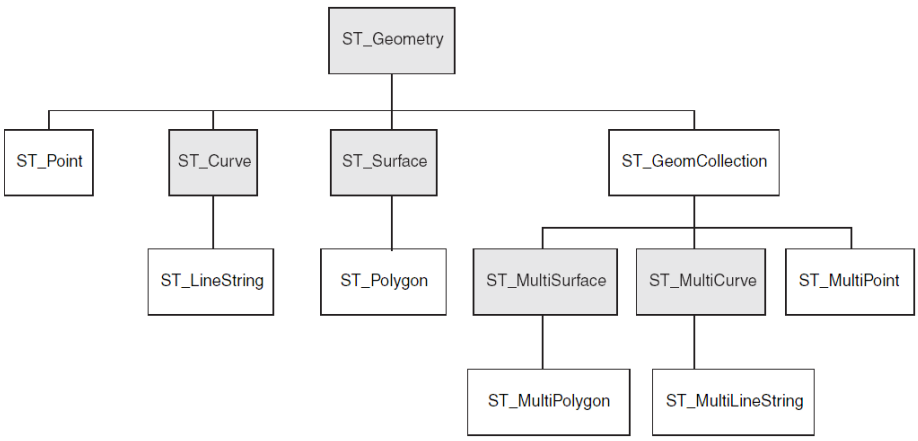
## 5.1 DB2 Spatial Extender

*DB2 Spatial Extender* breidt de functionaliteit van *IBM DB2 Universal Database* uit met spatial datatypes en functies om deze types te manipuleren. Deze functies zijn geïntegreerd in de bevragingstaal SQL en voldoen aan de "ISO SQL/MM Spatial Standard" [11] en de "OpenGIS Consortium's (OGC's) Simple Feature Specification for SQL" [17]. De hier besproken versie is versie 8.2 en kan geïnstalleerd worden op 32bit en 64bit uitvoeringen van Windows (2000 & NT), AIX, HP-UX, Solaris Operating Environment en Linux for Intel. DB2 Spatial Extender ondersteunt ook Linux for S/390 (32bit) maar niet Linux for zSeries (64bit). Om ruimtelijke data te visualiseren kan gebruik gemaakt worden van een extern programma zoals *ESRI*'s ArcView GIS of van "ArcExplorer voor DB2", een programma dat in samenwerking met *ESRI* gemaakt is.

Er wordt eerst ingegaan op de ondersteunde datatypes van de Spatial Extender. Dan wordt naar de gebruikte index gekeken en wanneer deze gebruikt kan worden. Ten slotte lichten we enkele functies toe waarmee de spatial data geanalyseerd kan worden.

### 5.1.1 DB2 - Datatypes

Ruimtelijke gegevens worden in DB2 Spatial Extender via een kolom van geometrieën in een tabel opgeslagen, de mogelijke types met hun naamgeving zijn te zien in figuur 5.1.




Figuur 5.1: DB2 Spatial Types (bron [7]).

Deze types volgen de indeling zoals gespecificeerd in de OpenGIS standaard en zijn als volgt in te delen. *ST_Point*, *ST_Linestring* en *ST_Polygon* voor enkelvoudige features, *ST_MultiPoint*, *ST_MultiLineString* en *ST_MultiPolygon* voor meervoudige features (meervoudig in de betekenis van 'de samenvoeging van samen horende delen') en een gemeenschappelijk datatype *ST_Geometry* als men niet zeker is welk datatype te gebruiken. De types in de grijze kaders in figuur 5.1 zijn niet instantieerbaar, dit wil zeggen dat er geen constructor voor dat type bestaat. Het is echter wel mogelijk om de constructor van een afgeleide klasse te gebruiken. Een voorbeeld hiervan is ST_Curve, concreet betekent dit dat er in DB2 geen krommen gebruikt kunnen worden maar enkel rechte segmenten. Dit type kan in een latere versie natuurlijk nog altijd geïmplementeerd worden.

Voordat we ruimtelijke gegevens op kunnen slaan in *DB2*, moet er eerst een spatial database aangemaakt worden. Dit zorgt ervoor dat de spatial types, functies en indexen gebruikt kunnen worden. We nemen aan dat we al over zo een database beschikken (en dat de *Spatial Extender* natuurlijk al geïnstalleerd is). De volgende stap is dan om een tabel te voorzien op spatial data. Dit gebeurt door in het schema van de tabel een kolom op te nemen voor de geometrieën. Deze kolom kan aangemaakt worden: (1) Bij de creatie van de tabel, waarbij een spatial kolom opgenomen wordt in het CREATE TABLE commando. (2) Door een spatial kolom aan een bestaande tabel toe te voegen met het ALTER TABLE commando. (3) Door in het DB2 Command Center via de GUI een spatial kolom aan te maken. (4) Bij het importeren van data, waarbij er gespecificeerd wordt dat een nieuwe tabel met een spatial kolom gemaakt moet worden. In listing 5.1 is een voorbeeld te zien van het voorzien van een spatial kolom bij het creëren van een tabel via de *command line processor* (*CLP*).

Listing 5.1: Een spatial kolom aanmaken in DB2.

```
Create Table sample_points(id SmallInt, geometry ST_Point);
Create Table sample_lines(id SmallInt, geometry ST_LineString);
Create Table sample_polygons(id SmallInt, geometry ST_Polygon);
```

Nadat een spatial kolom voorzien is in de tabel, moet deze kolom nog geregistreerd worden. Dit moet gebeuren om toegang te verlenen aan sommige visualisatie tools en aan spatial indexen. Bij "ArcExplorer for DB2" moet de integriteit van de data gegarandeerd worden, dit gebeurt door de restrictie dat iedere spatial waarde hetzelfde referentiesysteem gebruikt. Deze restrictie kan opgelegd worden door de kolom te registreren. Om toegang te verlenen aan spatial indexen moet ook hier alle data hetzelfde spatial referentiesysteem gebruiken. Als dit niet gebeurt dan kan een index verkeerde resultaten

teruggeven. Het registreren van een spatial kolom kan gedaan worden in het *DB2 Control Center*, door het commando "db2se register_spatial_column" te gebruiken (zie listing 5.2) of door het importeren van een *SDE transfer file*, waarbij de spatial kolom van een nieuwe tabel automatisch geregistreerd wordt.

Listing 5.2: Registreren van een spatial kolom in DB2.

```
call db2gse.ST_register_spatial_column(tabel_schema, tabel_naam, kolom_naam,
SRS_naam, ?, ?);
```

Nadat we de tabel in gereedheid gebracht hebben, kunnen er ruimtelijke gegevens in geladen worden. Dit kan door de gegevens manueel in te voeren met het INSERT commando, ze te *geocoden* met de meegeleverde of een externe geocoder of ze te *importeren* vanuit een *shape file* of vanuit een *SDE transfer file*. Een voorbeeld van het manueel invoeren van gegevens is te zien in listing 5.3 en een voorbeeld van het importeren uit een *shape file* is te zien in listing 5.4.

Listing 5.3: Manueel invoeren van spatial data in DB2.

```
Insert Into sample_points(id, geometry)
      Values (1, ST_Point('point(6 5)',1));
Insert Into sample_lines (id, geometry)
      Values (10, ST_LineString('linestring (1 6, 7 1)', 1) ),
             (11, ST_LineString('linestring (2 2, 3 3)', 1) );
Insert Into sample_polygons(id, geometry)
      Values (20, ST_Polygon('polygon((1 3, 5 4, 2 1, 1 3))', 1) );
```

Listing 5.4: Inladen van spatial data uit een shape file in DB2.

```
call db2gse.ST_Import_Shape( '/opt/teleatlas/belbel_______sm', Null,
'WGS84_SRS_1003', Null, 'gemeentes', Null, Null, Null, 'geometry', Null, Null,
Null, Null, Null, Null, Null, Null, Null, ?, ?);
```

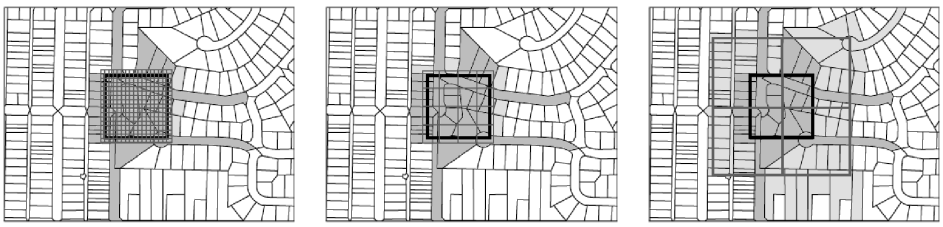
De code in listing 5.3 zorgt van het invoeren van één punt, twee linestrings en één polygon samen met hun $ID$'s in de drie tabellen die in listing 5.1 aangemaakt werden. Met de code in listing 5.4 worden alle kolommen van de *shape file*: *belbel_____sm* (een bestand van *Tele Atlas* met de middelpunten van Belgische gemeenten) ingeladen in een nieuwe tabel met de naam *gemeentes*, het gebruikte referentiesysteem is *WGS84_SRS_1003* en de kolom voor de spatial data krijgt de naam *geometry*. De overige waardes kunnen gebruikt worden om optionele parameters mee te geven, bijvoorbeeld om alleen specifieke kolommen van de *shape file* in te laden.

Voor de syntax van de overige methoden om ruimtelijke gegevens in te voeren, wordt verwezen naar de *User manual* van *DB2 Spatial Extender* (hoofdstukken 9 en 10 van [7]).

### 5.1.2 DB2 - Indexen

Om een betere performantie te krijgen voor queries over spatial data, wordt aangeraden om een index te creëren voor deze data. *DB2 Spatial Extender* voorziet in *grid indexing* voor meer-dimensionale gegevens en is in het bijzonder geoptimaliseerd voor twee-dimensionale data in een geprojecteerd coördinaten systeem. De spatial grid index gebruikt een *fixed* raster, hierbij wordt een entry gemaakt voor iedere doorsnede van een cel en de *minimum bounding box* (*mbb*) van een geometry.

Het bijzondere aan de index van *DB2* is dat er tot drie index-niveaus (*grid levels*) gedefinieerd kunnen worden. Eén DB2-index komt dus overeen met drie aparte grid indexen waarbij elk niveau een andere korrel/celgrootte heeft. Dit wordt geïllustreerd in figuur 5.2 waarbij het zwarte kader het query window voorstelt, het donkergrijze raster de index, de donkergrijze polygonen de door het query window geselecteerde gebieden en de lichte polygonen de *false hits* van de index.



(a) Kleine grid (b) Medium grid (c) Grote grid

Figuur 5.2: Grid niveaus bij één DB2 index (bron [7]).

Het voordeel van meerdere niveaus is dat objecten van verschillende grootte efficiënter geïndexeerd kunnen worden. Als een object op een bepaald niveau in meer dan vier cellen ligt, wordt het object naar een hoger niveau gepromoveerd. Als een object echter in meer dan tien cellen ligt op het hoogste niveau, dan wordt er een *overflow* index (vierde niveau) gebruikt die door het systeem gedefinieerd is, dit om te veel entries in de index te vermijden. Voor de beste performantie moeten de index-niveaus zo gekozen worden dat de overflow index niet gebruikt hoeft te worden. Het nadeel van meerdere niveaus is dat ieder niveau doorzocht moet worden voor een spatial query ook als de geometrieën ongeveer dezelfde grootte hebben. Hierdoor zal een query op een relatie met drie index-niveaus minder efficiënt zijn dan wanneer er maar één index-niveau aanwezig is.

Voor het bepalen van de optimale celgrootte en het aantal niveaus bestaat er in *DB2* de *Index Advisor*. Deze analyseert de ruimtelijke gegevens in een relatie en stelt voor ieder niveau een celgrootte voor. Het commando om de *Index Advisor* aan te roepen moet rechtstreeks in een systeem-opdrachtpromt uitgevoerd worden. Een voorbeeld hiervan voor de kolom *shape* van de tabel *counties* is in listing 5.5 te zien.

Listing 5.5: DB2 Index Advisor.

```
gseidx Connect To mydb User userID Using password Get Geometry Statistics
For Column userID.counties(shape) Advise;
```

| Query Window Size | Suggested Grid Sizes | | | Cost |
|---|---|---|---|---|
| 0.1 | 0.7 | 2.8 | 14.0 | 2.7 |
| 0.2 | 0.7 | 2.8 | 14.0 | 2.9 |
| 0.5 | 1.4 | 3.5 | 14.0 | 3.5 |
| 1 | 1.4 | 3.5 | 14.0 | 4.8 |
| 2 | 1.4 | 3.5 | 14.0 | 8.2 |
| 5 | 1.4 | 3.5 | 14.0 | 24 |
| 10 | 2.8 | 8.4 | 21.0 | 66 |
| 20 | 4.2 | 14.7 | 37.0 | 190 |
| 50 | 7.0 | 14.0 | 70.0 | 900 |
| 100 | 42.0 | 0 | 0 | 2800 |

Zoek in de tabel naar de grootte van het meest gebruikte *query window*, dit is de grootte van het gebied dat gewoonlijk weergegeven wordt op het scherm. De grootte wordt in dit voorbeeld uitgedrukt in decimale coördinaten. Als het weergegeven gebied meestal 0.5 graden groot is (bij benadering 55 km), creëer dan een index met drie niveaus met celgroottes 1.4, 3.5 en 14.0. Deze index kan gecreëerd worden met het commando in listing 5.6. Om te controleren welke indexen al aanwezig zijn in DB2 kan het commando in listing 5.7 gebruikt worden.

Listing 5.6: Spatial index creatie in DB2.

```
Create Index counties_shape_idx On userID.counties(shape)
Extend Using db2gse.Spatial_Index(1.4,3.5,14.0);
```

Listing 5.7: List indexes in DB2.

```
Select tabname, indname From syscat.indexes Where tabschema = schemaName;
```

Het kan ook zijn dat een bestaande index niet meer *passend* is voor een bepaalde relatie, dit kan bijvoorbeeld voorkomen als de gegevens door een reeks *updates* veranderd zijn. Ook hiervoor kan de *Index Advisor* hulp bieden door statistieken te verzamelen over (een deel van) de index en eventueel nieuwe celgroottes aan te bevelen. Voor meer uitleg hierover zie de *User manual* van

*DB2 Spatial Extender* (hoofdstuk 11 van [7].)

Als de spatial index éénmaal gecreëerd is kan hiervan geprofiteerd worden door een bepaalde groep spatial functies, de *comparison functies* die twee geometrieën met elkaar vergelijken. In deze groep zitten de volgende functies: *ST_Contains, ST_Crosses, ST_Distance, ST_EnvIntersects, ST_Equals, ST_Intersects, ST_MBRIntersects, ST_Overlaps, ST_Touches, ST_Within* en *EnvelopesIntersect*. Opdat deze operaties een index gebruiken, is het nodig: (1) Dat de functie zich in het *WHERE* gedeelte van de query bevindt, dus er wordt geen index gebruikt als de functie in het *SELECT*, *HAVING* of *GROUP BY* gedeelte staat. (2) Dat de functie in het linkerdeel van het gebruikte predikaat staat. (3) Dat het gelijkheidsteken '=' gebruikt wordt om het resultaat van de functie met een andere expressie te vergelijken. (De enige uitzondering hierop is dat in het geval van de *ST_Distance* functie gebruik moet gemaakt worden van het *kleiner dan* symbool '<'.) (4) Dat de expressie in het rechterdeel de constante '1' is. (Behalve bij *ST_Distance* waar een positief getal volstaat.) (5) Dat de query over een kolom gaat waarop er een spatial index ligt.

Wanneer de *query optimizer* ervoor kiest om een spatial index voor één van bovenstaande functies te gebruiken, worden de volgende filter-stappen doorlopen.

- Eerst worden de gridcellen berekend die in de doorsnede zitten met het *query window*. Het query window wordt bepaald door de geometry die als tweede parameter aan de functie wordt meegegeven.
- Dan wordt de index gescand op entries die in het query window liggen. Hierbij worden alleen die entries teruggegeven waarvan hun *mbb* in het query window ligt.
- Tenslotte moeten de geometrieën in deze kandidaat-verzameling nog verder geanalyseerd worden om het juiste resultaat te bekomen.

In listing 5.8 zijn enkele voorbeelden te zien van functies die gebruik maken van de spatial index op de kolom *C.GEOMETRY*.

Listing 5.8: Voorbeeld van EnvelopesIntersect in DB2 (bron [7]).

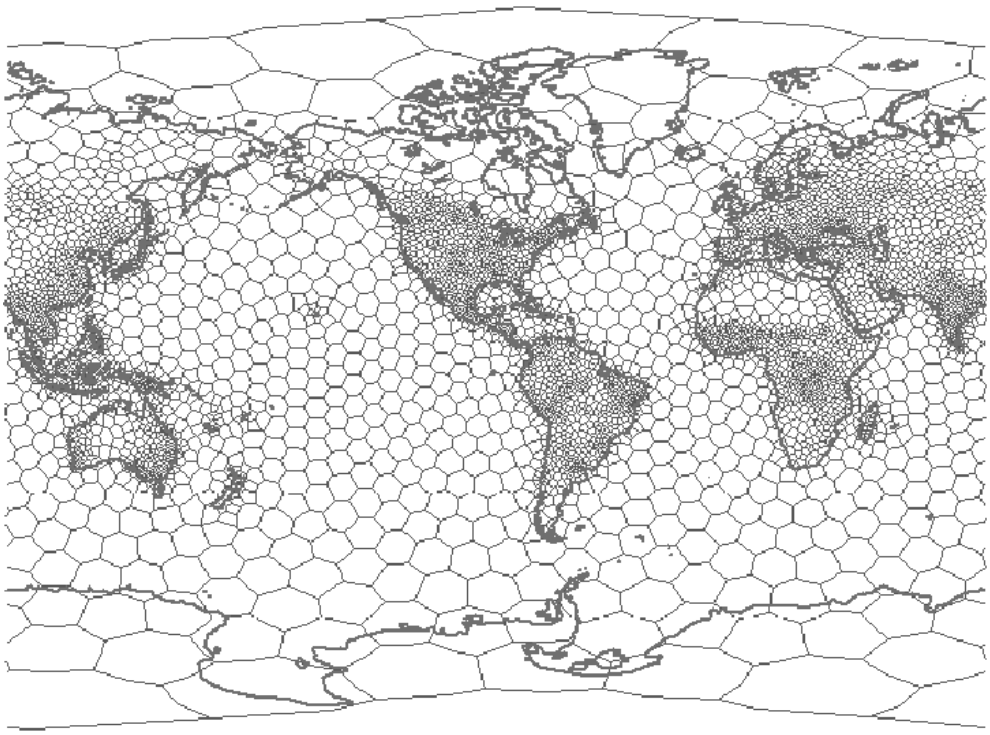
**Select** name **From** counties **AS** c
**Where EnvelopesIntersect**(c.geometry, −73.0, 42.0, −72.0, 43.0, 1) = 1;

**Select** name **From** counties **AS** c **Where ST_Intersects**(c.geometry, geometry2) = 1;

**Select** name **From** counties **AS** c **Where ST_Distance**(c.geometry, geometry2) < 500;

In het eerste voorbeeld in listing 5.8 wordt het query window handmatig gedefinieerd door de rechthoek met als hoekpunten links-onder (-73.0, 42.0) en rechts-boven (-72.0, 43.0). De laatste parameter '1' dient om het gebruikte spatial referentiesysteem voor te stellen. In het tweede voorbeeld in listing 5.8 wordt de *mbb* van de variabele *geometry2* als query window gebruikt. In het derde voorbeeld in listing 5.8 wordt gecontroleerd of de kortste afstand tussen elke twee punten van de geometrieën kleiner is dan 500 meter, deze eenheidsmaat kan ook expliciet opgegeven worden.

Er bestaat ook nog een ander soort index, de Voronoi index. Deze index maakt echter deel uit van *DB2 Geodetic Extender* en wordt gebruikt voor *geodetic data*, dit zijn ruimtelijke gegevens in het bolvormig geografische coordinaten systeem. Hierbij wordt gewerkt met de *minimum bounding* ***circle*** van geometrieën en wordt de ruimte ingedeeld in Thiessen veelhoeken (zie paragraaf 2.2.4.1 op pagina 20). Een voorbeeld van deze indeling is te zien in figuur 5.3. Voor meer informatie hierover wordt verwezen naar de *User manual* van *DB2 Geodetic Extender* (hoofdstuk 16 tot en met 19 van [7]).



Figuur 5.3: Geodetische indeling van de ruimte (bron [7]).

### 5.1.3 DB2 - Functies

Om ruimtelijke gegevens met *DB2 Spatial Extender* aan een spatial analyse te onderwerpen, kan gebruikt gemaakt worden van verschillende spatial functies. Deze functies kunnen gebruikt worden in *interactieve SQL statements*. Zo een SQL statement kan rechtstreeks via de *DB2 Command Editor*, het *DB2 Command Window* of de *DB2 command line processor* (*CLP*) gegeven worden. Een andere manier is door het SQL statement in een programma op te nemen dat embeded SQL voor DB2 toelaat. De spatial functies kunnen ingedeeld worden volgens het type van de operatie dat ze uitvoeren, zoals: (1) Het converteren van geometrieën naar en van andere dataformaten. (2) Het vergelijken van geometrieën op doorsnedes, grenzen en andere kenmerken. (3) Het teruggeven van informatie over eigenschappen van geometrieën. (4) Het genereren van nieuwe geometrieën aan de hand van bestaande geometrieën. (5) Overige functies.

#### 5.1.3.1 Conversie functies

Deze functies laten toe om geometrieën te converteren van en naar verschillende data uitwisselings formaten. We beginnen met functies om geometrieën aan te maken, ook wel de *constructor functies* genoemd. De ondersteunde formaten zijn: de *Well-known text* (WKT) representatie en de *Well-known binary* (WKB) representatie [17], de *ESRI shape* (shape) representatie [4] en de *Geography Markup Language* (GML) representatie [18]. Het WKT en WKB formaat is beide door OpenGIS gedefinieerd, de WKT representatie stelt de gegevens voor met behulp van ASCI terwijl de WKB representatie binair is, dit naar analogie met de database types CLOB en BLOB. Het shape formaat is binair en kan zowel ruimtelijke als alfanumerieke gegevens opslaan. Dit formaat wordt gebruikt voor de brongegevens van de case study in volgend hoofdstuk en worden daar in paragraaf 6.1.1 toegelicht. Het GML formaat is ook door OpenGIS gedefinieerd en hierin kunnen zowel ruimtelijke als alfanumerieke gegevens opgeslagen worden. Dit gebeurt met behulp van XML en wordt bijgevolg ook als tekst opgeslagen. De syntax van de constructor functie wordt in listing 5.9 getoond.

Listing 5.9: Syntax constructor functies in DB2.

```
[db2gse.] geometry_type ( WKT       , [ srs_id ] )
                          WKB
                          shape
                          GML
                          coordinaten
```

De parameter '*db2gse.*' duidt het gebruikte schema aan. De parameter '*geometry_type*' stelt de gewenste geometry voor en kan ieder '*instantieerbaar*' type zijn (ST_Point, ST_LineString, ...). De volgende parameter moet een spatial waarde bevatten in één van de vermelde formaten. In het geval van '$WKT$' of '$GML$' is dit een CLOB(2G), voor '$WKB$' of '*shape*' is dit een BLOB(2G). Het is ook mogelijk om rechtstreeks een coördinaten lijst in te voeren. De laatste (optionele) parameter '*srs_id*' stelt het $ID$ voor van het spatial referentiesysteem voor.

Een opgeslagen geometry kan op twee manieren terug geconverteerd worden naar een data uitwisselings formaat. Een eerste, impliciete manier is door aan te geven dat alle ruimtelijke resultaten van een query in het gewenste data formaat moeten teruggegeven worden. In het geval van '$WKT$' moet hiervoor de default *transform group* op *ST_WellKnownText* gezet worden. De tweede, expliciete manier is door in een query een conversie functie aan te roepen op de ruimtelijke resultaten. Voor '$WKT$' is dit ST_AsText, voor '$WKB$' is dit ST_AsBinary, voor '*shape*' is dit ST_AsShape en voor '$GML$' is dit ST_AsGML. In listing 5.10 wordt de $WKT$ representatie expliciet opgevraagd van de geometry *geom* in tabel *sample_geometry*.

Listing 5.10: Voorbeeld conversie functies in DB2 (bron [7]).

```
Select id, VarChar(db2gse.ST_AsText(geom), 50) As WKTGEOM
From sample_geometry;

ID     WKTGEOM
---    ------------------------------------
100    POINT (30.00000000 40.00000000)
200    LINESTRING (50.00000000 50.00000000, 100.00000000 100.00000000)
```

Behalve deze conversie functies ondersteunt *DB2 Spatial Extender* ook nog andere functies op data uitwisselings formaten, dit om aan de standaarden in [17] en [11] te voldoen. Deze functies zijn terug te vinden in de *User manual* van *DB2 Spatial Extender* (hoofdstuk 22 van [7]).

#### 5.1.3.2 Functies om geometrieën te vergelijken

Deze functies geven informatie terug over vergelijkingen tussen geometrieën, hiernaar wordt daarom ook verwezen als de *comparison functies*. Het resultaat van deze functies is ofwel 1 (één) als de vergelijking opgaat, ofwel 0 (nul) als er niet aan de vergelijking voldaan wordt, ofwel *null* als de vergelijking niet uitgevoerd kan worden. Het laatste geval kan voorkomen als de functie niet gedefinieerd is voor het type van de input parameters of als een parameter zelf *null* is.

Om de geometrieën te vergelijken maakt de *Spatial Extender* gebruik van het *Dimensionally Extended 9 Intersection Model* (DE-9IM). Dit model is een uitbreiding van het model voor topologische relaties dat reeds voorgesteld werd in paragraaf 2.3.3.1 op pagina 30. Daarbij wordt gekeken naar de doorsnedes van het inwendige en de grens van de geometrieën. Het DE-9IM voegt hieraan ook nog de doorsnede met het uitwendige toe, evenals de dimensie van al deze doorsnedes. Ondanks deze uitbreiding, kan dit model toch alleen maar dezelfde negen topologische relaties onderscheiden als vermeld in paragraaf 2.3.3.1. Dit komt omdat het voor twee-dimensionale gegevens niet uit maakt of het uitwendige beschouwd wordt of niet. Voor gegevens met een dimensie die echter hoger dan twee is, kan met het uitwendige aditionele relaties gedefinieerd worden. Het beschouwen van de dimensie van het resultaat van de doorsnedes is wel van belang. Hierdoor is het mogelijk om uitzonderlijke relaties, zoals bijvoorbeeld 'A *overlap_bdy_disjoint* B' met A en B linestrings, te definiëren. In *DB2* wordt naar deze relatie verwezen als *ST_Crosses* waarbij de doorsnede slechts uit één punt bestaat. Alle relaties tussen twee geometrieën worden in *DB2* voorgesteld in een *3-op-3* matrix, waarvan de waarde in een cel de dimensie aangeeft van het resultaat van de doorsnede. Deze waarde kan $-1$, 0, 1 of 2 zijn, waarbij $-1$ aangeeft dat de doorsnede leeg is. Tabel 5.1 illustreert deze matrix.

| | **Inwendige** | **Grens** | **Uitwendige** |
|---|---|---|---|
| **Inwendige** | dim(I(a) ∩ I(b)) | dim(I(a) ∩ G(b)) | dim(I(a) ∩ U(b)) |
| **Grens** | dim(G(a) ∩ I(b)) | dim(G(a) ∩ G(b)) | dim(G(a) ∩ U(b)) |
| **Uitwendige** | dim(U(a) ∩ I(b)) | dim(U(a) ∩ G(b)) | dim(U(a) ∩ U(b)) |

Tabel 5.1: De DE-9IM matrix (bron [7]).

Omdat voor sommige relaties niet alle cellen in de matrix een vaste dimensie moeten hebben, als het bijvoorbeeld niet uitmaakt of de doorsnede van het uitwendige leeg is of niet, zijn er enkele afkortingen voorzien om verschillende dimensiewaarden te combineren. Als de doorsnede niet leeg mag zijn, maar de dimensie niet ter zake doet (dimensie 0, 1 of 2), kan $T$ gebruikt worden. Als de doorsnede leeg moet zijn (dimensie $-1$), kan $F$ gebruikt worden. Als het niet uitmaakt of de doorsnede leeg is of niet (dimensie $-1$, 0, 1 of 2) kan $*$ gebruikt worden. Een 0 geeft aan dat de dimensie van de doorsnede exact 0 moet zijn. Terwijl een 1 of 2 aangeeft dat de dimensie maximum 1 respectievelijk 2 mag zijn.

*DB2 Spatial Extender* heeft twee functies die controleren of twee geometrieen in elkaar vervat zijn, namelijk *ST_Contains* en *ST_Within*. ST_Contains geeft als resultaat 1 (één) terug als de tweede geometry volledig in de eerste ligt. ST_Within doet juist het omgekeerde hiervan en heeft een positief resultaat als de eerste geometry volledig in de tweede ligt. De matrix van ST_Contains is gegeven in tabel 5.2 en die van ST_Within bekomt men door die van ST_Contains te spiegelen volgens zijn hoofdas.

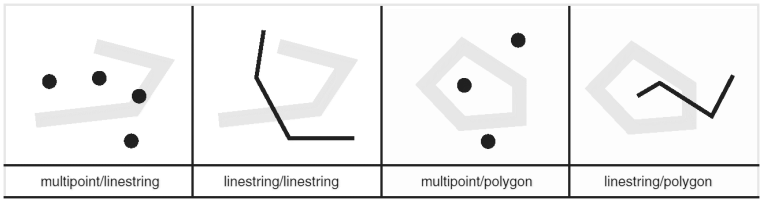
| | **Inwendige b** | **Grens b** | **Uitwendige b** |
|---|---|---|---|
| **Inwendige a** | T | * | * |
| **Grens a** | * | * | * |
| **Uitwendige a** | F | F | * |

Tabel 5.2: De matrix van ST_Contains (bron [7]).

Andere functies controleren dan weer op de doorsnedes tussen twee geometrieën. Deze zijn *ST_Intersects*, *ST_Crosses*, *ST_Overlaps* en *ST_Touches*. Opdat ST_Intersects een positief resulaat zou hebben, moet er aan één van vier mogelijke voorwaarden voldaan zijn, waarbij iedere voorwaarde overeen komt met een matrix. Ofwel moeten de inwendigen van de geometrieën elkaar snijden, ofwel hun grenzen, ofwel het inwendige van de eerste geometry met de grens van de tweede, ofwel de grens van de eerste met het inwendige van de tweede geometry. Bij iedere van de vier voorwaarden maakt het niet uit wat de dimensie van de doorsnede is ($T$) en ook niet of er nog andere delen van de geometrieën elkaar snijden ($*$).

Het resultaat van *ST_Crosses* is positief als de geometry gevormd door de doorsnede, een lagere dimensie heeft dan de input parameters of deel is van het inwendige van beide geometrieën. Deze functie is echter niet gedefinieerd als de eerste inputparameter een polygon of multipolygon is of als de tweede parameter een punt of multipunt is. In deze situatie zal het resultaat van ST_Crosses *null* zijn. Figuur 5.4 illustreert vier gevallen van ST_Crosses, hierbij is de zwarte geometry $a$ en de grijze $b$. Als de inputgeometrieën (multi)lines zijn, dan moet de dimensie van de doorsnede van de inwendigen 0 zijn (een punt). Voor de andere gevallen moet gelden dat het inwendige van de eerste geometry snijdt met zowel het inwendige als uitwendige van de tweede geometry.

*ST_Overlaps* vergelijkt geometrieën van dezelfde dimensie. Hierbij wordt een positief resultaat teruggegeven als de geometry in de doorsnede nog dezelfde dimensie heeft, maar deze niet gelijk is aan de input geometrieën. Tabel




Figuur 5.4: Voorbeeld ST_Crosses (bron [7]).

5.3 toont de matrix van punten, multipunten, polygonen en multipolygonen. Voor linestrings en multilinestrings moet de dimensie van de doorsnede van de inwendigen precies 1 zijn (een linestring), als deze lager is geeft ST_Overlaps een negatief resultaat terug.

| | **Inwendige b** | **Grens b** | **Uitwendige b** |
|---|---|---|---|
| **Inwendige a** | T | * | T |
| **Grens a** | * | * | * |
| **Uitwendige a** | T | * | * |

Tabel 5.3: De matrix van ST_Overlaps (bron [7]).

De laatste functie die op doorsnedes tussen twee geometrieën controleert is *ST_Touches*. Hierbij mogen de inwendigen elkaar niet snijden en moeten alle gemeenschappelijke punten zich op de grens van één van de twee geometrieën bevinden. Deze functie is alleen gedefinieerd als één van beide inputparameters een linestring, multilinestring, polygon of multipolygon is.

De volgende twee functies worden vooral gebruikt in de filterstap bij indexen en vergelijken de *enveloppes* en de *minimum bounding rectangles* (MBR's) van twee geometrieën. Deze functies zijn *ST_EnvIntersects* en *ST_MBRIntersects*. Voor de meeste geometrieën is de enveloppe en de MBR (minimum bounding rectangle) hetzelfde, behalve voor punten en verticale en horizontrale linestrings. Hiervoor is de enveloppe iets groter omdat deze een oppervlakte moet hebben en een MBR niet. *ST_EnvIntersects (a, b)* controleert dus of de enveloppes van twee geometrieën $a$ en $b$ elkaar snijden en is eigenlijk een afkorting voor ST_Intersects (ST_Envelope(a), ST_Envelope(b)). *ST_MBRIntersects* controleert of de MBR van twee geometrieën elkaar snijden. Omdat in het geval van een punt of een horizontale of verticale linestring de MBR samenvalt met de geometry, controleert deze functie eigenlijk de geometry zelf.

| | **Inwendige b** | **Grens b** | **Uitwendige b** |
|---|---|---|---|
| **Inwendige a** | T | * | F |
| **Grens a** | * | * | F |
| **Uitwendige a** | F | F | * |

Tabel 5.4: De matrix van ST_Equals (bron [7]).

*ST_Equals* controleert of twee geometrieën gelijk zijn, hierbij is de volgorde van de gebruikte punten in een linestring of polygon niet belangrijk. De matrix van deze relatie is te zien in tabel 5.4, deze verzekert dat de inwendigen elkaar snijden, maar dat de doorsnede van het inwendige of de grens van de ene geometry met het uitwendige van de andere geometry leeg is.

De functies *ST_EqualSRS* en *ST_EqualCoordsys* controleren niet of twee geometrieën gelijk zijn, maar of ze hetzelfde spatial referentiesysteem respectievelijk coördinaten systeem gebruiken. Voorwaarde bij ST_EqualSRS is dat de *ID*'s ervan in beide gevallen ingevuld moeten zijn.

De laatste voorgedefinieerde functie die geometrieën vergelijkt is *ST_Disjoint*. De functie controleert of twee geometrieën volledig van elkaar verschillen. De matrix hiervan staat in tabel 5.5.

| | **Inwendige b** | **Grens b** | **Uitwendige b** |
|---|---|---|---|
| **Inwendige a** | F | F | * |
| **Grens a** | F | F | * |
| **Uitwendige a** | * | * | * |

Tabel 5.5: De matrix van ST_Disjoint (bron [7]).

Tot slot voorziet *DB2* nog de functie *ST_Relate*. Deze functie krijgt als input twee geometrieën en een string met de tekstuele voorstelling van een willekeurige DE-9IM matrix. Als de geometrieën voldoen aan die matrix, heeft de functie een positief resultaat. Hierdoor is *DB2 Spatial Extender* voorzien op eventuele toekomstige nieuwe relaties.

#### 5.1.3.3 Functies voor informatie van geometrische eigenschappen

Deze groep van functies geeft informatie terug over verschillende eigenschappen van geometrieën. Informatie zoals het datatype, de gebruikte coördinaten en punten, de dimensie en of de geometry al dan niet gesloten, leeg of

simpel is.

Om het type van een geometry te weten te komen kan *ST_GeometryType* gebruikt worden. De informatie die bekomen kan worden over punten in een geometry is de dimensie van de coördinaten, welke hiervan ingevuld zijn ($X$, $Y$, $Z$ en/of $M$), de waardes hiervan en de *minimum* en *maximum* waardes in iedere dimensie. Met nog andere functies kan het meetkundige middelpunt van een geometry, het startpunt, middelpunt of eindpunt van een curve, het aantal objecten in een multiPoint, multiLineSting of een multiPolygon, of het $n^{de}$ object in een lijst van geometrieën verkregen worden. Het is ook mogelijk om de dimensie van de geometry te weten te komen, merk op dat dit niet hetzelfde is als die van de coördinaten. In het geval van een polygon kan de oppervlakte bepaald worden en van een curve kan de lengte bepaald worden. Er zijn eveneens functies om te controleren of een geometry simpel, gesloten of *empty* is. Tenslotte kan ook nog het $ID$ en de *naam* van het gebruikte spatial referentiesysteem opgevraagd worden.

#### 5.1.3.4 Functies die nieuwe geometrieën genereren

Deze paragraaf gaat over functies die nieuwe geometrieën afleiden van bestaande geometrieën. Hierbij sluiten enkele functies aan bij de vorige paragraaf, deze geven namelijk geometrieën terug die een eigenschap van een bestaande geometry zijn. Andere functies converteren tussen geometrieën, voegen een formatie van de omliggende ruimte toe aan een geometry, groeperen verschillende geometrieën tot één object of creëren variaties van een geometry.

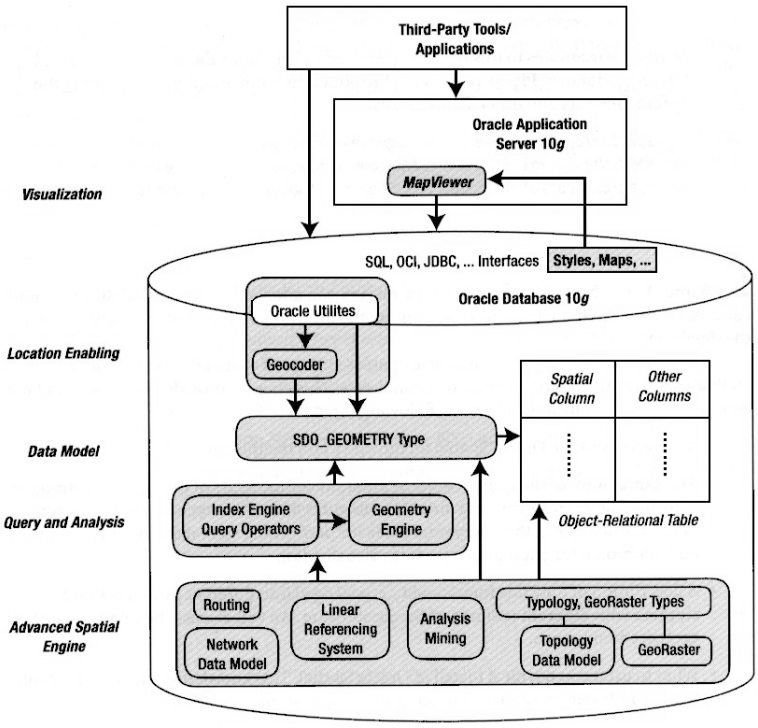
De groep van functies die eigenschappen van geometrieën teruggeeft, bestaat uit functies die de omtrek, de enveloppe of de minimum bounding box van een geometry bepalen. De volgende groep functies creëert nieuwe geometrieen door bestaande geometrieën te *converteren* naar één van de instantieerbare types, dit zijn de types in witte kaders in figuur 5.1 op pagina 73. Tot de groep functies die omliggende ruimte toevoegen aan een geometry, hoort *ST_Buffer* die een object uitbreid met alle punten die binnen een bepaalde straal van het originele object liggen. De functie *ST_ConvexHull* heeft als resultaat de kleinst mogelijke convexe polygon die alle punten van een geometry bevat. De functies *ST_Difference*, *ST_Intersection*, *ST_SymDifference* en *ST_Union* berekenen het verschil, de doorsnede, de XOR operatie en de unie van twee geometrieën. Met *ST_AppendPoint*, *ST_RemovePoint* en *ST_ChangePoint* kan een curve veranderd worden. En met *ST_X*, *ST_Y*, *ST_Z* en *ST_M* kunnen de coördinaten van een punt gewijzigd worden.

#### 5.1.3.5 Overige functies

Omdat deze functies in geen van de andere categoriën passen worden ze hier appart vermeld. *ST_Distance* berekent de kortste afstand tussen twee geometrieën. Hierbij kan optioneel ook nog de eenheidsmaat opgegeven worden waarin de afstand berekend moet worden. Met *ST_GetIndexParams* kunnen de parameters opgevraagd worden die bij de indexcreatie opgegeven zijn. En de laatste functie die hier voor DB2 vermeld wordt is de functie *ST_Transform* waarmee geometrieën tussen verschillende spatial referentiesystemen geconverteerd kan worden. Voor extra informatie over functies wordt verwezen naar de *User manual* van *DB2 Spatial Extender* (hoofdstuk 22 van [7])

## 5.2 Oracle Spatial

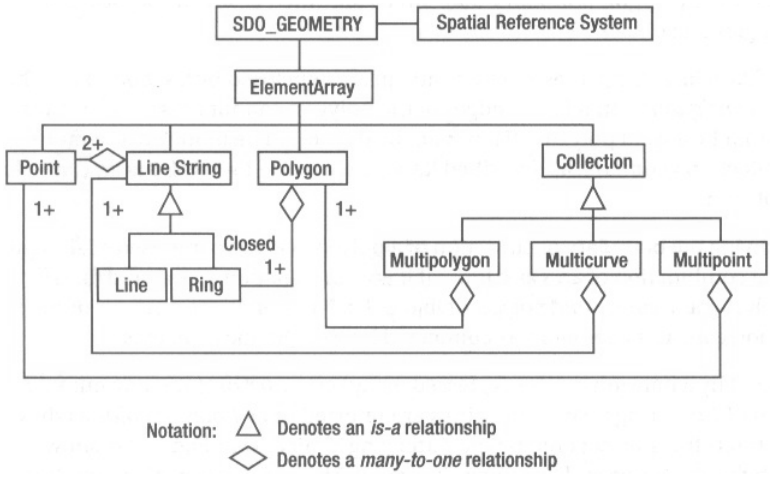
*Oracle Spatial* is een optie bij *Oracle Enterprise Edition* en breidt *Oracle Locator*, een basisset van spatial functionaliteit die zowel in de standaard als de enterprise editie aanwezig is, verder uit. Locator biedt een spatial datamodel en een beperkt aantal analyse operatoren die de *Index Engine* gebruiken. Oracle spatial voegt hier nog de volgende functionaliteit aan toe: spatial functies die de *Geometry Engine* gebruiken, de geocoder om adressen om te zetten naar punt geometrieën en de *Advanced Spatial Engine* functionaliteit zoals netwerk analyse en routebepaling. Al deze functionaliteit is geïntegreerd met de bevragingstaal SQL. Zie figuur 5.5 voor de architectuur van Oracle Spatial. De hier besproken versie is versie 10*g* release 1 en kan geïnstalleerd worden op Windows (32 & 64 bit), Solaris Operating System (SPARC & x86), AIX, HP, Linux (32 & 64 bit), IBM OS en Apple. Oracle vermeldt ook nog dat Locator alleen niet bedoeld is voor GIS-toepassingen die complexe analyse vereisen en dat beide systemen voor ontwikkelaars van toepassingen bedoeld zijn en niet voor eindgebruikers.



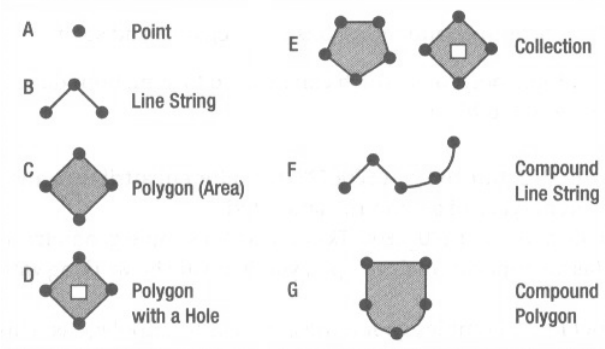

Figuur 5.5: Oracle architectuur (bron [19]).

### 5.2.1 Oracle - Datatypes

In Oracle Spatial worden alle ruimtelijke gegevens in één type opgeslagen namelijk SDO_Geometry. Hierin kunnen punten, line strings, polygonen en compexe geometrieën worden opgeslagen. Net zoals IBM heeft Oracle de "ISO SQL/MM Spatial Standard" [11] en de "OpenGIS Consortium's (OGC's) Simple Feature Specification for SQL" [17] bestudeerd en voldoet aan alle types gedefinieerd door het OGC en aan de meeste van die van ISO. Zie figuur 5.6 voor het klassediagram van de ondersteunde types en figuur 5.7 voor enkele voorbeelden van ondersteunde types.




Figuur 5.6: Logische implementatie van SDO_Geometry (bron [19]).




Figuur 5.7: Voorbeelden van Oracle geometrieën (bron [19]).

De interne structuur van SDO_Geometry is te zien in listing 5.11.

Listing 5.11: Het SDO_Geometry type van Oracle.

| Name | Null? | Type |
|---|---|---|
| SDO_GType | | Number |
| SDO_SRID | | Number |
| SDO_Point | | SDO_Point_Type |
| SDO_Elem_Info | | SDO_Elem_Info_Array |
| SDO_Ordinates | | SDO_Ordinate_Array |

*SDO_GType* bepaalt het type van de geometry (point, line, polygon, collection, multipoint, multiline of multipolygon). *SDO_SRID* bepaalt het spatial referentiesysteem. *SDO_Point* kan één drie-dimensionaal punt bevatten (een $x$, $y$ en $z$ waarde). Als SDO_Point gebruikt wordt, dan zijn de overige twee elementen *null*. Indien het object geen punt is of een punt met meer dan drie coördinaten (bijvoorbeeld een extra $m$ waarde), dan is SDO_Point *null* en worden de twee laatste elementen gebruikt. *Opmerking*: een drie-dimensionaal punt kan ook met SDO_Elem_Info en SDO_Ordinates opgeslagen worden, maar dit wordt wegens performantie redenen niet aangeraden. *SDO_Ordinates* bevat de coördinaten en *SDO_Elem_Info* definieert hoeveel en welke geometrieën er in SDO_Ordinates zitten, waar elke geometry begint, de offset en hoe de coördinaten van een geometry verbonden zijn (met rechte lijnen of circulaire bogen).

Voordat gebruik kan gemaakt worden van ruimtelijke gegevens moet de database hierop voorbereid worden. Dit gebeurt door een kolom van het type SDO_Geometry aan te maken. Dit kan bij (1) de creatie van een tabel, (2) door de kolom achteraf toe te voegen, (3) met de GUI van Oracle, de Enterprise Manager, (4) door het importeren van data in een nieuwe tabel. Een voorbeeld van het voorzien van een spatial kolom bij de creatie van een tabel via de command line processor *SQL Plus* is in listing 5.12 te zien.

Listing 5.12: Een spatial kolom aanmaken in Oracle.

**Create Table** test (id **Number**, geometry **SDO_Geometry**);

In deze tabel kunnen nu al ruimtelijke gegevens bewaard worden, maar voor validatie, index creatie en het queryen van deze data moet er eerst nog metadata voorzien worden. Deze metadata bevat de naam van de tabel en van de spatial kolom, het spatial referentiesysteem en de tolerantie voor iedere dimensie. De tolerantie bepaalt de precisie van de data, meer bepaald als de tolerantie een waarde heeft van 0.5 en de afstand tussen twee punten A en B is minder dan 0.5, dan worden A en B als eenzelfde locatie beschouwd.

Ruimtelijke gegevens invoeren kan manueel (met INSERT of UPDATE tabel_naam SET). Ze te geocoden met de geïntegreerde geocoder, hiervoor

moeten er wel zelf referentiegegevens voorzien worden. Ze te laden uit een tekstbestand door middel van de *SQL*Loader*, ook wel *bulk loading* genoemd, hiervoor is een controle bestand nodig dat definieert hoe de data opgesplitst moet worden en in welke kolom de data moet geplaatst worden. Of de gegevens te importeren, dit kan vanuit Oracle's platform onafhankelijk *.dmp* formaat, het Well-known text (WKT) formaat of vanuit het Well-known binary (WKB) formaat. Shapefiles moeten eerst geconverteerd worden naar tekst zodat deze met behulp van bulk loading geladen kunnen worden. In listing 5.13 wordt het punt $(-77, 37)$ ingevoerd in de tabel 'test' die in listing 5.12 gedefinieerd werd. Hierbij staat '2001' voor het GType *punt*, '8307' voor het SRID, de eerste *null* voor de $z$-coördinaat, de tweede *null* voor Elem_Info en de laatste *null* voor Ordinates. In listing 5.14 wordt het exporteren naar en importeren met behulp van het *.dmp* formaat geïllustreerd.

Listing 5.13: Manueel invoeren van data in Oracle.

```
Insert Into test (id, geometry) Values
(1, SDO_Geometry(2001, 8307, SDO_Point_Type(-79, 37, Null), Null, Null));
```

Listing 5.14: Exporteren en importeren van data in Oracle.

```
EXP user1/password File=test.dmp Tables=test;
IMP user2/password File=test.dmp Indexes=N;
```

Nadat de ruimtelijke gegevens ingevoerd zijn is het mogelijk deze te valideren en indien nodig te debuggen. Een voorbeeld hiervan is het opsporen en verwijderen van dubbele punten in een tabel. Voor extra informatie en de syntax van deze en overige methodes wordt verwezen naar de *User manual* van *Oracle Spatial* (hoofdstukken 4 en 5 van [29]).

Oracle voorziet ook datatypes om topologische en raster data op te slaan, namelijk *SDO_Topo_Geometry* en *SDO_GeoRaster*. In SDO_Topo_Geometry worden de topologische relaties standaard opgeslagen en queries hieromtrent die SDO_Filter of SDO_Relate gebruiken worden efficiënt uitgevoerd. Queries in verband met afstand hebben een slechtere performantie. *SDO_GeoRaster* slaat rastergegevens, zoals afbeeldingen, efficiënt op in een $N$-dimensionale matrix, biedt een aantal import en exportfuncties en laat toe om gegevens in dit formaat met *Oracle MapViewer* te visualiseren. Meer informatie is te vinden in de *User manual* van *Oracle Spatial* (bijlage C en D van [29]).

Een laatste bemerking is dat Oracle ook een mogelijkheid voorziet om netwerkgegevens op te slaan en te ondervragen. Dit gebeurt met behulp van een *node* tabel en een *link* tabel. Aan elke tabel kan een ruimtelijke voorstelling van de node of de link toegevoegd worden, maar deze is niet noodzakelijk

voor de netwerkstructuur of analyse. Bij dit netwerkmodel hoort een Java API dat voor een uitgebreide lijst analyse functies zorgt, één van deze functies lost zelfs het *traveling salesperson probleem* op. Voor meer informatie hierover wordt verwezen naar de *User manual* van *Oracle Spatial* (hoofdstuk 10 van [29]).

### 5.2.2 Oracle - Indexen

In Oracle wordt aangeraden om indexen te creëren voor ruimtelijke gegevens om zo een betere performantie te krijgen voor spatial queries. Oracle voorziet zowel in *quadtree indexing* als in *R-tree indexing*, maar raadt het gebruik van quadtrees, die nog uit een vorige versie stammen, af. Standaard wordt er bij de creatie van een index dan ook een R-tree gebruikt. R-tree indexen kunnen op twee-, drie- of vier-dimensionale data gecreëerd worden, maar als een andere dan de standaard dimensie twee gebruikt wordt kan alleen de primaire (of filter) stap uitgevoerd worden. De secundaire (of refinement) stap is in dit geval niet beschikbaar. Een ander voordeel van de R-tree index is dat ze, in tegenstelling tot de quadtree index, voor geodetische gegevens geschikt is. Voor spatial operatoren is zelfs een index op de geometrieën vereist.

Het aanmaken van een spatial index kan net zoals het creëren van een gewone index, in SQL. Voorwaarde is wel dat voor deze tabel zijn metadata ingevuld is. Als er een fout optreedt tijdens het proces, bijvoorbeeld als de metadata nog niet ingevuld is, moet de spatial index manueel verwijderd worden. In listing 5.15 wordt getoond hoe een standaard spatial index te creëren en te verwijderen. Om te controleren welke indexen al aanwezig zijn in Oracle kan het commando in listing 5.16 gebruikt worden. *Opmerking*: om een quadtree als index te gebruiken moet "PARAMETERS('SDO_Level=$X$')" achter het *create index* commando toegevoegd worden, hierbij staat de $X$ voor het aantal niveaus van de quadtree.

Listing 5.15: Spatial index creatie in Oracle.

```
Create Index indexNaam On tabelNaam(kolom) IndexType Is mdsys.Spatial_Index;
Drop Index indexNaam;
```

Listing 5.16: List indexes in Oracle.

```
Select index_name, table_name From user_sdo_index_info;
```

Als de spatial index eenmaal gecreëerd is, kan hiervan gebruik gemaakt worden door de spatial operatoren. Deze omvatten *SDO_Within_Distance*, *SDO_NN* (nearest neighbours), *SDO_Filter* en *SDO_Relate*. De parameters

van deze operatoren omvatten twee geometrieën en eventueel nog een parameterstring. De operator SDO_Filter voert alleen de filter stap uit en SDO_Relate voert zowel de filter als de refinement stap uit. Het topologisch predikaat van deze twee operatoren moet in de parameterstring worden meegegeven en is een van de volgende waardes: *INSIDE*, *CONTAINS*, *COVEREDBY*, *ON*, *COVERS*, *TOUCH*, *OVERLAPBDYINTERSECT*, *OVERLAPBDYDISJOINT*, *EQUAL*, *ANYINTERACT* of de concatenatie van één van deze waardes, gescheiden door een '+' teken. Om de *disjoint* relatie te bepalen moet de negatie van het resultaat van *anyinteract* gebruikt worden. Opdat voor deze operatoren een index gebruikt wordt, is het nodig: (1) Dat er een index ligt op de eerste geometry in de parameterlijst. (2) Dat het resultaat van de operatie vergeleken wordt met 'TRUE'. Merk op dat het voor de parameterstring *niet* uitmaakt of hierin hoofdletters gebruikt worden, maar dat 'TRUE' *wel* in hoofdletters moet staan. Een tweede opmerking is dat op de tweede geometry in de parameterlijst geen index hoeft te liggen, dit mag zelfs een dynamisch aangemaakte geometry zijn.

Wanneer de query optimizer ervoor kiest om voor een van de bovenstaande operatoren een spatial index te gebruiken worden volgende stappen doorlopen:

- De operatie wordt eerst uitgevoerd met behulp van de index waarbij de minimum bounding box (mbb) als benadering van de geometry wordt gebruikt. Dit wordt de primaire filter genoemd.
- Het resultaat van de vorige stap wordt doorgegeven aan de *Geometry Engine* die de operatie op de geometry zelf uitvoert. Dit wordt de secondaire filter genoemd.

In sommlige gevallen kan de *query optimizer* ervoor kiezen de index niet te gebruiken en alleen de secondary filter uit te voeren. Dit komt omdat *Oracle Spatial* slechts ruwe schattingen geeft van de kost van een spatial operator met of zonder index en dat deze kost niet vergelijkbaar is met de kost van andere operatoren in SQL. Als een gewone index gebruikt wordt in plaats van de spatial index bij een query met zowel spatial als andere operatoren, dan kan met de juiste *hints* gezorgd worden dat de spatial index toch gebruikt wordt. Om te bepalen of een spatial index al dan niet gebruikt wordt is het mogelijk om het *query execution plan* van de query op te vragen. In listing 5.17 is een voorbeeld van *SDO_NN* te zien waarbij er alleen een spatial index op de tabel 'klanten' is. In deze situatie heeft de *query optimizer* maar één keuze en zal hij de spatial index gebruiken. In listing 5.18 is er ook nog een

(gewone) index op het type van klanten ('Standaard' of 'Gold') en wordt expliciet de *hint* gegeven dat de spatial index gebruikt moet worden.

Listing 5.17: Voorbeeld van SDO_NN in Oracle (bron [29]).

```
Select kl.id, kl.naam From verkopers verk, klanten kl
Where SDO_NN(kl.geom, verk.geom)='TRUE' And verk.id = 1;
```

Listing 5.18: Hints voor indexen in Oracle (bron [29]).

```
Select /* No_Index(kl type_index) Index(kl geom_index) */
kl.id, kl.naam, kl.type From verkopers verk, klanten kl
Where SDO_NN(kl.geom, verk.geom)='TRUE' And verk.id = 1;
```

Merk op dat de tabel waarop de spatial index ligt als eerste parameter moet opgegeven worden en dat 'type_index' en 'geom_index' de naam is van de (gewone) index respectievelijk de spatial index die op de tabel klanten gedefinieerd is. Voor de syntax en meer informatie over indexen wordt verwezen naar de *User manual* van *Oracle Spatial* (hoofdstuk 8 van [29]).

### 5.2.3 Oracle - Functies

In *Oracle Spatial* zijn er verschillende spatial functies om ruimtelijke gegevens te analyseren. Deze functies kunnen rechtstreeks in SQL commando's gebruikt worden, die op hun beurt weer in de Command Line Processor *SQL Plus* ingegeven kunnen worden. Een ander manier is door gebruik te maken van een programmeertaal waarin SQL commando's voor Oracle uitgevoerd kunnen worden zoals *Java*, *C* en *Pro C*. Een derde optie is gebruik te maken van Oracle's Procedural Language extension to SQL (PL/SQL) waarin alle spatial types rechtstreeks gebruikt kunnen worden evenals de standaard SQL types (Number, VarChar, ...). Deze spatial functies kunnen ingedeeld worden volgens het type van operatie dat ze uitvoeren, zoals: (1) Het converteren van geometrieën naar en van andere dataformaten. (2) Het vergelijken van geometrieën op topologische relaties en afstanden. (3) Het opvragen van informatie over eigenschappen van geometrieën. (4) Het creëren van nieuwe geometrieën aan de hand van bestaande geometrieën.

#### 5.2.3.1 Conversie functies

Deze functies laten toe om geometrieën te converteren van en naar verschillende data uitwisselingsformaten. We beginnen met de functies om spatial data naar geometrieën om te zetten, de constructor functies. De ondersteunde formaten zijn de *Well-known text* (WKT) representatie en de *Well-known binary* (WKB) representatie [17]. De syntax van de constructor functie wordt in listing 5.19 getoond.

Listing 5.19: Syntax constructor functie in Oracle.

```
SDO_Geometry ( WKT, srid                                    )
               WKB, srid
               gtype, srid, point, elem_info, ordinates
```

Een opgeslagen geometry kan ook weer terug naar de *WKT* of *WKB* representatie omgezet worden. Vanaf deze versie (*10g*) heeft Oracle hier ook de mogelijkheid om *naar* de *Geography Markup Language* (GML) representatie om te zetten aan toegevoegd. Deze omzetting moet expliciet gebeuren door een conversie functie aan te roepen. Voor *WKT* is dit *Get_WKT*, voor *WKB* is dit *Get_WKB* en voor *GML* is dit de functie *To_GMLGeometry* van het *SDO_Util* package. Het resultaat hiervan is respectievelijk een CLOB, een BLOB en een CLOB. Een voorbeeld van omzetting naar *WKT* is te zien in listing 5.20 en naar *GML* in listing 5.21.

Listing 5.20: Voorbeeld conversie naar WKT in Oracle.

```
Select geometry.Get_WKT() As WKT From test;

WKT
----------------
Point (-77 37)
```

Listing 5.21: Voorbeeld conversie naar GML in Oracle.

```
Select To_Char(SDO_Util.To_GMLGeometry(geometry) As GML From test;

GML
----------------
<gml:Point srsName="SDO:8307" xmlns:gml=http:www.test.gml">
  <gml:coordinates decimal="." cs="," ts=" ">
    -77,37
  </gml:coordinates>
</gml:Point>
```

#### 5.2.3.2 Functies om geometrieën te vergelijken

Deze functies geven informatie terug over vergelijkingen tussen geometrieën. In deze paragraaf worden twee functies besproken: de eerste is *SDO_Distance* die de afstand bepaalt tussen twee geometrieën. Het resultaat van deze functie is altijd een getal. De tweede functie is *Relate* die de topologische relatie bepaalt tussen de twee geometrieën. Het resultaat hiervan is van het type VarChar2.

*SDO_Distance* berekent de minimum afstand tussen elke twee punten op

twee geometrieën. Hierbij worden zowel de hoekpunten als de geïnterpolleerde curves van iedere geometry in rekening gebracht. Als argumenten heeft deze functie twee SDO_Geometry objecten, een tolerantie waarde en een optionele parameterstring waarin de eenheid van het resultaat kan gekozen worden. Merk op dat SDO_Distance geen gebruik maakt van een spatial index, maar tuppel per tuppel vergelijkt. Daarom wordt aangeraden de SDO_Within_Distance operator te gebruiken waar mogelijk en SDO_Distance alleen om de reële afstand tussen de twee geometrieën terug te geven. Listing 5.22 illustreert hoe deze twee functies samen gebruikt kunnen worden. Eerst worden met de index alle klanten bepaald die op maximum 250 meter liggen van de verkoper waarvan het ID '1' is en daarna worden de afstanden bepaald met SDO_Distance.

Listing 5.22: Voorbeeld van SDO_Distance in Oracle (bron [29]).

```
Select kl.id, kl.name,
SDO_Geom.SDO_Distance(kl.geometry, verk.geometry, 0.5, 'unit=meter') afstand
From verkopers verk, klanten kl
Where verk.id = 1 And SDO_Within_Distance(kl.geometry, verk.geometry,
      'distance=0.25 unit=kilometer') = 'TRUE';
```

*Relate* bepaalt de topologische relatie tussen twee geometrieën en heeft de volgende argumenten: het eerste SDO_Geometry object, een *mask*-string waarin de te controleren topologische relatie staat, het tweede SDO_Geometry object en een tolerantiewaarde. Het *masker* kan de volgende waarden hebben: *DETERMINE* (om de relatie te bepalen), *INSIDE*, *COVEREDBY*, *CONTAINS*, *EQUAL*, *OVERLAPBDYDISJOINT*, *OVERLAPBDYINTERSECT*, *ON*, *TOUCH*, *ANYINTERACT* of *DISJOINT*. Het resultaat hiervan is 'TRUE' als de geometrieën elkaar kruisen en 'ANYINTERACT' opgegeven werd, de waarde van het *masker* als hieraan voldaan wordt, 'FALSE' als niet aan het *masker* voldaan wordt of het type van de relatie als 'DETERMINE' opgegeven werd. Merk op dat Relate, net zoals SDO_Distance, geen gebruik maakt van een spatial index en dat indien mogelijk de operator SDO_Relate gebruikt moet worden. Listing 5.23 illustreert hoe deze twee functies samen gebruikt kunnen worden. Eerst wordt met behulp van de index bepaald tussen welke twee sales regio's een topologische relatie bestaat en daarna wordt bepaald welke relatie dit is.

Listing 5.23: Voorbeeld van Relate in Oracle (bron [29]).

```
Select a.id, SDO_Geom.Relate(a.geometry, 'DETERMINE', b.geometry, 0.5) relatie
From sales_regions a, sales_regions b
Where b.id = 51 And a.id <> 51 And SDO_Relate(a.geometry, b.geometry,
      'mask=TOUCH+OVERLAPBDYDISJOINT+OVERLAPBDYINTERSECT')
      = 'TRUE';
```

```
  ID RELATIE
---- --------------------
  43 OVERLAPBDYINTERSECT
  63 TOUCH
```

#### 5.2.3.3 Functies voor informatie van geometrische eigenschappen

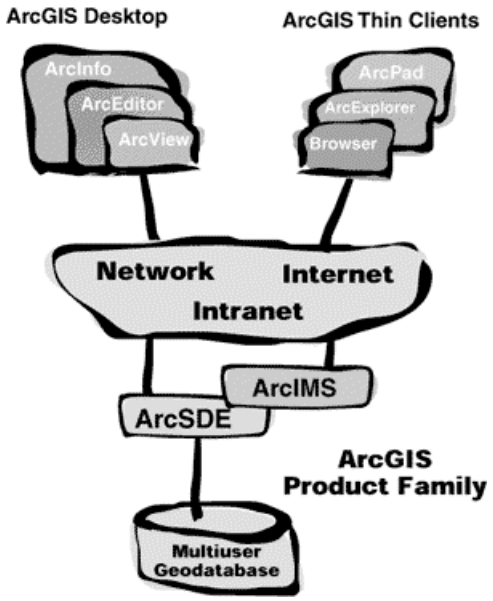
Deze groep van functies geeft informatie terug over verschillende eigenschappen van individuele geometrieën. Informatie zoals oppervlakte, lengte en bereik. Informatie over en/of wijzigingen aan de coördinaten van een SDO_Geometry object kunnen rechtstreeks op het object gebeuren. Om de lengte of oppervlakte van een geometry te weten te komen kunnen *SDO_Length* en *SDO_Area* gebruikt worden. Hierbij moet behalve de geometry ook nog een tolerantiewaarde en eventueel een eenheidsmaat opgegeven worden. Het bereik van een geometry wordt bepaald door zijn minimum bounding rectangle (*mbr*). De minimum of maximum waarde hiervan kan rechtstreeks opgevraagd worden voor een bepaalde dimensie met *SDO_Min_MBR_Ordinate* en *SDO_Max_MBR_Ordinate*. Met *SDO_Centeroid* kan het zwaartepunt van een geometry bepaald worden, met *SDO_Aggr_Centeroid* het zwaartepunt van een set van geometrieën en met *SDO_PointOnSurface* kan een willekeurig punt op een geometry opgevraagd worden.

#### 5.2.3.4 Functies die nieuwe geometrieën genereren

Deze paragraaf gaat over functies die nieuwe geometrieën afleiden van bestaande geometrieën. Tot deze groep functies hoort: De *SDO_Buffer* functie die een buffer rond een object creëert. Als argument heeft deze functie een geometry, de afstand van de buffer, een tolerantiewaarde en eventueel een eenheidsmaat. De set-functies die op twee spatial objecten inwerken: *SDO_Intersection*, *SDO_Difference*, *SDO_Union* en *SDO_XOR*. Hierbij moeten de twee geometrieën en een tolerantiewaarde opgegeven worden. De funtie *SDO_ConvexHull* creëert de kleinst mogelijke convexe polygon rond een object, terwijl *SDO_MBR* de kleinst mogelijke rechthoek rond een object creëert. *SDO_Aggr_ConvexHull* en *SDO_Aggr_MBR* doen hetzelfde maar dan voor een set van geometrieën. En de functie *SDO_Aggr_Union* creëert een spatial object dat de unie is van een set van geometrieën. Voor extra informatie en de syntax van deze en overige functies wordt verwezen naar de *User manual* van *Oracle Spatial* (hoofdstuk 9 van [29]).

## 5.3 ArcGIS

De ArcGIS familie bevat verschillende GIS toepassingen die ontwikkeld zijn door ESRI en die toelaten om ruimtelijke gegevens te creëren, beheren, integreren en te analyseren. De versie die hier besproken wordt is ArcGIS 8.1, kan geïnstalleerd worden op Windows NT, Windows 2000 of Windows XP en bestaat uit *ArcView*, *ArcEdit*, *ArcInfo*, *ArcGIS extensions*, *ArcSDE* en *ArcIMS*. Zie figuur 5.8 voor een overzicht. Naar ArcView, ArcEdit en ArcInfo wordt ook gerefereerd als *ArcGIS Desktop* omdat ze dezelfde architectuur, kern functionaliteit en ontwikkelingsomgeving hebben. Hiervan is ArcView de 'lichtste' met uitgebreide functies voor mappingen en analyse samen met simpele tools voor het bewerken van de data en geoprocessing. ArcEditor bevat de volledige functionaliteit van ArcView en voegt hier geavanceerde bewerkingsmogelijkheden aan toe. De 'zwaarste' van de drie is ArcInfo die de functionaliteit van de twee andere bevat inclusief geavanceerde geoprocessingstools. In de rest van deze tekst ligt de nadruk vooral op *ArcView* omdat de functionaliteit die hierin aangeboden wordt het meeste aansluit bij die van DB2 Spatial Extender en Oracle Spatial.




Figuur 5.8: Overzicht ArcGIS 8.1 (bron [31]).

Eerst worden de onderdelen van de ArcGIS familie toegelicht. Daarna wordt ingegaan op de ondersteunde datatypes. Tenslotte worden de functies van ArcView besproken.

### 5.3.1 ArcGIS - Onderdelen

**ArcGIS Desktop** bestaat uit de drie programma's *ArcCatalog*, *ArcMap* en *ArcToolbox*, afhankelijk van de variant (ArcView, ArcEditor of ArcInfo) zijn hierin sommige functies en/of tools niet beschikbaar.

**ArcCatalog** kan gebruikt worden om zowel spatial data als zijn metadata te beheren, doorzoeken, documenteren en te previewen. Deze gegevens kunnen zowel uit bestanden komen als uit een geodatabase. Een geodatabase is een relationele database waarin alfanumerieke en spatial gegevens opgeslagen kunnen worden en die gekoppeld is met ArcGIS Desktop met behulp van ArcSDE.

**ArcMap** laat toe om ruimtelijke gegevens te visualiseren, bewerken en te analyseren. Deze toepassing wordt verder besproken in paragraaf 5.3.3.

**ArcToolbox** is een eenvoudige toepassing die verschillende geoprocessing tools groepeert. In ArcView en ArcEditor zijn dit een 20-tal tools om te converteren tussen verschillende dataformaten terwijl dit er in ArcInfo meer dan 100 zijn waarmee men de spatial data kan converteren, beheren en analyseren. Een aantal analyse tools hiervan zijn ook terug te vinden in ArcMap onder de *geoprocessing wizard* en dit zowel in ArcView, ArcEditor als ArcInfo.

**ArcGIS extensions** zijn functionele uitbreidingen voor ArcGIS Desktop. Enkele beschikbare extensions zijn: De *Spatial Analyst* die extra analyse functionaliteit toevoegt, zoals de gewogen-kost afstand en analyse van grid-data. De *3D Analist* die toelaat om drie-dimensionale gegevens te visualiseren en te analyseren. De *Geostatistical Analyst* die toelaat om trend analyses te doen. En de *StreetMap* die voorziet in geocoding.

**ArcSDE** zorgt voor de interface tussen ArcGIS Desktop en een geodatabase. In een geodatabase wordt iedere feature als een object in één tuppel opgeslagen samen met zijn alfanumerieke data. Het datamodel dat hiervoor gebruikt wordt is voor een groot stuk analoog aan dat van een *coverage* en kan nog verder uitgebreid worden met zelfgedefinieerde objecten. Voor meer informatie over deze datatypes, zie paragraaf 5.3.2.

**ArcIMS** voegt *Internet Mapping Services* toe aan een ArcGIS systeem waarmee men spatial data over een netwerk kan streamen. Deze gegevens kunnen dan visualiseerd en geanalyseerd worden door alle ArcGIS Desktop clients en lightweight viewers zoals *ArcExplorer*.

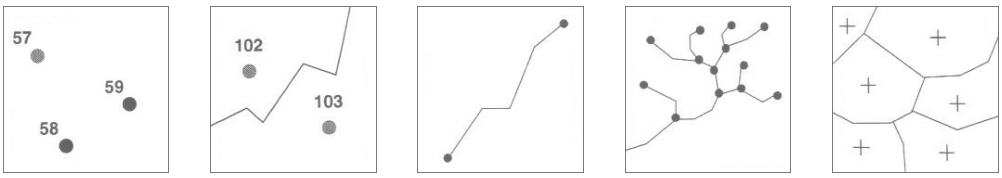
### 5.3.2 ArcGIS - Datatypes

ArcGIS ondersteunt zowel datatypes in een database als *file-based* datatypes. Twee van de file-based datatypes zijn *coverages* en *shapefiles*. In deze types kan vector data opgeslagen worden, hierbij worden unieke indentifiers gebruikt om gegevens te linken aan hun attributen die in andere binaire bestanden zijn opgeslagen. Bij het datatype gebaseerd op een database, kortweg een *geodatabase* genoemd, worden zowel de spatial data als de attributen in één rij van een tabel opgeslagen. In deze paragraaf worden eerst de coverages besproken, dan de shapefiles en als laatste de geodatabase. Andere datatypes die door ArcGIS ondersteund worden, maar die niet aan bod komen, zijn *grids* (afbeeldingen in verschillende formaten), *Triangulated Irregular Networks* (TINs) en *Computer Aided Design* (CAD) bestanden.

#### 5.3.2.1 Coverages

Coverages worden gebruikt om vector data en hun onderlinge topologische relatie op te slaan. Een coverage is opgebouw uit *primaire*, *samengestelde* en *secundaire* featuretypes. Eén coverage kan meerdere types bevatten.

De primaire features komen overeen met die van het topologische model uit paragraaf 2.2.4.6 op pagina 26. Deze features zijn: *Label points*, die zowel individuele punten kunnen voorstellen als attributen aan polygonen linken. Dit wordt geïllustreerd in figuren 5.9(a) en 5.9(b). *Nodes* als eindpunten van *arcs*, hiemee kan een alleenstaande breuklijn voorgesteld worden of een netwerk, wanneer verschillende arcs gegroepeerd worden. Dit is te zien in figuren 5.9(c) en 5.9(d). Een laatste primaire feature zijn de *polyons*, die gedefinieerd zijn door arcs. Pologons kunnen een arc delen, maar ze kunnen elkaar niet overlappen. Een voorbeeld hiervan is in figuur 5.9(e) te zien.

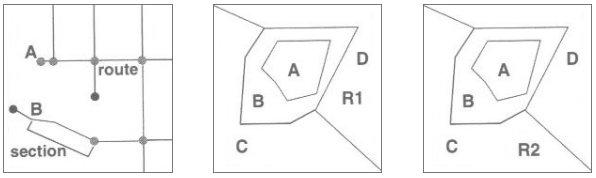



(a) Points (b) Labels (c) Arc (d) Genetwerkte arcs (e) Polygons

Figuur 5.9: Primaire features in coverages (bron [30]).

De samengestelde features *routes* en *sections* zijn lineaire features die opgebouwd zijn uit arcs en delen van arcs. Routes definiëren paden op een

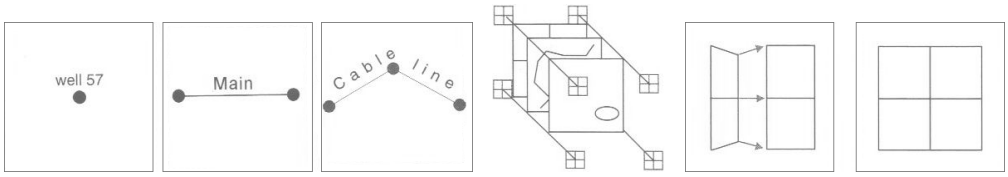
bestaand netwerk en sections identifceren delen van arcs. In figuur 5.10(a) is er een route gedefinieerd van de node $A$ tot het eindpunt van een sectie $B$. *Regions* zijn opgebouwd uit polygons, in tegenstelling tot polygons kunnen regions uit verschillende, niet aaneengesloten gebieden bestaan en mogen ze overlappen. Figuur 5.10(b) toont de regio $R1$ die bestaat uit de polygonen $A$ en $D$ en figuur 5.10(c) toont regio $R2$ die bestaat uit $A$ en $C$.




(a) Route / Section (b) Regio R1 (c) Regio R2

Figuur 5.10: Samengestelde features in coverages (bron [30]).

De secondaire features zijn: *annotation*, hiermee kan er tekst bij een punt, tussen twee punten of langs een serie van punten gezet worden. De locatie van de tekst wordt opgeslagen met behulp van geografische coördinaten. Zie figuur 5.11(a) voor een voorbeeld. *Tics* zijn geografische controlepunten die een gekende locatie voorstellen, hiermee is het mogelijk om verschillende layers correct over elkaar te positioneren. Dit wordt geïllustreerd in figuur 5.11(b). De laatste secondaire feature is een *link*, dit is een verplaatsingsvector die gebruikt wordt om de randen van twee aangrenzende coverages aan te laten sluiten. Figuur 5.11(c) illustreert dit.




(a) Annotatie (b) Tics (c) Links

Figuur 5.11: Secundaire features in coverages (bron [30]).

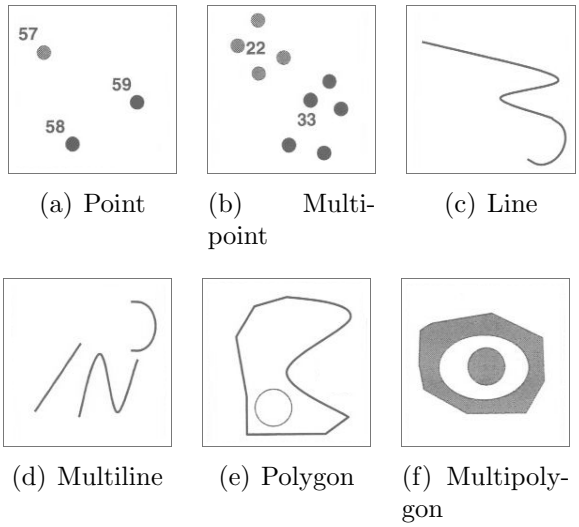
#### 5.3.2.2 Shapefiles

Shapefiles worden gebruikt om vector data op te slaan, hierbij worden de topologische relaties *niet* opgeslagen. Dit zorgt ervoor dat dit formaat minder

geschikt is voor topologische analyses, maar het is toch mogelijk, weliswaar minder efficiënt. Dit formaat wordt gebruikt voor de brongegevens van de case study in volgend hoofdstuk en worden daar in paragraaf 6.1.1 verder toegelicht.

#### 5.3.2.3 Geodatabases

Met geodatabases kunnen zowel vector, raster als alfanumerieke gegevens opgeslagen worden via objecten in een relationele database. Voor een single-user geodatabase wordt MS Access gebruikt terwijl een multi-user geodatabase geïmplementeerd is in uitgebreidere commerciële databases zoals DB2, InforMix, SQL Server of Oracle. Hier wordt alleen vector data besproken. De standaard features in een geodatabase zijn *points*, *lines* en *polygons*, deze kunnen allen uit meerdere delen bestaan. De lines en de polygons kunnen uit rechte lijnsegmenten, circulaire bogen en/of Bézier splines bestaan. Zie figuur 5.12 voor voorbeelden van deze features.




(a) Point (b) Multipoint (c) Line

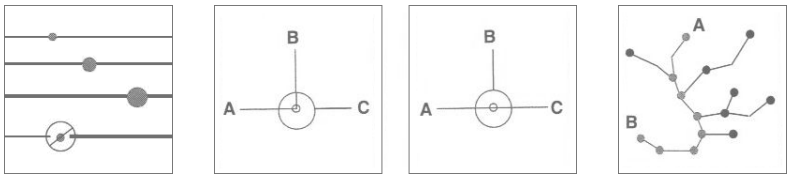
(d) Multiline (e) Polygon (f) Multipolygon

Figuur 5.12: Standaard features in geodatabases (bron [30]).

Naast deze standaard features zijn er ook nog *custom* features, hierbij kan de weergave van het object gedefinieerd zijn afhankelijk van bepaalde parameters, zoals op welke schaal het object bekeken wordt. Het is ook mogelijk om een zelfgedefinieerde interface te creëren, zo kan er bijvoorbeeld informatie over de eigenaar van een gebouw getoond worden of zelfs de plattegrond ervan. Twee van deze custom features zijn voorgedefinieerd, namelijk de *network junctions* en de *network edges*. Een simpele network junction kan een

fitting voorstellen die twee waterleidingen verbindt, hierbij wordt er gecontroleerd of de edges/leidingen die aangesloten worden wel de juiste diameter hebben. Een complexe junction kan zelfs interne onderdelen hebben, zoals een schakeling die bepaalt welke edges van het netwerk op een gegeven moment met elkaar verbonden zijn. Een simpele junction wordt getoond in figuur 5.13(a) en een complexe in figuur 5.13(b).

Een simpele network edge verbindt twee junctions en controleert of er voldaan wordt aan de verbindingsvoorwaarden, in het geval van waterleidingen of ze de juiste diameter hebben. Een complexe edge kan meerdere junctions bevatten en toch één enkele feature zijn. Iedere juction en (deel-)edge hiervan kan een custom feature zijn met een eigen interface en/of restrictie. In figuur 5.13(c) wordt een complexe edge getoond die begint bij $A$ en eindigt in $B$.




(a) Simpele junction (b) Complexe junction (c) Complexe edge

Figuur 5.13: Network features in geodatabases (bron [30]).

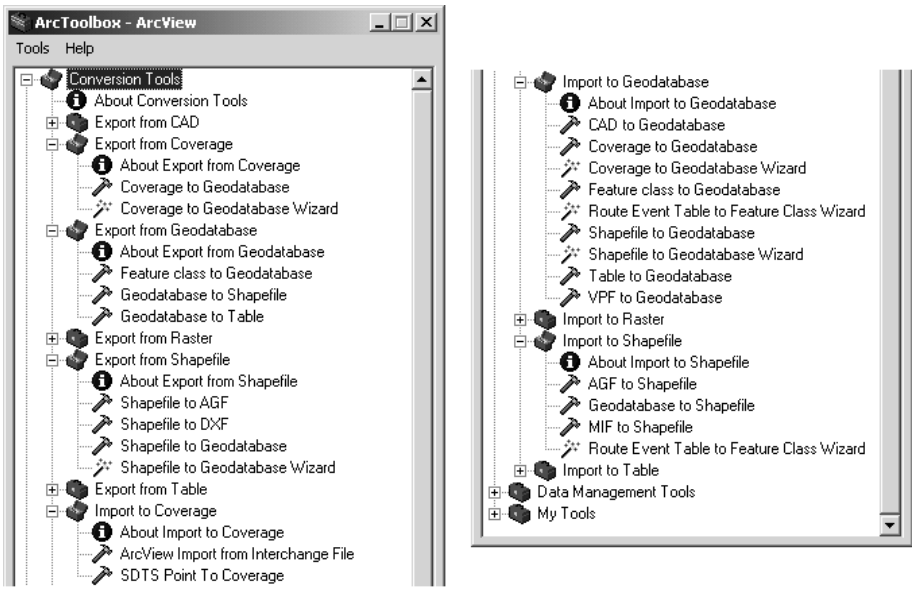
In geodatabases zijn er twee vormen van topologie mogelijk: *planar topologies* of *geometric networks*. Een planar topology is gebaseerd op *simpele* features (geen annotaties, samengestelde of network features) en bepaalt de relatie daartussen. Deze topology is opgebouwd uit *edges*, *nodes* en *pseudonodes*. Edges bepalen lines en/of polygons, nodes bepalen eindpunten van edges en pseudonodes zijn nodes die maar twee edges verbinden. In een topology kunnen *line features* eindpunten, edges en segmenten van andere line features delen, *area features* kunnen grenzen en polygonen van andere area features en edges van line features delen, en *point features* kunnen eindpunten en middenpunten van line features delen. Een geometric network is een logisch netwerk dat opgebouwd is uit de custom features network junctions en network edges. Hierbij is het mogelijk om extra attributen en/of restricties toe te voegen, zoals een kost voor het doorlopen van een edge. Met dit netwerk kan dan een analyse uitgevoerd worden, zoals het zoeken van verbonden features, het opsporen van lussen of het vinden van gemeenschappelijke voorouders.

### 5.3.3 ArcGIS - Functies

ArcGIS bevat verschillende functies/tools om spatial data te analyseren. Alle functies zijn beschikbaar via een GUI door middel menu's en/of *wizards*. Deze functies zijn verspreid terug te vinden in ArcMap, ArcToolbox en ArcCatalog. De spatial functies kunnen ingedeeld worden in: (1) Het converteren van geometrieën naar en van andere dataformaten. (2) Het vergelijken van geometrieën op topologische relaties en afstanden. (3) Het bewerken en opvragen van informatie over eigenschappen van geometrieën. (4) Het creëren van nieuwe geometrieën aan de hand van bestaande geometrieën.

#### 5.3.3.1 Conversie tools

Deze tools zijn terug te vinden in ArcToolbox en gedeeltelijk in ArcCatalog, ze laten toe om datasets te converteren van en naar verschillende data formaten. Omdat ArcGIS, in tegenstelling tot DB2 en Oracle, geen centraal spatial datatype heeft, zijn er per type meerdere conversie functies naar andere types. Merk hierbij op dat niet tussen elke twee types geconverteerd kan worden. In ArcToolbox staan alle tools samen gegroepeerd, een (opgesplitst) screenshot hiervan is te zien in figuur 5.14. Voor sommige tools bestaat er ook een wizard die hetzelfde doet als de tool, maar waarbij stap voor stap de parameters ingegeven moeten worden.

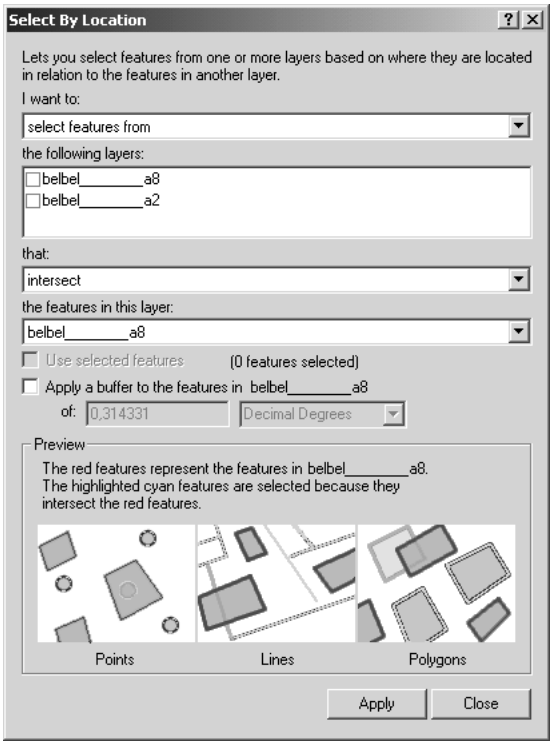



Figuur 5.14: ArcGIS - ArcToolbox.

In ArcCatalog kan men de tools en wizards bereiken door rechts te klikken op de dataset die geconverteerd moet worden en dan exporteren te selecteren. In het export-menu dat dan uitklapt, worden dan de tools die bij het bron datatype horen getoond. Importeren via ArcCatalog is gelijkaardig, maar is alleen mogelijk naar een geodatabase.

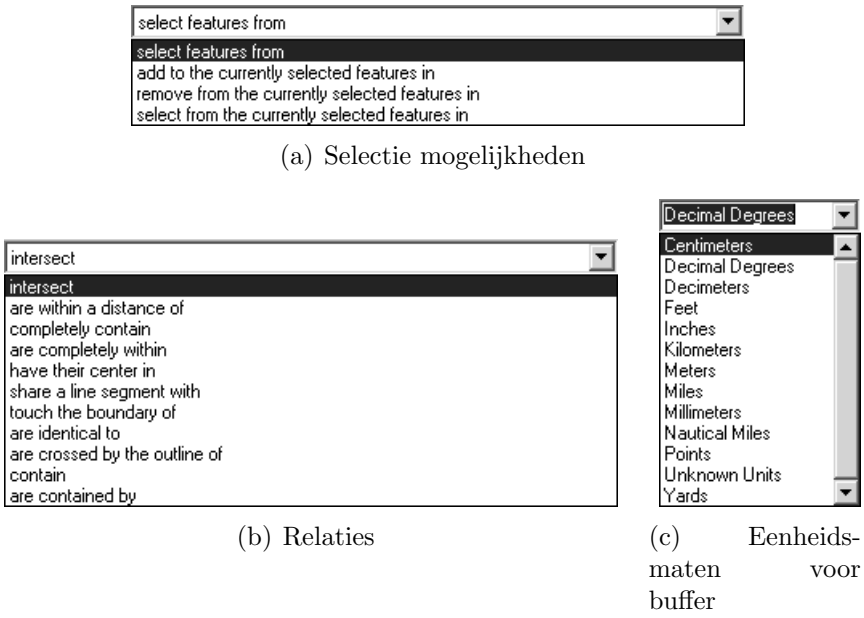
#### 5.3.3.2 Functies om geometrieën te vergelijken

Deze functies bepalen de topologische realtie en/of afstand tot een andere geometry. De functies worden door ArcMap gebruikt om geometrieën te selecteren uit een layer en kunnen niet rechtstreeks uitgevoerd worden door de gebruiker. Om geometrieën te selecteren die voldoen aan een bepaalde relatie, klik in de 'Hoofdmenu' werkbalk op 'Selection' en kies voor 'Select By Location ...'. Dan verschijnt er een venster zoals in figuur 5.15. Hierin kan opgegeven worden *hoe* geselecteerd moet worden, *uit welke layer* geometrieën geselecteerd moeten worden, wat de *relatie* hiertussen is, welke de *bron layer* is, of alleen de reeds geselecteerde geometrieën van de bron layer gebruikt mogen worden en of een *buffer* gecreëerd moet worden.




Figuur 5.15: Select by location in ArcGIS.

De mogelijkheden *hoe* te selecteren worden getoond in figuur 5.16(a), volgens welke *relatie* er geselecteerd kan worden in figuur 5.16(b) en de eenheidsmaten voor een *buffer* in figuur 5.16(c).




(a) Selectie mogelijkheden

(b) Relaties

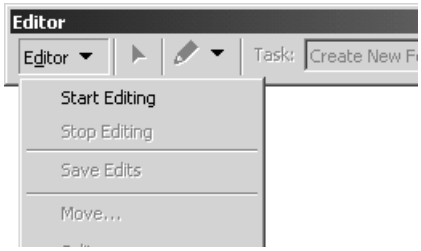
(c) Eenheidsmaten voor buffer

Figuur 5.16: Opties bij: Select by location.

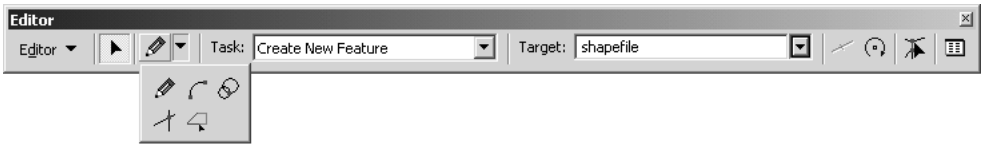
#### 5.3.3.3 Bewerken en opvragen van informatie van geometrieën

Met ArcMap is het mogelijk om nieuwe geometrieën te creëren en om coördinaten van bestaande geometrieën op te vragen en/of te bewerken. Hiervoor moet eerst een layer geladen worden, dit kan een bestaande dataset zijn of er kan een nieuwe layer aangemaakt worden (voor nieuwe geometrieën). Voeg dan de *Editor* werkbalk toe en klik op 'Start Editing', zie figuur 5.17(a). Indien er meerdere layers actief zijn moet hieruit de gewenste layer geselecteerd worden. Om een nieuwe geometry te creëren moet eerst een *sketch* getekend worden met behulp van één van de edit-tools, zie figuur 5.17(b). Hierbij biedt het menu onder een rechter muis klik ook enkele opties, zoals het instellen van een hoek, lengte, ... tussen twee punten. Eén sketch kan uit meerdere geometrieën bestaan die, als ze *gefinaliseerd* worden, één enkele geometry vormt. Een afgewerkte sketch of een reeds bestaande geometry kan bewerkt worden door eerst de gewenste geometry te selecteren en dan de taak 'Modify Feature' te kiezen. Tijdens het bewerken van een geometry kunnen ook

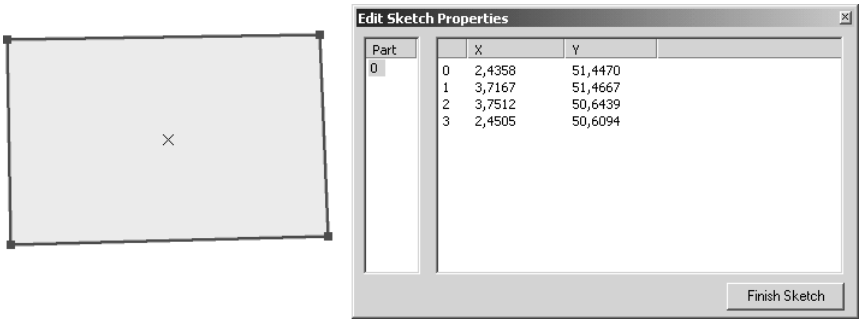
zijn coördinaten opgevraagd worden door rechts te klikken op één van zijn (hoek)punten of de verbindingslijnen daartussen en 'Proporties' te kiezen. Dit zorgt voor een venster met links een lijst van geometrieën en rechts een lijst van de (hoek)punten waaruit de geselecteerde geometry is opgebouwd. Zie figuur 5.17(c) voor een voorbeeld hiervan. Nadat de gewenste wijzigingen uitgevoerd zijn en alle sketches gefinaliseerd zijn, moeten de wijzigingen opgeslagen worden en moet er op 'Stop Editing' geklikt worden.




(a) Start editing




(b) Editor tools



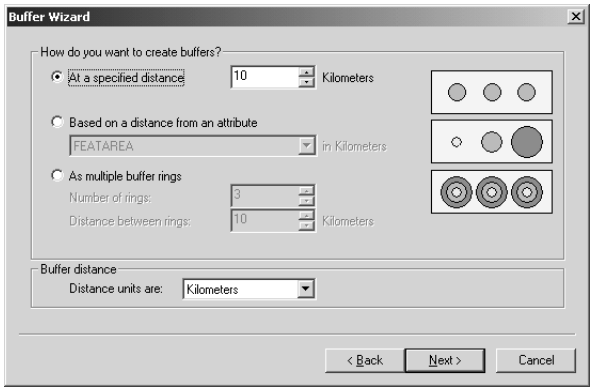

(c) Coördinaten van een sketch

Figuur 5.17: ArcGIS Editor.

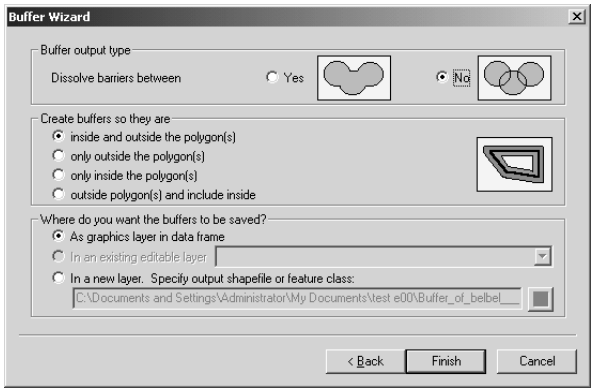
#### 5.3.3.4 Tools die nieuwe geometrieën genereren

Deze laatste groep tools laat toe om nieuwe geometrieën af te leiden van bestaande. Deze tools zijn terug te vinden in ArcMap onder het menu 'Tools' en in de ArcToolbox (alleen voor ArcInfo). In ArcMap zijn de tools in twee

wizards verwerkt, namelijk de *Buffer Wizard* en de *GeoProcessing Wizard*. Om een buffer rond een geometry te creëren moet eerst de gewenste layer opgegeven worden en eventueel een selectie gemaakt worden in deze layer. Dan kan de breedte en het aantal buffer-ringen opgegeven worden, zie figuur 5.18(a). En of buffers die overlappen samengevoegd moeten worden, aan welke zijde van de geometry de buffer gecreëerd moet worden en tenslotte in welke layer het resultaat moet worden opgeslagen, zie figuur 5.18(b). De GeoProcessing wizard combineert verschillende tools, hiermee is het mogelijk om features in een layer samen te voegen (*dissolve*), layers te *mergen*, een layer te *clippen*, de doorsnede te nemen van twee layers en de unie te nemen van twee layers, zie figuur 5.19.

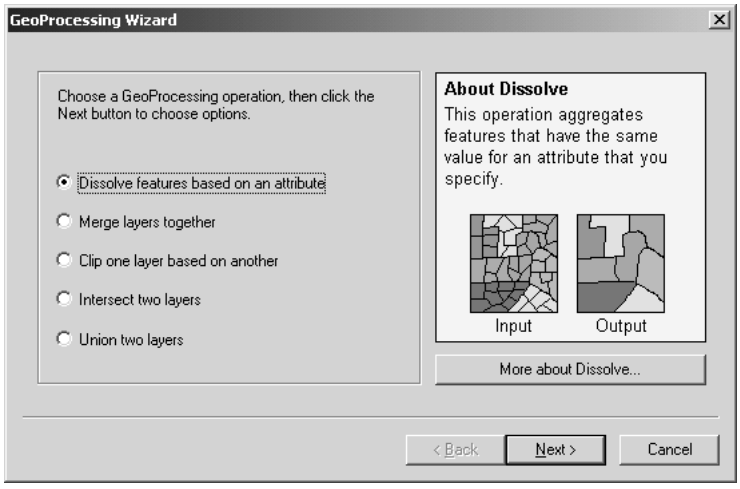



(a) Buffer (stap 1)

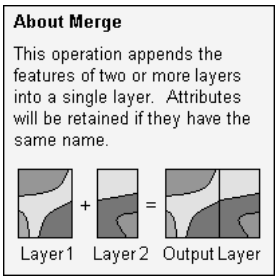



(b) Buffer (stap 2)

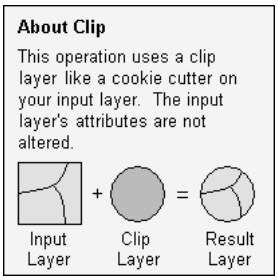
Figuur 5.18: ArcGIS buffer wizard.



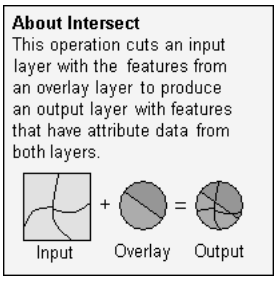

(a) Dissolve



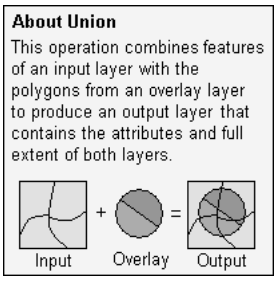

(b) Merge (detail)




(c) Clip (detail)




(d) Doorsnede (detail)




(e) Unie (detail)

Figuur 5.19: ArcGIS geoprocessing wizard.

# Hoofdstuk 6

# Case Study

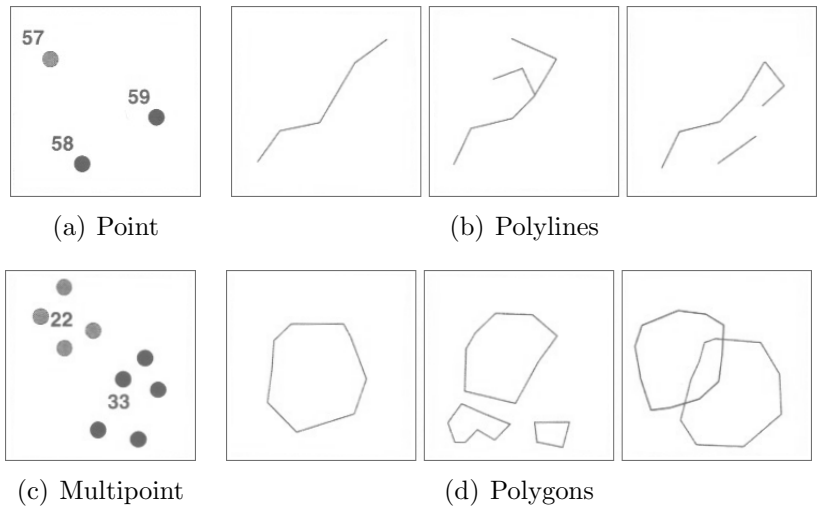
Dit hoofdstuk beschouwt de case study met *DB2 Spatial Extender*, *Oracle Spatial* en *ArcGIS*. Hierbij lichten we eerst de brongegevens toe waarmee er gewerkt wordt. Dan worden de referentie queries en het verwachte resultaat hiervan gegeven. Er wordt aangenomen dat ieder systeem al geïnstalleerd is en dat er, indien nodig, een gebruiker met voldoende rechten is aangemaakt. Dan wordt getoond hoe de benodigde geografische en alfanumerieke gegevens ingeladen kunnen worden. Tenslotte kan er met de case study begonnen worden, hierbij wordt voor iedere query besproken *hoe* deze uitgevoerd moet worden (indien de query al ondersteund wordt), wat het *resultaat* ervan is en welke *problemen* hierbij voorgekomen zijn.

## 6.1 Brondata

Voor het uitvoeren van de case study maken we gebruik van geografische gegevens van Tele Atlas [24]. Tele Atlas levert high-end gegevens die een aanzienlijke oppervlakte dekken, compleet zijn, een hoge nauwkeurigheid hebben en vrij recent zijn (eind 2003). Deze data is opgedeeld in verschillende layers die opgeslagen zijn in het *shapefile* formaat. In deze case study wordt gewerkt met de geografische gegevens van België; gegevens over de rest van West Europa en de Verenigde Staten zijn, indien gewenst, ook beschikbaar bij Tele Atlas. Voor de case study is er een selectie gemaakt uit het grote aantal beschikbare layers, zoals het netwerk van wegen, waterlopen, landgebruik, adresgegevens, administratieve gebieden, enzovoorts. Een volledig overzicht van alle beschikbare layers is te vinden in [34].

### 6.1.1 Shapefiles

De brongegevens waarmee in deze case study gewerkt wordt zijn van het shapefile formaat dat ontwikkeld werd door ESRI [4]. Eén layer komt overeen met één shapefile, hierin worden geografische gegevens relatief eenvoudig opgeslagen. Een object bestaat enkel uit een set van vector coördinaten en er worden geen topologische relaties tussen objecten bijgehouden. Omdat alleen de 'hoekpunten' van features worden bijgehouden zijn shapefiles meestal kleiner, worden sneller getekend en zijn gemakkelijker te bewerken dan geografische data in andere formaten. Shapefiles ondersteunen *points*, *multipoints*, *polylines* en *polygons*. Polylines mogen uit verschillende delen bestaan en deze delen hoeven niet met elkaar verbonden te zijn. Ook polygons mogen uit verschillende delen bestaan en mogen zelfs overlappen, deze overlapping wordt echter niet specifiek opgeslagen in een topologische relatie. Deze types worden geïllustreerd in figuur 6.1. Voorlopig kan iedere shapefile echter maar één van bovenstaande types bevatten, deze restrictie wordt in de toekomst eventueel nog weggewerkt door ESRI. De huidige shapefiles ondersteunen dus ruwweg dezelfde geometrieën als de geometrieën in [17] die door het OpenGIS Consortium gedefinieerd zijn (zie ook paragraaf 2.2.3 op pagina 14). Behalve geografische gegevens kunnen er ook alfanumerieke gegevens in shapefiles opgeslagen worden voor de attributen die bij de ruimtelijke objecten horen.

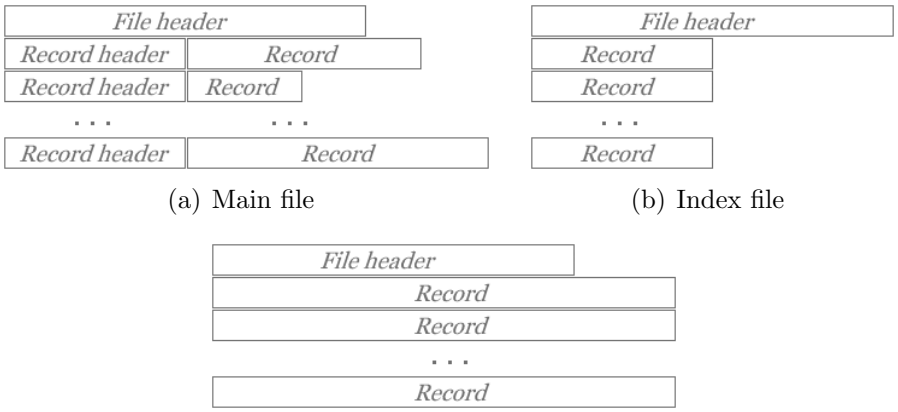



(a) Point (b) Polylines

(c) Multipoint (d) Polygons

Figuur 6.1: Shapefile geometrieën (bron [30]).

Eén shapefile bestaat uit verschillende bestanden met telkens dezelfde prefix. Hiervan zijn de *Main file* (.shp), de *Index file* (.shx) en de *dBASE file* (.dbf) verplichte bestanden. De main file bevat records van variabele lengte met in ieder record een geometry met zijn lijst van vector coördinaten (zie figuur 6.2(a)). De index file bevat voor ieder record in de main file zijn offset vanaf het begin van de main file en zijn lengte (zie figuur 6.2(b)). De dBASE file bevat de attributen van ieder record in de main file. Hierbij is de één-op-één relatie tussen deze bestanden gebaseerd op het record nummer. De records met attributen moeten dus in dezelfde volgorde staan als de records in de main file (zie figuur 6.2(c)).




(a) Main file (b) Index file

(c) dBASE file

Figuur 6.2: Shapefile structuur (bron [4]).

Behalve deze verplichte bestanden bestaan er ook nog een aantal optionele bestanden die meestal alleen door ArcView (de standaard versie van ArcGIS) gebruikt worden [10]. Deze zijn de *Projection file* (.prj), de *Legend file* (.avl), de *Metadata file* (.xml), de *Spatial Index files* (.sbn en .sbx) (*of* .fbn en .fbx als de main file read-only is), de *Attribute Index files* (.ain en .aih) en de *Geocoding Index files* (.ixs en .mxs). Van deze optionele bestanden is alleen de projection file voorzien bij de brongegevens, deze bevat de gebruikte projectie en datum. De overige bestanden zijn niet aanwezig bij de brondata. De legend file bevat een voorgedefinieerde kleurenlegende voor een bepaalde layer. De metadata file is nodig om shapefiles op het internet te gebruiken. De index bestanden tenslotte worden door ArcView gecreëerd indien er een index nodig is, zoals bij het uitvoeren van een spatial join, theme-on-theme selectie, link tables of create index commando. Bij een read-only bron directory worden deze index bestanden verwijderd als ArcView afgesloten wordt.

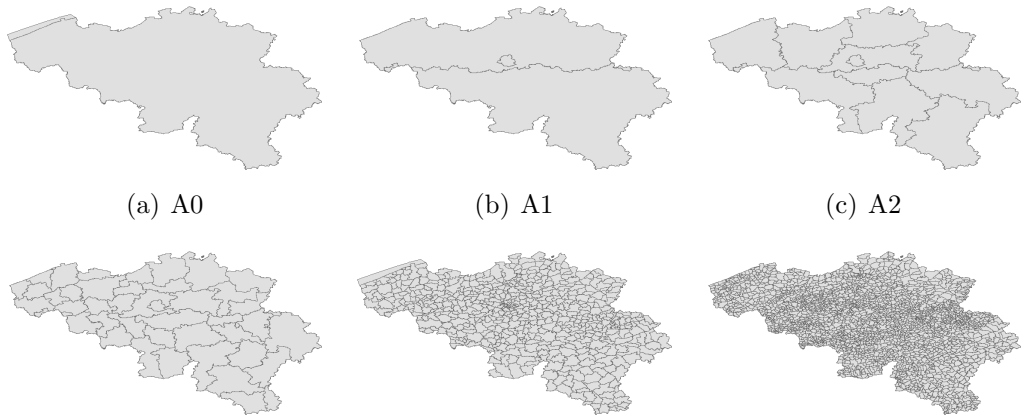
### 6.1.2 Gebruikte data

Nu we weten dat een shapefile uit meerdere bestanden bestaat en wat de functie van ieder bestand is, worden de gebruikte shapefiles naderbij bekeken. Deze bevatten gegevens over het **wegennetwerk** en de **administratieve gebieden** in België. Het is niet omdat het hier maar over twee onderwerpen gaat, dat er ook maar twee (sets van) shapefiles voor zijn. Elk van de twee onderwerpen is onderverdeeld in verschillende niveaus die elk in een aparte layer zitten. De administratieve gebieden van België bestaan uit tien niveaus die in de shapefiles 'belbel_____a0' tot 'belbel_____a9' opgeslagen zijn en het wegennetwerk bestaat uit negen niveaus in de shapefiles 'belbel_____00' tot 'belbel_____08'. Voor de eenvoud wordt in het volgende alleen met de laatste twee letters naar de layers verwezen (A0, A1 ... A9 en 00, 01 ... 08). Het hoogste niveau bevindt zich in de shapefile met het laagste nummer en het laagste niveau heeft het hoogste nummer. Alhoewel er voor de administratieve gebieden tien niveaus beschikbaar zijn worden er maar zes van gebruikt, layer A3 tot en met A6 zijn namelijk leeg. Voor het wegennetwerk worden wel alle niveaus gebruikt. Een overzicht van de indeling in layers is te zien in tabel 6.1.

| Bestandsnaam | Omschrijving |
|---|---|
| belbel_____a0 | Country |
| belbel_____a1 | Region |
| belbel_____a2 | Province |
| belbel_____a7 | District |
| belbel_____a8 | Munucipalty |
| belbel_____a9 | Sub-municipality |
| | |
| belbel_____00 | Motorway |
| belbel_____01 | Main Road |
| belbel_____02 | Other Major Road |
| belbel_____03 | Secondary Road |
| belbel_____04 | Local Connecting Road |
| belbel_____05 | Local Road of High Importance |
| belbel_____06 | Local Road |
| belbel_____07 | Local Road of Minor Importance |
| belbel_____08 | Other |

Tabel 6.1: Overzicht layers brondata.

In de layers van de administratieve gebieden zijn enkel polygonen gebruikt en voor het wegennetwerk zijn er alleen polylines gebruikt. Maar naast geometrieën bevat iedere layer ook nog alfanumerieke gegevens. Voor de administratieve gebieden zijn dit: zijn *'ID'*, zijn *feature type* ('FEATTYP'), zijn *naam* ('NAME'), zijn *taalcode* ('NAMELC'), zijn *oppervlakte* in $m^2$ ('FEAT-AREA'), zijn *omtrek* in $m$ ('FEATPERIM') en één tot tien *administratieve codes* ('ORDER00' tot 'ORDER09'). Voor de layers A8 en A9 komt hierbij nog hun *populatie* ('POP') en een *populatieklasse* ('POPCLASS'). Het ID en het feature type identificeren het object en zijn type. De naam en de taalcode bevatten de officiële naam en taal van het gebied. De oppervlakte en omtrek zijn triviaal. De administratieve code zorgt voor de hiërarchie van de gebieden, het hoogste niveau heeft zo maar één code, het tweede niveau heeft twee codes, enzovoorts. Alle gebieden die volgens een hoger niveau samen horen hebben dezelfde code voor dat niveau. Een voorbeeld, Limburg bevindt zich in layer A2 en heeft '7' als code van de $3^e$ orde, omdat Limburg bij Vlaanderen hoort en Vlaanderen code '1' heeft, heeft Limburg '1' als code van de $2^e$ orde. De code van de $1^e$ orde voor Limburg is 'BEL' omdat dit ook de code van België is. De populatie bevat het aantal inwoners van een gebied en de populatieklasse tenslotte bevat een getal van 0 tot 6 dat aangeeft in welk inwoners-bereik dit ligt. In figuur 6.3 is een overzicht van de verschillende administratieve gebieden te zien.



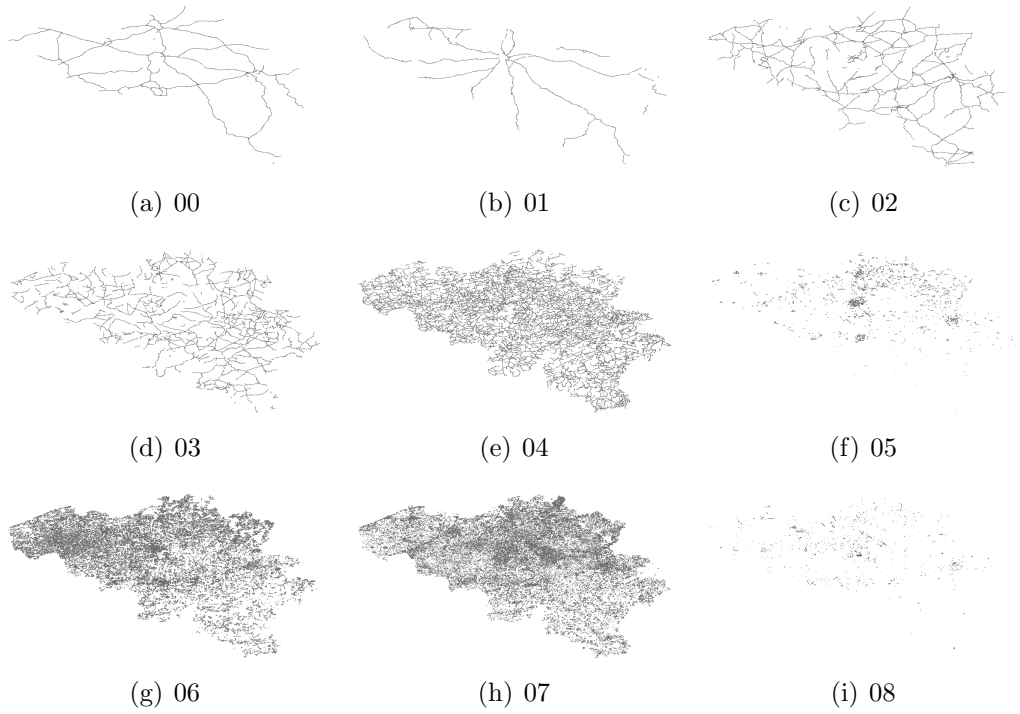
(a) A0 (b) A1 (c) A2

(d) A7 (e) A8 (f) A9

Figuur 6.3: Overzicht administratieve gebieden.

Net zoals bij de administratieve gebieden bevat iedere layer van het wegennetwerk een aantal alfanumerieke velden: zijn *functional road class* ('FRC'),

zijn *naam* ('NAME'), zijn *taalcode* ('NAMELC'), zijn *wegnummer* ('ROUTENUM'), zijn *wegtype* ('RTETYP'), zijn *richting* ('RTEDIR') en zijn *richtings validatie* ('RTEDIRVD'). De functional road class geeft aan tot welk niveau het object hoort, deze verdeling is volgens de grootte van de weg gedaan voor heel België in zijn geheel. De naam en de taalcode bevatten de officiële naam en taal van de weg. Het route nummer bevat de code van de weg, bijvoorbeeld 'E40'. Het wegtype bevat nog een extra indeling van de wegen in klasse één tot en met elf. De laatste twee velden worden alleen voor gegevens van de Verenigde Staten gebruikt en geven de richting aan waarin de weg loopt (noord, oost, zuid of west) en of het verkeer er in de positieve of negatieve zin over moet. In figuur 6.4 is een overzicht van de verschillende wegenniveaus te zien en figuur 6.5 toont een combinatie van alle wegen.



(a) 00 (b) 01 (c) 02

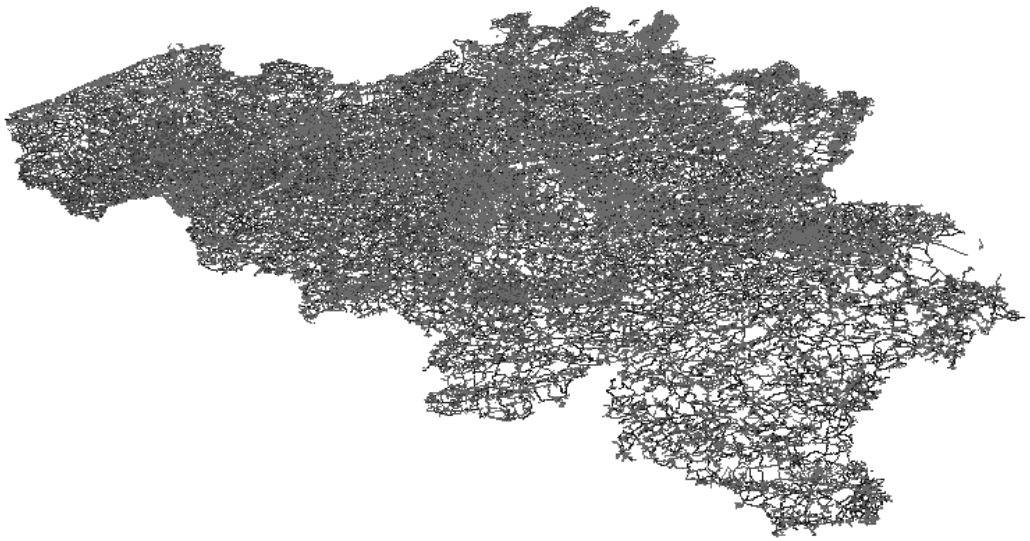
(d) 03 (e) 04 (f) 05

(g) 06 (h) 07 (i) 08

Figuur 6.4: Overzicht wegenniveaus.

## 6.2 Queries

De case study omvat twaalf queries die in drie categorieën in te delen zijn, namelijk alfanumerieke queries, spatial queries en interactieve queries. De



Figuur 6.5: Combinatie alle wegenniveaus.

eerste categorie bevat queries die alleen betrekking hebben tot de niet-spatial gegevens, hierbij wordt dus alleen gebruik gemaakt van de data in de alfanumerieke velden. De queries in de tweede categorie maken wel gebruik van de spatial gegevens, hierbij moet er dus een spatial operatie of predikaat gebuikt worden. Bij de laatste categorie wordt een spatial input van de gebruiker verwacht. De bedoeling van deze queries is te onderzoeken *welke* queries mogelijk zijn met de verschillende systemen en *hoe* deze queries uitgevoerd kunnen worden. Voor de duidelijkheid wordt voor iedere query aangegeven tot welke categorie hij behoort. Alfanumerieke queries worden met **alf** aangeduid, spatial queries met **spat** en interactieve queries met **inter**.

*Query 1* Bepaal het aantal gemeentes in België (**alf**).
*Query 2* Bepaal het aantal inwoners in Diepenbeek (**alf**).
*Query 3* Bepaal het aantal inwoners in Limburg (**alf**).
*Query 4* Geef een lijst van alle gemeentes in Limburg (**alf**).
*Query 5* Bepaal de gemeentes grenzend aan Diepenbeek (**spat**).
*Query 6* Output[8] de gemeente Diepenbeek (**spat**).
*Query 7* Bepaal de gemeentes in Antwerpen die groter zijn dan de

[8]Visueel in ArcGIS, als tekst in DB2 Spatial Extender en Oracle Spatial.

| | |
|---|---|
| | grootste gemeente in Limburg door middel van de oppervlakte gegeven in de attributen (**alf**). |
| *Query 8* | Bepaal de gemeentes in Antwerpen die groter zijn dan de grootste gemeente in Limburg door de oppervlakte te berekenen van de geometry (**spat**). |
| *Query 9* | Bepaal alle autostrades (*RouteNum* LIKE 'E%') die door Diepenbeek lopen (**spat**). |
| *Query 10* | Bepaal alle gemeentes die maximaal 500 meter van de autostrade 'E313' liggen (**spat**). |
| *Query 11* | Geef informatie over de gemeente die aangeklikt wordt op het scherm (**inter**). |
| *Query 12* | Geef informatie over de gemeente die het volgende punt bevat: (5.39229169963549, 50.9270236262395)(**spat**). |

Bij **query 1** moet simpelweg het aantal verschillende gemeentes in A8 geteld worden.

Het resultaat van **query 2** kan uitgelezen worden uit het populatie attribuut van Diepenbeek in A8.

Bij **query 3** moet de som genomen worden van de populatie attributen in A8 waarvan hun administratieve code overeenkomt met die van Limburg.

**Query 4** heeft hetzelfde predikaat als query 3, maar nu moeten de namen van alle gemeentes teruggegeven worden.

**Query 5** is de eerste spatial query van de case study en het resultaat hiervan moet alle gemeentes bevatten die grenzen aan Diepenbeek.

**Query 6** vraagt om de gemeente Diepenbeek als output te leveren. De bedoeling hiervan is om te kijken welke output data types/visualisatie mogelijkheden ieder systeem bied.

Bij **query 7** moeten eerst alle gemeentes van Limburg bepaald worden en hieruit moet de grootste genomen worden (volgens hun attribuut-waarde). Daarna moeten alle gemeentes in Antwerpen geselecteerd worden en tenslotte mogen hiervan alleen die gemeentes in het resultaat komen die groter zijn (volgens hun attributen) dan de grootste in Limburg. Deze query is alfanumeriek en dient als voorbereiding op de volgende query.

**Query 8**, hierbij wordt hetzelfde predikaat gebruikt als in query 7, maar

nu moet de grootte van de gemeentes berekend worden met behulp van hun geometry.

Bij **query 9** moeten alle autostrades bepaald worden die doorheen Diepenbeek lopen, een weg wordt beschouwd als een autostrade als zijn wegnummer (*RouteNum*) van de vorm 'E%' is. Omdat in de data van Tele Atlas zulke wegen op alle niveaus kunnen voorkomen moet hiervoor in alle layers van het wegennetwerk gekeken worden. Een illustratie hiervan is het verbindingsstuk op een autostrade tussen de twee rijrichtingen.

**Query 10** is gelijkaardig aan de vorige query, alleen moeten hierbij de gemeentes geselecteerd worden in A8 waardoor de autostrade 'E313' met een buffer van 500 meter loopt. Ook hierbij moet in alle layers van het wegennetwerk gekeken worden om alle segmenten van de 'E313' te vinden.

De voorlaatste query —**query 11**— is interactief, hierbij moeten de attributen van de gemeente waarin geklikt wordt teruggegeven worden. Omdat alleen ArcGIS de geometrieën kan weergeven, kan er dus ook alleen in ArcGIS op een gemeente geklikt worden.

In **query 12** wordt de interactiviteit uit query 11 gesimuleerd voor DB2 en Oracle door te doen alsof er geklikt wordt in het punt (5.39229169963549, 50.9270236262395). Dit punt komt uit de layer *junctions* (van Tele Atlas) waarvan de $x$- en $y$-coördinaat opgevraagd werden. Dit punt ligt in Diepenbeek en bepaalt de eindpunten van twee wegsegmenten in de buurt van de Universitaire Campus.

## 6.3 Uitvoering case study

In dit deel wordt per systeem besproken hoe de brongegevens van Tele Atlas ingeladen kunnen worden en hoe de twaalf referentie queries uitgevoerd worden. Hierbij worden eveneens de moeilijkheden besproken die daarbij zijn opgedoken.

### 6.3.1 DB2 Spatial Extender

Het uitvoeren van de case study met *DB2 Spatial Extender* werd onder Linux op *Spooky* gedaan, dit is een server van het LUC/UHasselt met als specificaties: een dual Intel Xeon 3.20GHz systeem met 2GB werkgeheugen. Hierop is DB2 versie 8.2.3 met de Spatial Extender geïnstalleerd waarbij een gebrui-

ker met voldoende rechten is aangemaakt.

Om de brongegevens in te laden moeten de bestanden leesbaar zijn voor de 'others' (bestandspermissies: rwxr-xr-x) en moet de gebruikersaccount toelaten om een nieuwe tabel te creëren. Het importeren van de volledige shapefile *belbel_____00* gebeurt met het volgende commando:

```
call db2gse.ST_Import_Shape('/tmp/shapefiles/belbel_____00', null,
'WGS84_SRS_1003', null, 'str_00', null, null, null, 'geometry',
null, null, null, null, null, null, null, null, null, ?, ?);
```

Hierbij bevat de eerste parameter het pad naar de shapefile. De derde parameter (*srs_name*) bevat de naam van het spatial referentiesysteem. Deze naam moet bestaan in de tabel 'db2gse.ST_Spatial_Reference_Systems' en kan bepaald worden door te kijken welke naam uit de tabel het meeste overeenkomt met de naam in het *.prj* bestand, in dit geval GCS_WGS_1984. De vijfde parameter is de *tabel naam*, in dit geval 'str_00'. De negende parameter bepaalt de naam van de spatial kolom, hier 'geometry'. Voor de overige parameters wordt verwezen naar de *User manual* van *DB2 Spatial Extender* [7].

Met het import commando worden alle benodigde shapefiles ingeladen. Om dit commando niet telkens via de *command line* in te hoeven geven worden al deze commando's in een tekstbestand ingevoerd. Door dit bestand in DB2 uit te voeren met 'db2 -tvf *bestandsnaam*' worden alle statements in het bestand na elkaar uitgevoerd, voorwaarde is wel dat ieder commando met een puntkomma ';' beëindigd moet worden. Wanneer alle brongegevens ingeladen zijn, kunnen de queries uitgevoerd worden.

*Query 1: Bepaal het aantal gemeentes in België (**alf**).* Hiervoor wordt gewoon een *count* gedaan van het aantal verschillende rijen in A8.

Listing 6.1: DB2 Spatial Extender - Query 1

```
Select Count(Distinct(id)) As Aantal_Gemeentes From adm_a8;
```

Het resultaat van deze query is een tabel met één kolom, deze heeft als naam 'AANTAL_GEMEENTES' en één rij met de waarde '590'.

*Query 2: Bepaal het aantal inwoners in Diepenbeek (**alf**).* De oplossing hiervan is terug te vinden in de kolom *pop* van A8 van het tuppel dat 'Diepenbeek' als *name* heeft.

Listing 6.2: DB2 Spatial Extender - Query 2

```
Select pop From adm_a8 Where name = 'Diepenbeek';
```

Het resultaat hiervan is de kolom 'POP' met als waarde '16899'.

*Query 3: Bepaal het aantal inwoners in Limburg (**alf**).* Om deze query op te lossen moet in A8 de som genomen worden van de waardes in het veld *pop* voor alle tuppels die tot Limburg horen. Een tuppel van A8 hoort bij Limburg als zijn veld *order02* overeenkomt met het veld *order02* van een tuppel in A2 waarvan de *name* gelijk is aan 'Limburg'.

Listing 6.3: DB2 Spatial Extender - Query 3

```
Select Sum(a8.pop) As Pop_Limburg From adm_a2 A2, adm_a8 A8
Where A2.name = 'Limburg' And A2.order02 = A8.order02;
```

Als resultaat heeft deze query de tabel met als kolomnaam 'POP_LIMBURG' met één rij met de waarde '785228'.

*Query 4: Geef een lijst van alle gemeentes in Limburg (**alf**).* Hiervoor moet de *name* van alle tuppels teruggegeven worden die tot Limburg horen.

Listing 6.4: DB2 Spatial Extender - Query 4

```
Select A8.name From adm_a2 A2, adm_a8 A8
Where A2.name = 'Limburg' And A2.order02 = A8.order02;
```

Het resultaat van deze query is de kolom 'NAME' met als waardes (in de volgende volgorde): Peer, Neerpelt, Diepenbeek, Zutendaal, Bilzen, Fourons, Houthalen-Helchteren, Herk-de-Stad, As, Bocholt, Lummen, Tongeren, Gingelom, Zonhoven, Lommel, Herstappe, Leopoldsburg, Nieuwerkerken, Borgloon, Ham, Bree, Maaseik, Overpelt, Hamont-Achel, Riemst, Hoeselt, Halen, Hasselt, Opglabbeek, Kortessem, Maasmechelen, Wellen, Alken, Dilsen-Stokkem, Heers, Meeuwen-Gruitrode, Genk, Kinrooi, Tessenderlo, Hechtel-Eksel, Lanaken, Sint-Truiden, Beringen en Heusden-Zolder.

*Query 5: Bepaal de gemeentes grenzend aan Diepenbeek (**spat**).* Deze query vereist de spatial functie *ST_Touches* die controleert of twee polygonen elkaar raken.

Listing 6.5: DB2 Spatial Extender - Query 5

```
Select A8_b.name From adm_a8 A8_a, adm_a8 A8_b
Where db2gse.ST_Touches(A8_a.geometry, A8_b.geometry) = 1
      And A8_a.name = 'Diepenbeek';
```

Het resultaat hiervan is de kolom 'NAME' met de waardes: Bilzen, Hoeselt, Hasselt, Kortessem en Genk.

*Query 6: Output (als tekst/binair) de gemeente Diepenbeek (**spat**).* Bij deze

query moet de polygon van Diepenbeek in alle ondersteunde formaten teruggegeven worden. Dit kan in het Well-known text, het Well-known binary, het Geography Markup Language of het ESRI shape formaat.

Listing 6.6: DB2 Spatial Extender - Query 6

```
Select db2gse.ST_AsText(geom) From adm_a8 Where name = 'Diepenbeek';
Select db2gse.ST_AsBinary(geom) From adm_a8 Where name = 'Diepenbeek';
Select db2gse.ST_AsGML(geom) From adm_a8 Where name = 'Diepenbeek';
Select db2gse.ST_AsShape(geom) From adm_a8 Where name = 'Diepenbeek';
```

Iedere query heeft als resultaat de voorstelling van de polygon van Diepenbeek in het gewenste formaat.

*Query 7: Bepaal de gemeentes in Antwerpen die groter zijn dan de grootste gemeente in Limburg door middel van de oppervlakte gegeven in de attributen (**alf**).* Hiervoor wordt eerst de maximale oppervlakte bepaald van de gemeentes in Limburg. Vervolgens wordt deze oppervlakte vergeleken met de oppervlakte van de gemeentes in Antwerpen.

Listing 6.7: DB2 Spatial Extender - Query 7

```
Select A8_a.name, A8_a.featarea
From adm_a8 A8_a, adm_a8 A8_b, adm_a2 A2_a
Where A8_b.featarea In
        (Select Max(A8_c.featarea) From adm_a8 A8_c, adm_a2 A2_b
         Where A2_b.name = 'Limburg' And A2_b.order02 = A8_c.order02)
      And A8_a.featarea > A8_b.featarea
      And A2_a.name = 'Antwerpen' And A2_a.order02 = A8_a.order02;
```

Deze query heeft de volgende drie (gemeente, oppervlakte) tuppels als resultaat: (Mol, 114617666), (Antwerpen, 203548389) en (Geel, 110093679).

*Query 8: Bepaal de gemeentes in Antwerpen die groter zijn dan de grootste gemeente in Limburg door de oppervlakte te berekenen van de geometry (**spat**).* Dit is in principe dezelfde query als de vorige, met het verschil dat hier de oppervlakte berekend wordt met behulp van de spatial functie *ST_Area*.

Listing 6.8: DB2 Spatial Extender - Query 8

```
Select A8_a.name, A8_a.featarea, db2gse.ST_Area(A8_a.geometry) As Area
From adm_a8 A8_a, adm_a8 A8_b, adm_a2 A2_a
Where db2gse.ST_Area(A8_b.geometry) In
        (Select Max(db2gse.ST_Area(A8_c.geometry))
         From adm_a8 A8_c, adm_a2 A2_b
         Where A2_b.name = 'Limburg' And A2_b.order02 = A8_c.order02)
      And db2gse.ST_Area(A8_a.geometry) > db2gse.ST_Area(A8_b.geometry)
      And A2_a.name = 'Antwerpen' And A2_a.order02 = A8_a.order02;
```

Het resultaat hiervan is hetzelfde als bij de vorige query. Ter referentie wordt per gemeente zijn attribuut oppervlakte en de berekende oppervlakte getoond, de eerste is in $m^2$ uitgedrukt en de tweede in decimale graden (conversie naar $m^2$ is niet mogelijk met DB2 Spatial Extender).

| NAME | FEATAREA | AREA |
|---|---|---|
| Mol | 114617666 | +1.47505701948121E−002 |
| Antwerpen | 203548389 | +2.62104579733156E−002 |
| Geel | 110093679 | +1.41479459767488E−002 |

*Query 9: Bepaal alle autostrades (RouteNum LIKE 'E%') die door Diepenbeek lopen (**spat**).* Omdat autostradesegmenten in alle negen layers van het wegennetwerk kunnen voorkomen, is het voor deze query nodig dat al deze autostrades geïdentificeerd worden. Dit kan gedaan worden door voor iedere layer de unie te nemen van de autostrades die in het resultaat zitten. Om te controleren welke autostrade-segmenten door Diepenbeek lopen kan de spatial functie *ST_Intersects* gebruikt worden.

Listing 6.9: DB2 Spatial Extender - Query 9

```
Select s.FRC, s.NAME, s.NAMELC, s.ROUTENUM, s.RTETYP,
       s.RTEDIR, s.RTEDIRVD
From str_00 s, adm_a8
Where db2gse.ST_Intersects(s.geometry, adm_a8.geometry) = 1
        And adm_a8.name = 'Diepenbeek' And routenum Like 'E%'
  Union All
Select ... From str_01 s, ...
  Union All .....;
```

Het resultaat hiervan zijn de volgende twee segmenten die alleen maar verschillen in hun geometry.

| FRC | NAME | NAMELC | ROUTENUM | RTETYP | RTEDIR | RTEDIRVD |
|---|---|---|---|---|---|---|
| 0 | – | – | E313 | 1 | – | – |
| 0 | – | – | E313 | 1 | – | – |

*Query 10: Bepaal alle gemeentes die maximaal 500 meter van de autostrade 'E313' liggen (**spat**).* Net zoals bij query 9 moet hier in alle layers van het wegennetwerk gezocht worden naar de autostrade E313. Hierna moet rond ieder segment van de E313 een buffer van 500 meter aangemaakt worden met de spatial functie *ST_Buffer*. Dan kan gecontroleerd worden welke gemeentes hiermee in de doorsnede liggen.

Listing 6.10: DB2 Spatial Extender - Query 10

```
Select Distinct a8.name From str_00 s, adm_a8 a8
Where db2gse.ST_Intersects(a8.geometry,
                           db2gse.ST_Buffer(s.geometry, 500, 'meter')) = 1
```

```
        And s.routenum = 'E313'
  Union
Select Distinct a8.name From str_01 s, ...
  Union .....;
```

Het resultaat van deze query is de kolom 'NAME' met daarin de waardes: Bassenge, Beringen, Bilzen, Diepenbeek, Geel, Grobbendonk, Ham, Hasselt, Herentals, Herenthout, Herstal, Heusden-Zolder, Hoeselt, Juprelle, Laakdal, Liège, Lummen, Meerhout, Nijlen, Olen, Ranst, Riemst, Tessenderlo, Tongeren, Westerlo en Zandhoven.

*Query 11: Geef informatie over de gemeente die aangeklikt wordt op het scherm (**inter**).* Deze query wordt niet ondersteund door DB2 Spatial Extender wegens het ontbreken van een grafische interface.

*Query 12: Geef informatie over de gemeente die het volgende punt bevat: (5.39229169963549, 50.9270236262395)(**spat**).* Met deze query wordt gesimuleerd dat op het vermelde punt geklikt wordt. Hiervoor moet enkel gecontroleerd worden in welke gemeente het punt ligt. In deze query wordt gekozen om de coördinaten in het *DOUBLE* formaat in de constructor functie *ST_Point* in te geven.

Listing 6.11: DB2 Spatial Extender - Query 12

```
Select name From adm_a8
Where db2gse.ST_Intersects(adm_a8.geometry,
                db2gse.ST_Point(+5.39229169963549E+000,
                                +5.09270236262395E+001, 1003) ) = 1
```

Zoals gewenst is het resultaat van deze query de kolom 'NAME' met als enige waarde 'Diepenbeek'.

Als eindopmerkingen bij de case study met *DB2 Spatial Extender* wordt nog vermeld dat: Iedere referentie query onmiddellijk of na slechts enkele seconden uitgevoerd was; Dat het toevoegen van een spatial index bijgevolg een geringe invloed had op de performantie; Dat alle gebruikte spatial functies intuïtief waren in gebruik en dat er geen 'moeilijke' problemen opgedoken zijn tijdens de uitvoering van de case study.

### 6.3.2 Oracle Spatial

De case study met Oracle Spatial werd onder Windows 2000 Server op een lokale computer uitgevoerd, meer bepaald op een Intel Pentium 3, 500MHz met 786MB werkgeheugen. Hierop is Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 geïnstalleerd samen met zijn spatial optie. (Deze versie

is beschikbaar in de downloadsectie van de site van Oracle [33].) Via de Enterprise Manager (de web-based GUI van Oracle) is een nieuwe user aangemaakt met voldoende rechten (*roles* in Oracle) en systeem privileges

Oracle kan shapefiles niet rechtstreeks inladen, maar deze kunnen wel geconverteerd worden naar een formaat dat wel ingeladen kan worden. Tele Atlas heeft de gegevens al op voorhand geconverteerd naar het *.dat* formaat dat door middel van *bulk loading* ingeladen kan worden. Hiervoor moeten eerst de gewenste tabellen aangemaakt en geregistreerd worden door de SQL-statements uit de gepaste meegeleverde *.sql* bestanden uit te voeren. Dit kan in de command line processor *SQL Plus* door de statements uit de bestanden hiernaar te kopiëren of door het bestand in zijn geheel uit te voeren met 'start *bestandsnaam*'. Nadat de gecomprimeerde *.dat* bestanden uitgepakt zijn kunnen deze geladen worden met de *SQL*Loader*. Zo kan het bestand *BEL_00.dat* geladen worden met behulp van zijn controle bestand *BEL_00_BEL.ctl* door volgend commando in de *Opdrachtprompt* of via een *.bat* bestand uit te voeren:

```
sqlldr user/password BEL_00_BEL
```

Tele Atlas voorziet ook verschillende *.sql* en *.bat* bestanden om alle meegeleverde data in één keer te laden. Deze bestanden kunnen aangepast worden zodat alleen de voor de case study benodigde brongegevens geladen worden. Wanneer alle deze gegevens ingeladen zijn, kunnen de referentie queries uitgevoerd worden.

*Query 1: Bepaal het aantal gemeentes in België (**alf**).* Hiervoor moet een *count* gedaan worden van het aantal verschillende tuppels in A8.

Listing 6.12: Oracle Spatial - Query 1

```
Select Count(Distinct(id)) As Aantal_Gemeentes From adm_a8;
```

Het resultaat van deze query is de kolom 'AANTAL_GEMEENTES' met als enige waarde '590'.

*Query 2: Bepaal het aantal inwoners in Diepenbeek (**alf**).* Deze waarde bevindt zich in het veld *pop* van de tabel A8 waar de *name* van het tuppel 'Diepenbeek' is.

Listing 6.13: Oracle Spatial - Query 2

```
Select pop From bel_a8 Where name = 'Diepenbeek';
```

Als resultaat heeft deze query de kolom 'POP' met daarin de waarde '16899'.

*Query 3: Bepaal het aantal inwoners in Limburg (**alf**).* Hiervoor moet de som genomen worden van alle *pop* velden van de tuppels van Limburg uit tabel A8. Om te controleren of een tuppel tot Limburg hoort moet de *order02* waarde van Limburg uit tabel A2 overeenkomen met die uit A8.

Listing 6.14: Oracle Spatial - Query 3

```
Select Sum(a8.pop) As Pop_Limburg From bel_a2 a2, bel_a8 a8
Where a2.name = 'Limburg' And a2.order02 = a8.order02;
```

De totale populatie in Limburg is '785228'.

*Query 4: Geef een lijst van alle gemeentes in Limburg (**alf**).* Deze query moet van elk tuppel in Limburg de waarde in het veld *name* teruggeven.

Listing 6.15: Oracle Spatial - Query 4

```
Select a8.name From bel_a2 a2, bel_a8 a8
Where a2.name = 'Limburg' And a2.order02 = a8.order02;
```

Het resultaat hiervan is de kolom 'NAME' met als rijen (gegroepeerd per 11 rijen waarbij de kolomheader per groep herhaald wordt): Fourons, Houthalen-Helchteren, Herk-de-Stad, As, Bocholt, Lummen, Tongeren, Peer, Neerpelt, Diepenbeek, Zutendaal, Bilzen, Herstappe, Leopoldsburg, Nieuwerkerken, Maaseik, Overpelt, Hamont-Achel, Riemst, Hoeselt, Borgloon, Ham, Bree, Gingelom, Zonhoven, Lommel, Hechtel-Eksel, Lanaken, Sint-Truiden, Beringen, Heusden-Zolder, Halen, Hasselt, Opglabbeek, Kortessem, Maasmechelen, Wellen, Alken, Dilsen-Stokkem, Heers, Meeuwen-Gruitrode, Genk, Kinrooi en Tessenderlo.

*Query 5: Bepaal de gemeentes grenzend aan Diepenbeek (**spat**).* Dit is de eerste spatial query in Oracle, hiervoor kan ofwel de spatial operator *SDO_Relate* gebruikt worden (mits er een index op de spatial kolom gedefinieerd is) ofwel de spatial functie *Relate*.

Listing 6.16: Oracle Spatial - Query 5

```
Select A8_b.name From bel_a8 A8_a, bel_a8 A8_b
Where SDO_Relate(A8_b.geom, A8_a.geom, 'mask=TOUCH') = 'TRUE'
      And A8_a.name = 'Diepenbeek';

Select A8_b.name From bel_a8 A8_a, bel_a8 A8_b
Where SDO_Geom.Relate(A8_a.geom, 'TOUCH', A8_b.geom, 0.005) = 'TOUCH'
      And A8_a.name = 'Diepenbeek';
```

In beide gevallen is het resultaat een tabel met de kolom 'NAME' met als waardes: Bilzen, Hoeselt, Hasselt, Kortessem en Genk.

*Query 6: Output (als tekst/binair) de gemeente Diepenbeek (**spat**).* Bij deze query moet de polygon van Diepenbeek in alle ondersteunde formaten teruggegeven worden. Dit kan in het Well-known text, het Well-known binary of het Geography Markup Language formaat.

Listing 6.17: Oracle Spatial - Query 6

```
Select To_Char(a8.geom.Get_WKT()) As WKT From bel_a8 a8
Where a8.name = 'Diepenbeek';

Select a8.geom.Get_WKB() As WKB From bel_a8 a8
Where a8.name = 'Diepenbeek';

Select To_Char(SDO_Util.To_GMLGeometry(geom)) As GmlGeometry
From bel_a8 a8 Where a8.name = 'Diepenbeek';
```

Elke query geeft de voorstelling van de polygon van Diepenbeek terug in het gewenste formaat. Het converteren naar een shapefile wordt momenteel niet door Oracle ondersteund, maar kan indien gewenst wel door een externe toepassing gedaan worden.

*Query 7: Bepaal de gemeentes in Antwerpen die groter zijn dan de grootste gemeente in Limburg door middel van de oppervlakte gegeven in de attributen (**alf**).* Hiervoor moet eerst de oppervlakte van de grootste Limburgse gemeente bepaald worden zodat de oppervlakte van iedere Antwerpse gemeente hiermee vergeleken kan worden.

Listing 6.18: Oracle Spatial - Query 7

```
Select A8_a.name, A8_a.featarea
From bel_a8 A8_a, bel_a8 A8_b, bel_a2 A2_a
Where A8_b.featarea In
        (Select Max(A8_c.featarea) From bel_a8 A8_c, bel_a2 A2_b
         Where A2_b.name = 'Limburg' And A2_b.order02 = A8_c.order02)
      And A8_a.featarea > A8_b.featarea
      And A2_a.name = 'Antwerpen' And A2_a.order02 = A8_a.order02;
```

Als resultaat heeft deze query een tabel met de twee kolommen 'NAME' en 'FEATAREA' met daarin de tuppels: (Mol, 114617666), (Geel, 110093679) en (Antwerpen, 203548389).

*Query 8: Bepaal de gemeentes in Antwerpen die groter zijn dan de grootste gemeente in Limburg door de oppervlakte te berekenen van de geometry (**spat**).* Deze query berekent hetzelfde als de vorige query, maar nu door de oppervlakte van de gemeentes te berekenen met de spatial functie *SDO_Area*.

Listing 6.19: Oracle Spatial - Query 8

```
Select A8_a.name, A8_a.featarea, SDO_Geom.SDO_Area(A8_a.geom, 0.005) As Area
From bel_a8 A8_a, bel_a8 A8_b, bel_a2 A2_a
Where SDO_Geom.SDO_Area(A8_b.geom, 0.005) In
          (Select Max(SDO_Geom.SDO_Area(A8_c.geom, 0.005))
           From bel_a8 A8_c, bel_a2 A2_b
           Where A2_b.name = 'Limburg' And A2_b.order02 = A8_c.order02)
       And SDO_Geom.SDO_Area(A8_a.geom, 0.005) >
                              SDO_Geom.SDO_Area(A8_b.geom, 0.005)
       And A2_a.name = 'Antwerpen' And A2_a.order02 = A8_a.order02;
```

Het resultaat van deze query is hetzelfde als bij de vorige query. Ter referentie wordt per gemeente zijn attribuut oppervlakte en de berekende oppervlakte getoond, beide in $m^2$ uitgedrukt.

| NAME | FEATAREA | AREA |
|---|---|---|
| Mol | 114617666 | 114617743 |
| Geel | 110093679 | 110088338 |
| Antwerpen | 203548389 | 203549582 |

*Query 9: Bepaal alle autostrades (RouteNum LIKE 'E%') die door Diepenbeek lopen (**spat**).* Omdat autostradesegmenten op ieder niveau van het wegennetwerk kunnen voorkomen, moet voor deze query in al de negen layers gezocht worden en de resultaten van iedere layer samengevoegd. Hiervoor kan zowel de spatial operator *SDO_Relate* als de spatial functie *Relate* gebruikt worden. In onderstaande listing wordt de functie *Relate* gebruikt.

Listing 6.20: Oracle Spatial - Query 9

```
Select s.FRC, s.NAME, s.NAMELC, s.ROUTENUM, s.RTETYP,
       s.RTEDIR, s.RTEDIRVD
From bel_00 s, bel_a8 A8
Where SDO_Geom.Relate(s.geom, 'ANYINTERACT', A8.geom, 0.005) = 'TRUE'
       And A8.name = 'Diepenbeek' And s.routenum Like 'E%'
  Union All
Select ... From bel_01 s, ...
  Union All .....;
```

Als resultaat heeft deze query de volgende twee segmenten, met enige verschil hun geometry.

| FRC | NAME | NAM | ROUTENUM | RTETYP | RTEDIR | RTEDIRVD |
|---|---|---|---|---|---|---|
| 0 | | | E313 | 1 | | |
| 0 | | | E313 | 1 | | |

*Query 10: Bepaal alle gemeentes die maximaal 500 meter van de autostrade 'E313' liggen (**spat**).* Net zoals bij de vorige query moet hierbij in alle layers van het wegennetwerk gezocht worden naar de autostrade E313. Dan moet rond ieder segment van de E313 een buffer van 500 meter aangemaakt worden

met de spatial functie *SDO_Buffer*. Ook hier kan de spatial functie *Relate* gebruikt worden om te bepalen welke gemeentes overlappen met de gebufferde E313 segmenten, maar op de testcomputer (Pentium 3, 500MHz) duurde dit meer dan 4 uur. Een betere oplossing is de spatial operator *SDO_Relate* te gebruiken *met de parameters in de juiste volgorde*, dus de tabel met de index als eerste parameter. In dit geval duurt de query slechts 2 minuten.

Listing 6.21: Oracle Spatial - Query 10

```
Select Distinct A8.name From bel_00 s, bel_a8 A8
Where s.routenum = 'E313' And
        SDO_Geom.Relate(SDO_Geom.SDO_Buffer(s.geom, 500, 0.01, 'unit=Meter'),
                        'ANYINTERACT', A8.geom, 0.005) = 'TRUE'
  Union .....;

Select Distinct a8.name From bel_00 s, bel_a8 a8
Where s.routenum = 'E313' And
        SDO_Relate(a8.geom, SDO_Geom.SDO_Buffer(s.geom, 500, 0.01,
                    'unit=Meter'), 'MASK=ANYINTERACT') = 'TRUE'
  Union
Select Distinct a8.name From bel_01 s, ...
  Union .....;
```

Het resultaat hiervan is de kolom 'NAME' met als waardes: Bassenge, Beringen, Bilzen, Diepenbeek, Geel, Grobbendonk, Ham, Hasselt, Herentals, Herenthout, Herstal, Heusden-Zolder, Hoeselt, Juprelle, Laakdal, Liège, Lummen, Meerhout, Nijlen, Olen, Ranst, Riemst, Tessenderlo, Tongeren, Westerlo en Zandhoven.

*Query 11: Geef informatie over de gemeente die aangeklikt wordt op het scherm (**inter**).* Deze query wordt niet ondersteund door Oracle Spatial wegens het ontbreken van een grafische interface.

*Query 12: Geef informatie over de gemeente die het volgende punt bevat: (5.39229169963549, 50.9270236262395)(**spat**).* Met deze query wordt gesimuleerd dat op het vermelde punt geklikt wordt. Hiervoor moet er gecontroleerd worden met welke gemeente dit punt in de doorsnede ligt. In deze query wordt gekozen om de coördinaten eerst als velden van het *SDO_Geometry* datatype in te vullen en vervolgens in het *Well-known text* formaat.

Listing 6.22: Oracle Spatial - Query 12

```
Select name From bel_a8 a8
Where SDO_Geom.Relate(a8.geom, 'ANYINTERACT',
            SDO_Geometry(2001, 8307, SDO_Point_Type(+5.39229169963549E+000,
                        +5.09270236262395E+001, null), null, null), 0.005) = 'TRUE';
```

```
Select name From bel_a8 a8
Where SDO_Geom.Relate(a8.geom, 'ANYINTERACT',
            SDO_Geometry(' POINT(5.39229169963549 50.9270236262395) ', 8307),
                         0.005) = 'TRUE';
```

Zoals verwacht is het resultaat van deze query de kolom 'NAME' met als enige waarde 'Diepenbeek'.

Als slotopmerkingen bij de case study met *Oracle Spatial* wordt nog vermeld dat: Alle queries, behalve query 10, onmiddellijk of na enkele tientallen seconden uitgevoerd waren; Dat bij de spatial queries die minder snel klaar waren de toevoeging van een index zorgde dat de uitvoeringstijd merkbaar verkleinde naar enkele seconden; Dat het door elkaar gebruiken van spatial *operaties* en spatial *functies* niet altijd even intuïtief werkt en dat dit ook voor de grootste problemen heeft gezorgd tijdens het testen.

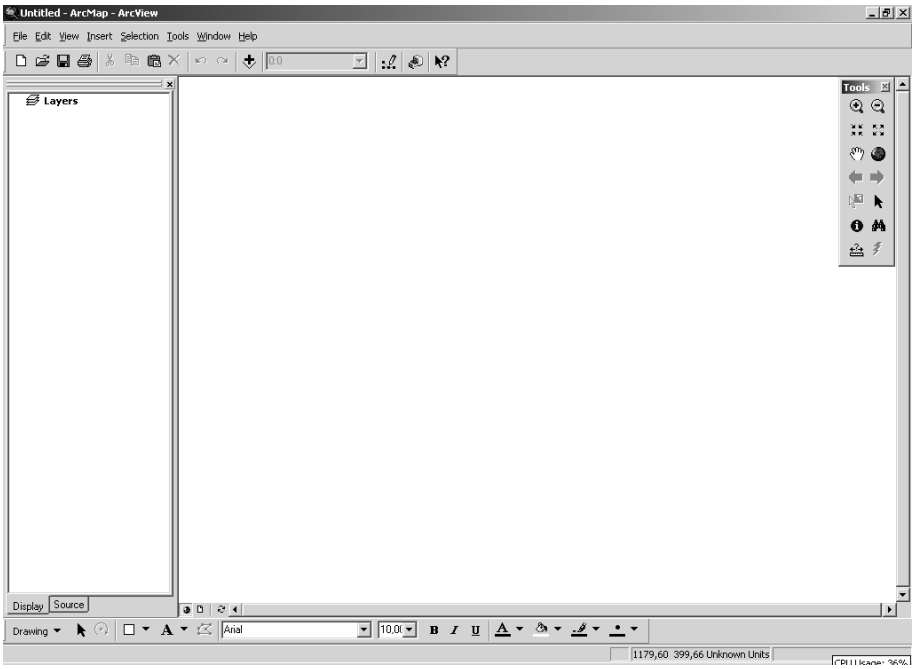
### 6.3.3 ArcGIS

Het uitvoeren van de case study met ArcGIS werd onder Windows 2000 Server op een lokale computer gedaan, de specificaties van dit systeem zijn: een Intel Pentium 3, 500MHz met 786MB werkgeheugen. Hierop is ArcGIS versie 8.1.0.642 geïnstalleerd, een speciale gebruiker aanmaken is hier niet nodig.

Er moeten ook geen gegevens op voorhand ingeladen worden omdat ArcGIS rechtstreeks met shapefiles kan werken. Om de referentie queries uit te voeren volstaat het enkel gebruik te maken van ArcMap. Wanneer ArcMap gestart wordt zijn er (uiteraard) nog geen gegevens geopend, deze begintoestand wordt geïllustreerd in figuur 6.6.

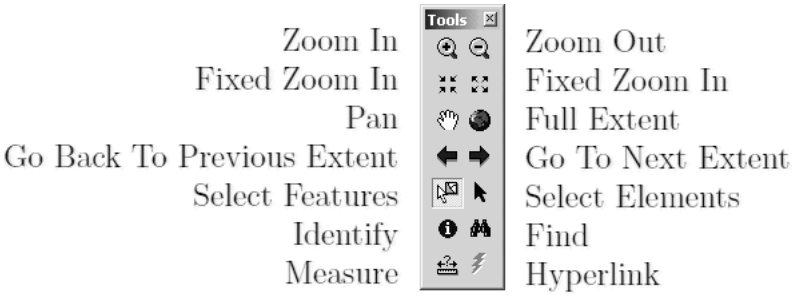
In ArcMap kan een shapefile geopend worden door op de  knop in de *standaard* werkbalk te klikken. In het dialoogvenster dat vervolgens verschijnt kunnen de gewenste shapefile bestanden, of bestanden in andere door ArcGIS ondersteunde formaten, geselecteerd worden. Nadat de selectie bevestigd wordt, worden de gegevens als layers toegevoegd aan het actieve dataframe. In de 'Table of Contents' die zich aan de linkerkant bevindt, is een overzicht te zien van alle dataframes en de daarin geopende layers.

De 'werktuigen' in de werkbalk *Tools*, rechtsboven in figuur 6.6, kunnen gebruikt worden om te navigeren over de data, om objecten te selecteren, te zoeken en om de attributen ervan op te vragen. In figuur 6.7 is een overzicht te zien van deze tools.




Figuur 6.6: Begintoestand ArcMap.

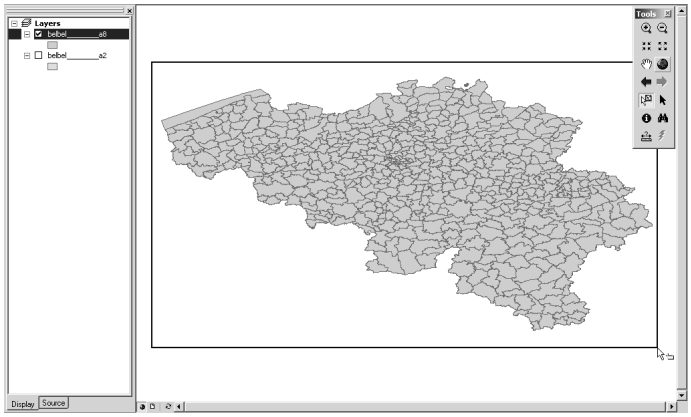
Om sneller te werken met de gegevens is het raadzaam om een spatial index te laten creëren door ArcCatalog. Dit kan gedaan worden door bij de eigenschappen van een layer naar de 'index' tab te navigeren, op 'Add' te klikken en te bevestigen. De spatial index bestanden (.sbn en .sbx) worden ook aangemaakt als voor een layer in de 'Editor' werkbalk op 'Start Editing' geklikt wordt.



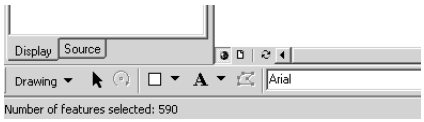

Figuur 6.7: ArcMap tools.

Met deze basis over ArcMap kan begonnen worden met het uitvoeren van de referentie queries.

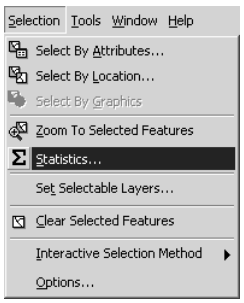
*Query 1: Bepaal het aantal gemeentes in België (**alf**).* De oplossing hiervan bekomt men door alle gemeentes in België te selecteren en dan op te vragen hoeveel objecten er geselecteerd zijn. Alle gemeentes selecteren kan door met de 'Select Features' tool een kader te trekken rond alle gemeentes zoals getoond wordt in figuur 6.8(a). Het aantal geselecteerde features verschijnt dan voor een paar seconden links onder in beeld, zie figuur 6.8(b). Een andere manier om dit aantal te weten te komen is door de statistieken van de huidige selectie op te vragen zoals getoond wordt in figuur 6.8(c). Hierdoor verschijnt een venster zoals in figuur 6.8(d) waarvan per layer de statistieken van de huidige selectie afgelezen kunnen worden.



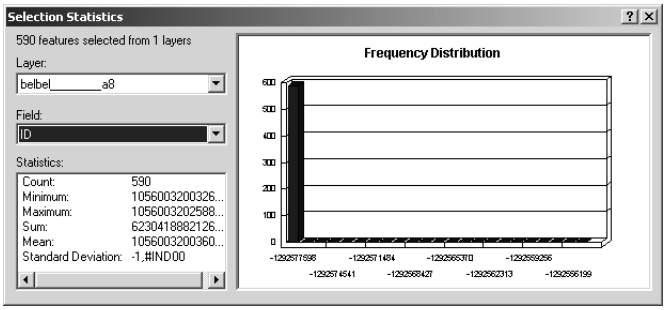

(a) Selecteren




(b) Resultaat - $1^{ste}$ manier



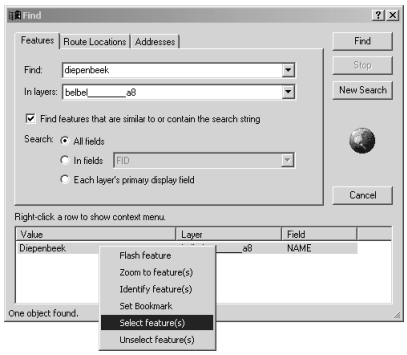

(c) Statistieken



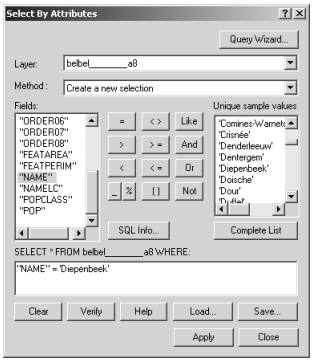

(d) Resultaat - $2^{de}$ manier

Figuur 6.8: ArcGIS - Query 1.

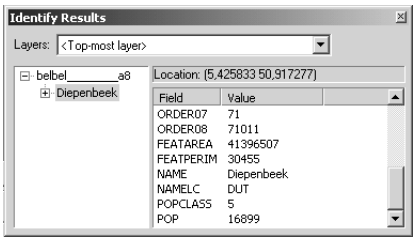
*Query 2: Bepaal het aantal inwoners in Diepenbeek (**alf**).* Hiervoor moet eerst de gemeente Diepenbeek gevonden en/of geselecteerd worden, dan pas kunnen zijn attributen opgevraagd worden. Deze zoekactie kan op meerdere manieren gebeuren, één manier is met de 'Find' tool (zie figuur 6.9(a)), een tweede is met de 'Select by Attributes' dialoog (zie figuur 6.9(b)). De attributen van een feature kunnen opgevraagd worden door er met de 'Identify' tool op te klikken (zie figuur 6.9(c)) of door er de statistieken van op te vragen (zie figuur 6.9(d)).



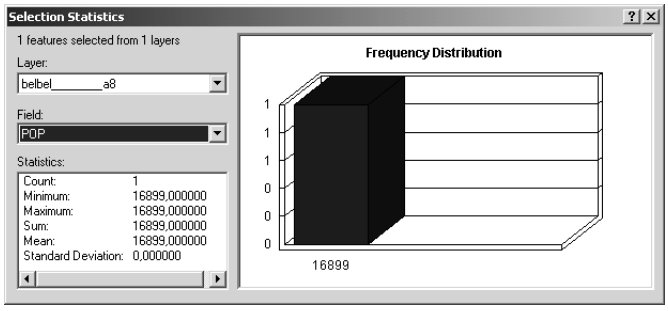

(a) Selecteren - $1^{ste}$ manier




(b) Selecteren - $2^{de}$ manier



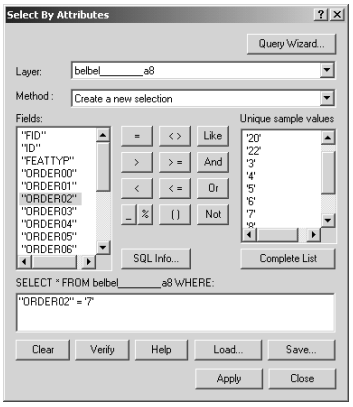

(c) Resultaat - $1^{ste}$ manier



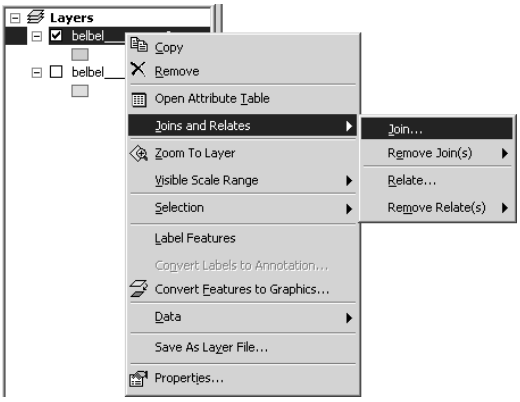

(d) Resultaat - $2^{de}$ manier

Figuur 6.9: ArcGIS - Query 2.

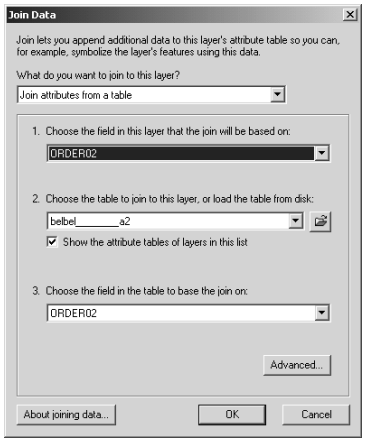
*Query 3: Bepaal het aantal inwoners in Limburg (**alf**).* Voor deze query moet de som genomen worden van het attribuut 'pop' van alle gemeentes in Limburg. Er zijn twee manieren om te bepalen of een gemeente in Limburg ligt: De eerste is door het attribuut 'order02' van de provincie Limburg op te zoeken in shapefile A2 (orde02 = '7') en vervolgens alle gemeentes in A8 te selecteren die ook deze waarde hebben (zie figuur 6.10(a)). De tweede manier is door A8 en A2 met elkaar te joinen op het attribuut 'order02' (zie figuren 6.10(b) en 6.10(c)). Vervolgens moeten hieruit die gemeentes geselecteerd worden waarvan de 'a2.name' gelijk is aan 'Limburg' (zie figuur 6.10(d)). Nadat alle gemeentes van Limburg geselecteerd zijn kan het resultaat gevonden worden in de statistieken van 'pop' bij de som, zie figuur 6.11.



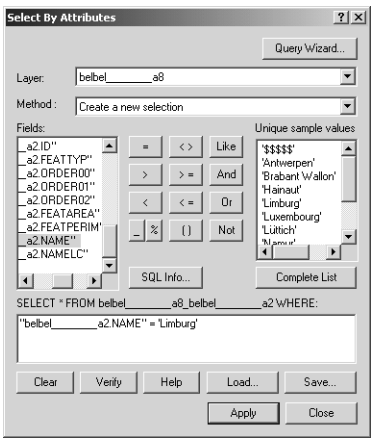

(a) Methode 1 - Selecteren




(b) Methode 2 - Joinen

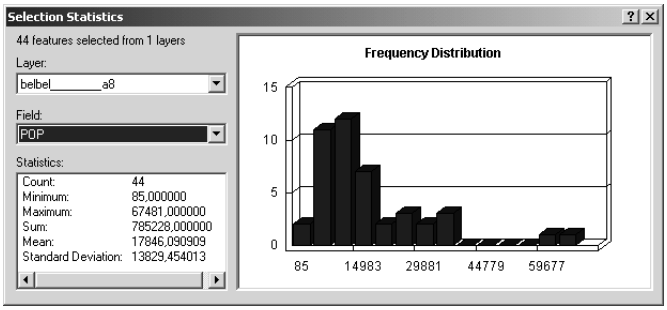



(c) Methode 2 - Join dialoog



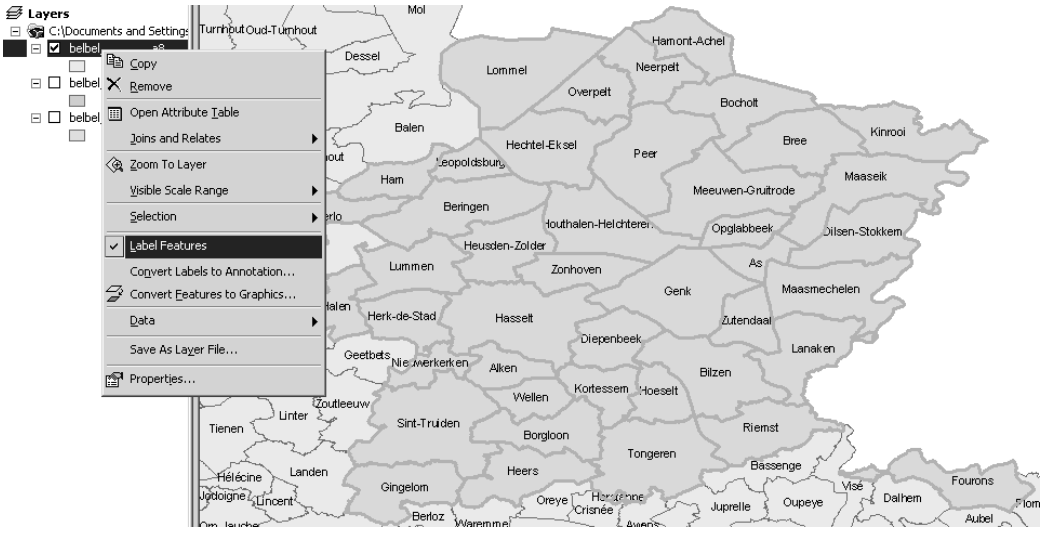

(d) Methode 2 - Selecteren

Figuur 6.10: ArcGIS - Query 3.

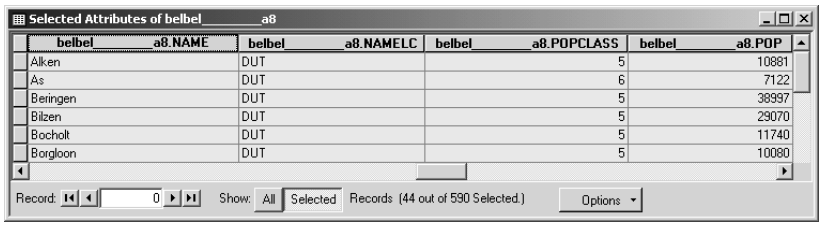



Figuur 6.11: ArcGIS - Query 3 resultaat.

*Query 4: Geef een lijst van alle gemeentes in Limburg (**alf**).* Een eerste mogelijkheid is om alle gemeentes van een label te voorzien, zie figuur 6.12. Bij de eigenschappen van een layer kan ingesteld worden welke attribuutwaarde, in dit geval 'name', als label getoond wordt. Voor de andere mogelijkheid moeten eerst alle gemeentes van Limburg in layer A8 geselecteerd worden. Klik vervolgens met de rechtermuisknop op deze layer en klik op 'Open Attribute Table'. Geef vervolgens aan dat alleen de geselecteerde features getoond moeten worden. Figuur 6.13 toont de attribuut tabel van de gejoinde shapefile uit de vorige query.

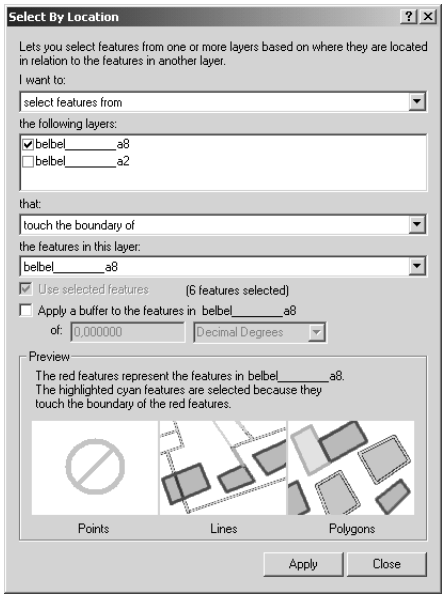



Figuur 6.12: ArcGIS - Query 4 resultaat ($1^{ste}$ manier).

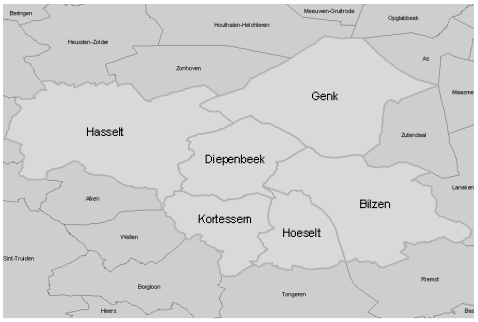



Figuur 6.13: ArcGIS - Query 4 resultaat ($2^{de}$ manier).

*Query 5: Bepaal de gemeentes grenzend aan Diepenbeek (**spat**).* Hiervoor moet eerst de gemeente Diepenbeek geselecteerd zijn. De aangrenzende gemeentes kunnen bepaald worden met de 'Select by Location' dialoog door hierin de relatie 'touch' te specificeren. Dit wordt geïllustreerd in figuur 6.14(a). Merk op dat Diepenbeek ook in het resultaat zit, zoals te zien is in figuur 6.14(b).




(a) 'Select by Location'

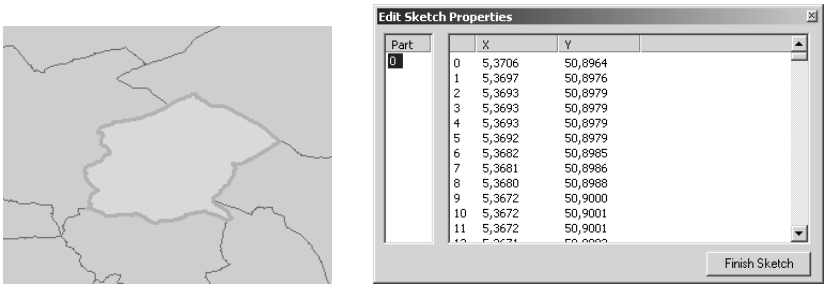



(b) Resultaat

Figuur 6.14: ArcGIS - Query 5.

*Query 6: Output (visualiseer) de gemeente Diepenbeek (**spat**).* Een polygon kan op twee manieren 'getoond' worden. De eerste manier is door gewoon te navigeren naar de gewenste gemeente (zie figuur 6.15(a)). De tweede manier

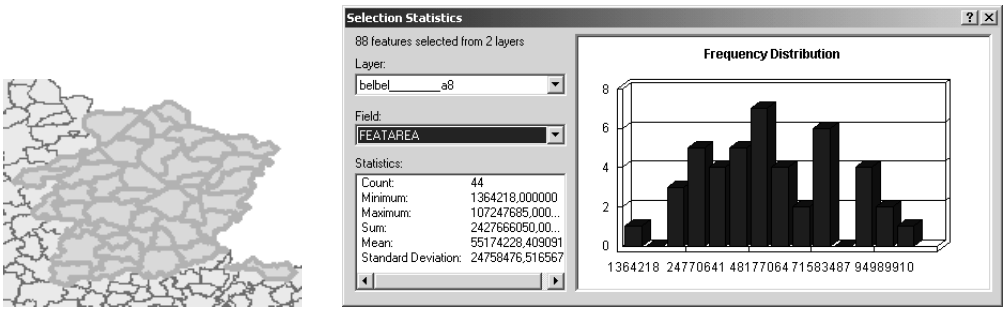
is door zijn coördinaten op te vragen tijdens het editeren (zie figuur 6.15(b)).




(a) Resultaat - $1^{ste}$ manier (b) Resultaat - $2^{de}$ manier

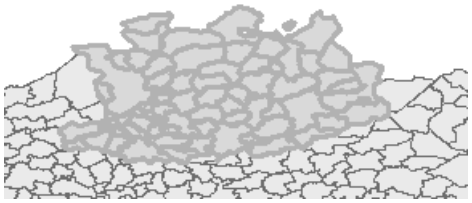
Figuur 6.15: ArcGIS - Query 6.

*Query 7: Bepaal de gemeentes in Antwerpen die groter zijn dan de grootste gemeente in Limburg door middel van de oppervlakte gegeven in de attributen (**alf**).* Hiervoor moet eerst de oppervlakte bepaald worden van de grootste gemeente in Limburg, dit is 107247685 $m^2$ (zie figuren 6.16(a) en 6.16(b)). Vervolgens wordt de oppervlakte van iedere Antwerpse gemeente vergeleken met deze waarde (zie figuren 6.17(a) en 6.17(b)) en worden alleen die gemeentes geselecteerd die groter zijn (zie figuren 6.17(c) en 6.17(d)). Merk op dat het niet mogelijk is deze query in één keer uit te voeren. Dit komt omdat de operatie 'Max()' niet gebruikt kan worden in de 'Where' module. Een andere reden is dat het ook niet mogelijk is in ArcGIS om een tabel/shapefile met zichzelf te joinen.

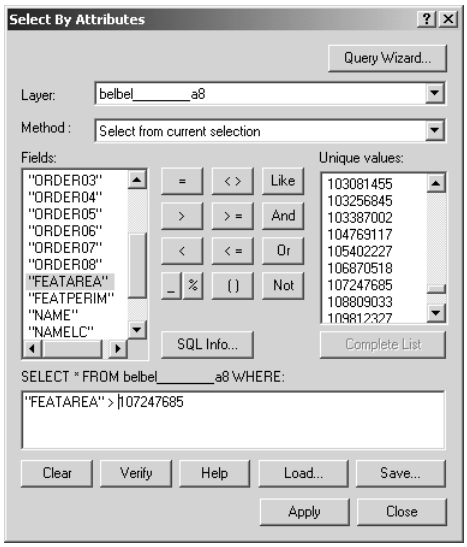



(a) Selectie Limburg (b) Maximale oppervlakte

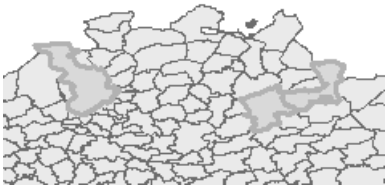
Figuur 6.16: ArcGIS - Query 7.



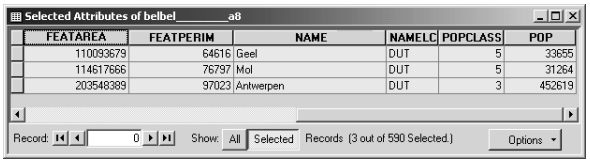
(a) Selectie Antwerpen




(b) Selecteren



(c) Resultaat - $1^{ste}$ manier



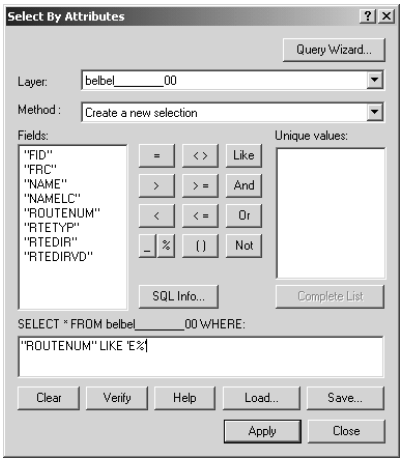

(d) Resultaat - $2^{de}$ manier

Figuur 6.17: ArcGIS - Query 7 (vervolg).

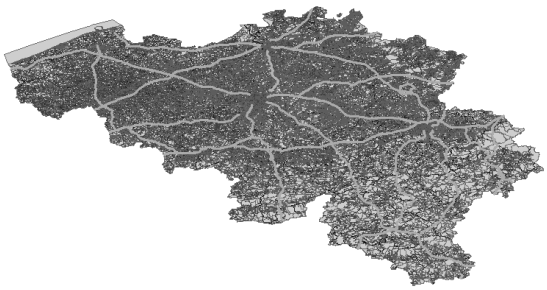
*Query 8: Bepaal de gemeentes in Antwerpen die groter zijn dan de grootste gemeente in Limburg door de oppervlakte te berekenen van de geometry (**spat**).* Deze query is niet efficiënt uit te voeren met ArcGIS. Het is wel mogelijk om de oppervlakte van een polygon te bepalen met de ArcGIS extension *3D Analyst* (zie figuur 6.18). (Hiervoor is wel een aparte licentie vereist, deze was echter niet beschikbaar tijdens het opstellen van deze verhandeling.) Een eerste moeilijkheid is het bepalen van de gemeente van Limburg met de maximale oppervlakte. Tenzij dit een optie is van *3D Analyst*, zal de oppervlakte van iedere gemeente manueel met elkaar vergeleken moeten worden. Vervolgens moet gecontroleerd worden welke gemeentes van Antwerpen groter zijn dan de grootste van Limburg. Dit zal wederom manueel moeten gebeuren (tenzij dit met *3D Analyst* gedaan kan worden).


Figuur 6.18: ArcGIS - Query 8 - 3D Analyst.

*Query 9: Bepaal alle autostrades (RouteNum LIKE 'E%') die door Diepenbeek lopen (**spat**).* Hiervoor moeten eerst alle autostrades geselecteerd worden. Dit kan gedaan worden door in iedere layer van het wegennetwerk die features te selecteren waarvan het attribuut *RouteNum* LIKE 'E%' is. De selectiemethode mag zowel 'nieuwe selectie' of 'toevoegen aan bestaande selectie' (mits er niets geselecteerd is in iedere layer) zijn omdat selecties in verschillende layers apart gebeuren. Figuur 6.19(a) illustreert het selecteren van autostrades in layer 00.



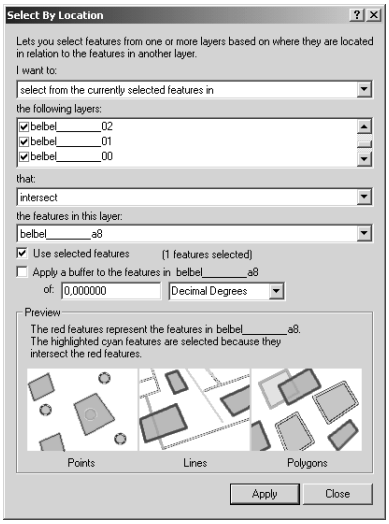

(a) Selecteren autostrades



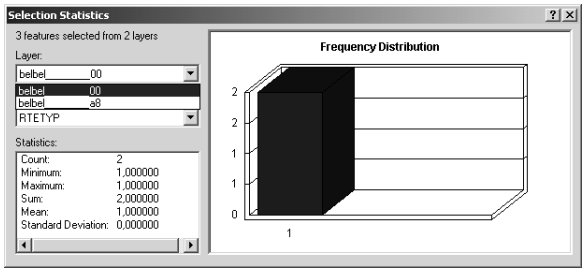
(b) Totale selectie

Figuur 6.19: ArcGIS - Query 9.

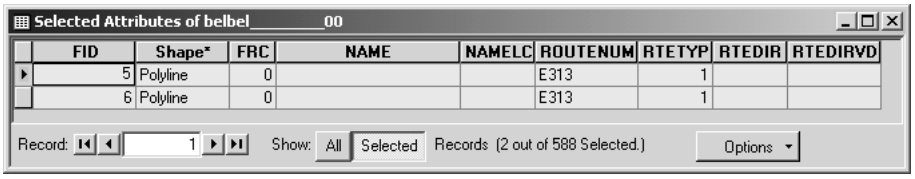
Vervolgens moet de gemeente Diepenbeek geselecteerd worden (toegevoegd aan de autostrades). Figuur 6.19(b) toont een overzicht van de tot nu toe geselecteerde features. Om tot de oplossing te komen moet gecontroleerd worden welke wegsegmenten in de doorsnede liggen met de gemeente Diepenbeek, zie figuur 6.20(a). Via de statistieken in figuur 6.20(b) kan gezien worden dat de autostrades alleen uit layer 00 komen. De resultaten in de attribuut tabel van deze layer zijn te zien in figuur 6.20(c).




(a) Autostrades door Diepenbeek




(b) Statistieken resultaat

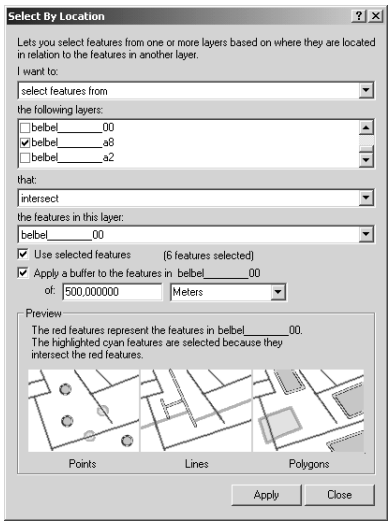



(c) Resultaat

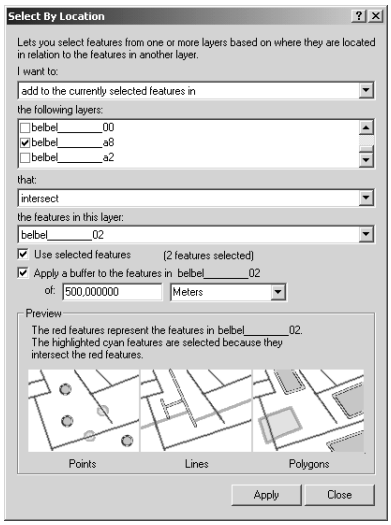
Figuur 6.20: ArcGIS - Query 9 (vervolg).

*Query 10: Bepaal alle gemeentes die maximaal 500 meter van de autostrade 'E313' liggen (**spat**).* Voor deze query moeten eerst alle segmenten van de E313 geselecteerd worden in alle layers van het wegennetwerk. Daarna moet per layer met een buffer van 500 $m$ gecontroleerd worden welke gemeentes in layer A8 hiermee overlappen. Figuur 6.21(a) toont deze procedure voor layer 00. Bij alle volgende layers moet aan de bestaande selectie in A8 toegevoegd worden, figuur 6.21(b) illustreert dit voor layer 02. Het resultaat is in figuur 6.21(c) te zien, de lichtgrijze gemeentes liggen langs de E313, de E313 is de

grijze lijn door deze gemeentes en de zwarte lijnen zijn de overige autostrades. In figuur 6.21(d) is de attribuut tabel van het resultaat te zien.

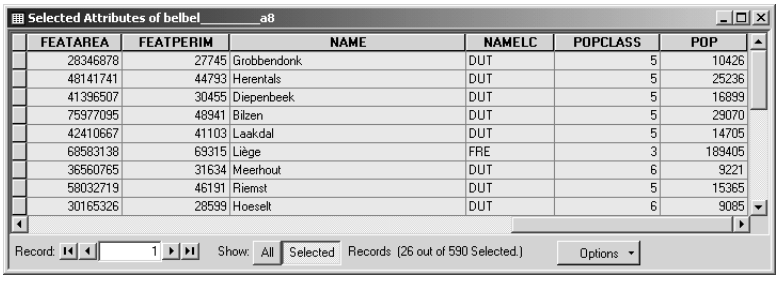



(a) Selecteren in layer 00




(b) Selecteren in layer 02

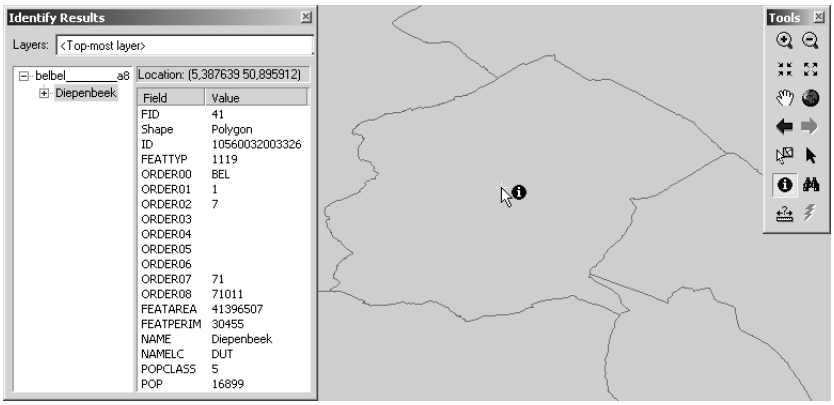
(c) Resultaat - $1^{ste}$ manier




(d) Resultaat - $2^{de}$ manier

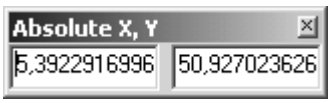
Figuur 6.21: ArcGIS - Query 10.

*Query 11: Geef informatie over de gemeente die aangeklikt wordt op het scherm (**inter**).* Deze interactieve query is al meerdere keren uitgevoerd als suboperatie in de vorige queries. Hiervoor moet enkel met de 'Identify' tool op de gewenste polygon geklikt worden zoals getoond wordt in figuur 6.22.




Figuur 6.22: ArcGIS - Query 11 - Interactie.

*Query 12: Geef informatie over de gemeente die het volgende punt bevat: (5.39229169963549, 50.9270236262395)(**spat**).* Voor deze query wordt aangenomen dat dit punt nog niet bestaat als een feature, daarom moeten we dit eerst aanmaken. Hiervoor moet een nieuwe shapefile aangemaakt worden. Omdat één shapefile maar één type ondersteund, kan dit punt niet in een reeds gebruikte shapefile toegevoegd worden. Het aanmaken van een shapefile kan met ArcCatalog. Hierbij moet als spatial referentiesysteem hetzelfde systeem als bij de overige brongegevens opgegeven worden, namelijk 'WGS 1984'. Deze lege shapefile moet dan toegevoegd worden aan het dataframe. Met behulp van de *Editor* werkbalk kan hierin een nieuw punt gecreëerd worden. Klik met de rechtermuisknop op een lege plaats en kies voor '*Absolute X, Y...*' om een dialoogvenster zoals in figuur 6.23 te bekomen waarin de waardes '5,39229169963549' en '50.9270236262395' ingevuld kunnen worden. Hierna moet enkel nog gecontroleerd worden welke gemeente in de doorsnede met dit punt ligt en welke zijn attributen zijn.




Figuur 6.23: ArcGIS - Query 12.

Als eindbemerkingen bij de case study met *ArcGIS* wordt nog vermeld dat: Iedere deelbewerking van elke query (na het bevestigen van de operatie) al na enkele seconden klaar was met uitvoeren; Dat het op voorhand aanmaken van spatial indexen verantwoordelijk is voor deze snelle uitvoering; Dat zonder deze indexen zelfs het identificeren van een feature, wanneer er meerdere layers geopend zijn, te langzaam gaat om aangenaam te kunnen werken; Dat de GUI van ArcGIS ervoor zorgt dat niet alle SQL expressies, zoals subexpressies, uit te voeren zijn; Dat het gebruik van 'NOT LIKE ...' in de 'Select by Attributes' dialoog hetzelfde resultaat geeft als 'LIKE ...', waardoor een query anders geformuleerd moest worden; Dat de GUI de volgorde van de deelbewerkingen definieert terwijl dit niet altijd de logische volgorde is en dat deze andere redeneerwijze de meeste kopzorgen opleverde.

# Hoofdstuk 7

# Vergelijking

In dit laatste hoofdstuk worden de drie commerciële systemen uit de vorige twee hoofdstukken met elkaar vergeleken. Hierbij zal de nadruk in eerste instantie op de database systemen *DB2 Spatial Extender* en *Oracle Spatial* liggen en in mindere mate op het GIS systeem *ArcGIS*. Hierbij zal blijken dat de twee spatial database systemen niet zo veel van elkaar verschillen in hun kernfunctionaliteit. Vervolgens wordt ArcGIS vergeleken met deze twee spatial database systemen als één geheel.

## 7.1 Oracle versus DB2

De versies van de twee systemen die hier vergeleken worden zijn: versie $10g$ release 1 van *Oracle Spatial* ([33], [23] en hoofdstuk 2 van [29]) en versie 8.2 van *DB2 Spatial Extender* ([32] en [23]). Beide systemen zijn spatial extensies van object-relationele DBMS die geïmplementeerd zijn met behulp van *abstracte datatypes* en *stored procedures*.

Oracle biedt vanaf versie 7.2 (1996) de mogelijkheid om meer-dimensionale/ spatial gegevens op te slaan, weliswaar in verschillende rijen in een relationele tabel. Vanaf versie $8i$ (1997) is er ondersteuning voor object georiënteerde types. Hierdoor kunnen ruimtelijke gegevens in één rij in één object, namelijk *SDO_Geometry*, worden opgeslagen. Vanaf versie 8.1.5 (1999) voldoet Oracle ook aan de "OpenGIS Consortium's Simple Feature Specification for SQL" [17]. In versie $9i$ (2001) en $10g$ (2003) wordt nog extra functionaliteit toegevoegd, zoals de *Advanced Spatial Engine* en extra datamodellen.

IBM ondersteunt vanaf versie 5.2 (1998) gestructureerde objecten. Deze kunnen gebruikt worden in *user-defined functions*, maar het duurt nog tot versie

7 (2001) totdat er object-specifieke *methods* op gedefinieerd kunnen worden. Versie 7.1 (2001) is de eerste versie waarin spatial gegevens kunnen worden opgeslagen in één object dat een subtype van *ST_Geometry* moet zijn. Vanaf deze versie voldoet IBM ook aan de "OpenGIS Consortium's Simple Feature Specification for SQL" [17]. In versie 8 (2003) wordt nog extra functionaliteit toegevoegd, zoals de *Index Advisor* en betere integratie met geocoders.

In onderstaande tabellen worden de spatial datatypes, indexen en functies van beide systemen langs elkaar gezet om ze beter te kunnen vergelijken.

| | **Oracle Spatial** | **DB2 Spatial Extender** |
|---|---|---|
| Punt | SDO_Geometry | ST_Point |
| MultiPunt | SDO_Geometry | ST_MultiPoint |
| Lijn | SDO_Geometry | ST_LineString |
| MultiLijn | SDO_Geometry | ST_MultiLineString |
| Polygon | SDO_Geometry | ST_Polygon |
| MultiPolygon | SDO_Geometry | ST_MultiPolygon |
| Willekeurige verzameling | SDO_Geometry | ST_GeomCollection |

Tabel 7.1: Oracle versus DB2 - Datatypes.

Zoals te zien is in tabel 7.1 worden in Oracle alle soorten geometrieën in één type opgeslagen. Dit heeft zijn voor- en nadelen: zo kan om het even welke soort geometry in dit type opgeslagen worden, maar in het geval dat de constructor van *SDO_Geometry* gebruikt wordt moet naast de coördinaten ook de code van het type en de code van de verbindingen tussen opeenvolgende punten opgegeven worden. In DB2 is er voor iedere soort een type, maar er is ook nog een globaal type mocht het niet op voorhand bekend zijn welke soort geometry erin opgeslagen zal worden.

| **Oracle Spatial** | **DB2 Spatial Extender** |
|---|---|
| well-known text | well-known text |
| well-known binary | well-known binary |
| coördinaten | geography markup language |
| .dmp file* (Oracle) | shapefile (ESRI) |
| | shapefile* (ESRI) |
| | coördinaten |
| | spatial data engine* (ESRI) |

Tabel 7.2: Oracle versus DB2 - Ondersteunde input datatypes.

| **Oracle Spatial** | **DB2 Spatial Extender** |
|---|---|
| well-known text | well-known text |
| well-known binary | well-known binary |
| geography markup language | geography markup language |
| coördinaten | shapefile (ESRI) |
| .dmp file* (Oracle) | shapefile* (ESRI) |
| | coördinaten |
| | spatial data engine* (ESRI) |

Tabel 7.3: Oracle versus DB2 - Ondersteunde output datatypes.

Tabel 7.2 toont de datatypes waarvan rechtstreeks gegevens ingeladen kunnen worden. De datatypes met een asterisk (*) moeten met behulp van een import/export commando getransporteerd worden. De types zonder asterisk kunnen voorkomen als resultaat van een SQL expressie. Oracle lijkt hier lichtjes in het nadeel, maar omdat het 'well-known text' en 'well-known binary' formaat open standaarden zijn kunnen de meeste toepassingen die met andere datatypes werken hiernaar converteren. Merk op dat de 'geography markup language' ook een open standaard is maar deze niet door Oracle kan ingeladen worden. Een andere opmerking is dat de goede ondersteuning van DB2 voor de ESRI types komt omdat IBM en ESRI hierover een gezamenlijke overeenkomst hebben afgesloten. Tabel 7.3 toont de datatypes waaruit rechtstreeks gegevens uitgevoerd kunnen worden. Merk op dat Oracle sinds versie $10g$ wel de 'geography markup language' ondersteund.

| **Oracle Spatial** | **DB2 Spatial Extender** |
|---|---|
| R-tree index | grid index (3 niveaus) |
| quadtree index | |

Tabel 7.4: Oracle versus DB2 - Indexen.

Van de indextypes in tabel 7.4 ondersteunen zowel de grid index van DB2 als de R-tree index van Oracle meerdere dimensies, maar zijn ze geoptimaliseerd voor twee dimensies. De (gedateerde) quadtree index in Oracle werkt, omwille van zijn constructie, slecht op twee dimensies. DB2 heeft ook nog een *spatial index advisor* die ervoor zorgt dat de grid 'past' op de gegevens waarop de index gedefinieerd is. Oracle bevat ook een spatial index advisor, maar deze is enkel nodig voor de quadtree en wordt bijna niet meer gebruikt. Voor een R-tree is geen advisor nodig omdat deze zichzelf balanceert en op de gegevens werkt en niet op de ruimte waarin de gegevens zich bevinden.

De operatoren in Oracle die de spatial index gebruiken zijn SDO_NN (nearest neighbours), SDO_Within_Distance, SDO_Filter en SDO_Relate. Om bij de laatste twee operatoren het topologische predikaat te bepalen moet een 'masker' opgegeven worden. Dit masker, samen met de overeenkomstige functies van DB2 die ook gebruik maken van een spatial index, is te zien in tabel 7.5. In tegenstelling tot bij Oracle is in DB2 voor iedere topologische relatie een aparte functie voorzien.

| **Oracle Spatial** | **DB2 Spatial Extender** |
|---|---|
| CONTAINS | ST_Contains |
| INSIDE | ST_Within |
| TOUCH | ST_Touches |
| EQUAL | ST_Equals |
| ANYINTERACT | ST_Intersects |
| OVERLAPBDYINTERSECT | ST_Overlaps |
| OVERLAPBDYDISJOINT | ST_Crosses |

Tabel 7.5: Oracle versus DB2 - Topologische maskers/functies.

De maskers 'COVERS', 'COVEREDBY' en 'ON' maken ook gebruik van de index, maar hebben geen overeenkomstige functie in DB2. De functies *ST_Distance*, *ST_MBRIntersects* en *ST_EnvIntersects* van DB2 gebruiken ook een index, maar hebben dan weer geen overeenkomstig masker in Oracle. Oracle bevat ook een paar functies die geen gebruik maken van een spatial index maar die toch overeenkomen met een functie uit DB2. Zo is er de *Relate* functie die hetzelfde masker kan hebben als *SDO_Relate* en de functie *SDO_Distance* die hetzelfde doet als de DB2 functie *ST_Distance*, met dit verschil dat in DB2 wel gebruik wordt gemaakt van een spatial index.

| **Oracle Spatial** | **DB2 Spatial Extender** |
|---|---|
| SDO_Buffer | ST_Buffer |
| SDO_Union | ST_Union |
| SDO_Intersection | ST_Intersection |
| SDO_Difference | ST_Difference |
| SDO_XOR | ST_SymDifference |
| SDO_ConvexHull | ST_ConvexHull |

Tabel 7.6: Oracle versus DB2 - Creatie functies.

Tabel 7.6 toont enkele spatial functies die nieuwe geometrieën creëren en in

tabel 7.7 zijn een paar functies te zien die eigenschappen van geometrieën opvragen. Hierbij valt het op dat, behalve voor het symmetrisch verschil, alleen de prefix van deze functienamen verschillen. Het opvragen van coordinaten van een punt gebeurt in Oracle door rechtstreeks de object velden hiervan aan te spreken, terwijl dit in DB2 met de functies *ST_X, ST_Y, ST_Z* en *ST_M* moet gedaan worden.

| **Oracle Spatial** | **DB2 Spatial Extender** |
|---|---|
| SDO_Length | ST_Length |
| SDO_Area | ST_Area |

Tabel 7.7: Oracle versus DB2 - Opvraag functies.

Zoals blijkt uit de voorgaande illustraties is de kernfunctionaliteit van *Oracle Spatial* en *DB2 Spatial Extender* vrij gelijkaardig. Enkele features die wel in Oracle terug te vinden zijn maar niet in DB2 zijn datatypes om topologische, netwerk en raster gegevens op te slaan. Ook extra in Oracle is de ondersteuning van geodetische data, om deze in DB2 te krijgen moet een andere *Extender* gebruikt worden. De trend in Oracle is dan ook het produceren van één groot, uitgebreid product, terwijl bij IBM de nadruk meer ligt op het maken van een performante toepassing met een specifieker toepassingsgebied.

## 7.2 ArcGIS versus Spatial DBMS

Omdat *DB2 Spatial Extender* en *Oracle Spatial* zo weinig van elkaar verschillen worden ze hier samen genomen onder de noemer *Spatial DBMS*. De versie van ArcGIS waarmee de spatial DBMS vergeleken wordt is versie 8.1 ([31] en [23]). Dit is, zoals de naam al doet vermoeden, een Geographic Information System. ArcGIS 8 (1999) is ontstaan uit de samenvoeging van het wijdverspreide ArcView 3.xx en het krachtige maar dure ArcInfo 7.xx, hierbij kregen beide programma's een restyling van de (lichtjes verouderde) GUI. In het uiteindelijke resultaat is het oude ArcView bijna niet meer te herkennen. ArcView 8 is een compleet vernieuwde en verbeterde toepassing en een integraal deel van de ArcGIS familie. Vanaf versie 8.1 (2001) voldoet ArcGIS ook aan de "OpenGIS Simple Features Implementation Specification for OLE/COM" [16].

ArcGIS kan, net zoals de spatial DBMS, overweg met ruimtelijke en alfanumerieke gegevens. Deze data kan zowel in bestanden als in een database

worden opgeslagen. Omdat ArcGIS de gegevens rechtstreeks benadert is er geen centraal datatype aanwezig, dit wordt echter gecompenseerd door een groot aantal ondersteunde dataformaten. Een overzicht hiervan wordt getoond in tabel 7.8.

| **ArcGIS** | **Spatial DBMS** |
|---|---|
| Coverages | well-known text |
| Shapefiles | well-known binary |
| Grids | geography markup language |
| TINs | shapefile (ESRI) |
| Images (verschillende formaten) | spatial data engine (ESRI) |
| Vector Product Format (VPF) files | .dmp file (Oracle) |
| CAD files (verschillende formaten) | |
| Tables (verschillende formaten) | |
| Geodatabases (verschillende DBMS) | |

Tabel 7.8: ArcGIS versus Spatial DBMS - Datatypes.

Alhoewel het niet de enige functie van een GIS is, is het visualiseren en presenteren van spatial data de meest opvallende eigenschap van ArcGIS. Om door gegevens te navigeren is er een GUI nodig. Deze GUI zorgt ervoor dat een aantal operaties gemakkelijker uitgevoerd kunnen worden dan in een spatial DBMS, een voorbeeld hiervan is het selecteren van een feature. Maar de GUI zorgt er tegelijkertijd ook voor dat de meer uitgebreide queries moeilijker uit te voeren zijn omdat ze bijvoorbeeld opgesplitst moeten worden of omdat er gewoonweg een andere logica dan in spatial DBMS gebruikt moet worden om een probleem op te lossen.

In ArcGIS is het ook mogelijk om op verschillende van de ondersteunde datatypes een spatial index te creëren. Zo kan er met ArcCatalog een spatial index gedefinieerd worden op een shapefile. Dit heeft een *.sbn* en een *.sbx* bestand als resultaat, maar omdat deze bestanden een gesloten formaat hebben/niet gedocumenteerd zijn door ESRI blijft het gissen naar de gebruikte indexmethode. Voor een geodatabase kan er een spatial index grid aangemaakt worden. Voor een *personal* geodatabase (een Microsoft Access bestand) kan maximum één rastergrootte gedefinieerd worden terwijl voor een *multi-user* geodatabase (via ArcSDE) tot drie verschillende rastergroottes opgegeven kunnen worden. Een laatste voorbeeld van een spatial index is voor een coverage, hierbij worden per featuretype twee bestanden *.*bn.adf* en *.*bx.adf* gemaakt waarbij de '*' vervangen moet worden door de overeenkomstige letter van het type ('*a*' voor arcs, '*p*' voor polygonen, '*t*' voor

annotatie en '$x$' voor punten). Net zoals bij de shapefile het geval is, is het formaat van deze bestanden niet gedocumenteerd door ESRI en bijgevolg kan de indexmethode hiervan niet bepaald worden.

Een spatial index kan in verschillende situaties door ArcGIS gebruikt worden. Bijvoorbeeld bij de visualisatie van een subset van gegevens (met andere woorden als er ingezoomd wordt), bij het editeren van een layer, bij het opvragen van attributen van een feature met de 'Identify' tool van ArcMap, bij het uitvoeren van een spatial join en bij het analyseren van ruimtelijke gegevens.

Omdat ArcGIS een behoorlijk aantal datatypes ondersteund zijn er verschillende functies voorzien om hiertussen te converteren. Het is echter niet mogelijk om tussen elke twee types te converteren, dit komt omdat de formaten gewoonweg niet compatibel zijn met elkaar. Wel moet deze conversie voor een ganse layer ineens gebeuren, om een aantal (geselecteerde) features naar een ander formaat om te zetten is het nodig hiervan eerst een aparte layer te maken. Wat opvalt is dat ArcGIS geen ondersteuning biedt voor de door OpenGIS gedefinieerde open formaten, namelijk het well-known text formaat, het well-known binary formaat en de geography markup language.

In ArcGIS kunnen geometrieën geselecteerd worden aan de hand van de volgende (topologische) predikaten: *intersect, within distance, completely contain, completely within, center in, share line segment, touch boundary, identical, crossed by outline, contain* en *contained by.* Net zoals bij DB2 en Oracle het geval is, komen deze predikaten voor een groot stuk overeen met die van een spatial DBMS. Het verschil hierbij is dat *alle* objecten in één (of meerdere) layer(s) vergeleken worden met de (geselecteerde) feature(s) in een andere layer en een predikaat niet rechtstreeks tussen twee geometrieën kan getest worden.

Het editeren van geometrieën is een operatie die gemakkelijker in een GIS gedaan kan worden dan in een spatial database. Dit komt omwille van de GUI die bij ArcGIS gebruikt kan worden tegenover het gebruik van commando's in DB2 en Oracle. Voor de meeste spatial DBMS bestaan er echter ook grafische editors die het editeren van ruimtelijke gegevens vergemakkelijkt.

Met ArcGIS kunnen ook nieuwe geometrieën gemaakt worden met behulp van bestaande. Dit is gaat echter alleen maar voor een volledige layer ineens en niet voor specifieke objecten. De ondersteunde operaties zijn: *buffer, dissolve, merge, clip, intersect* en *union.* Een spatial DBMS ondersteunt naast

deze operaties ook nog enkele andere, zoals het verschil, het symmetrisch verschil, de aggregatie van verschillende objecten tot één geometry en het creëren van een *convex hull*.

Uit deze vergelijking is duidelijk dat een GIS applicatie, zoals *ArcGIS*, niet hetzelfde is of kan doen als een spatial database systeem, zoals *DB2 Spatial Extender* of *Oracle Spatial*. Aan de ene kant is ArcGIS door zijn GUI beperkt in het opstellen van queries, terwijl de spatial DBMS de volle kracht van SQL kunnen gebruiken. Maar aan de andere kant ondersteunt ArcGIS meer datatypes dan de spatial DBMS, biedt het (betere) tools om ruimtelijke gegevens te bewerken en laat het toe om gegevens en resultaten van analyses hierop te presenteren. Enkele tekortkomingen van de *file-based* gegevensopslag in een GIS-applicatie zijn met de introductie van de geodatabase weggewerkt. Zo kunnen grote hoeveelheden ruimtelijke en alfanumerieke gegevens *seamlessly* opgeslagen worden en wordt er automatisch in *recovery* voorzien.

Algemeen genomen kan een (spatial) database systeem uitstekend gebruikt worden voor de implementatie van een GIS toepassing. Hierdoor worden alle voordelen van een database systeem verder gepropageerd naar het GIS systeem, terwijl het zijn volledige functionaliteit behoudt. Een spatial database systeem kan echter voor meer gebruikt worden dan alleen GIS applicaties. Zo kunnen zijn positieve eigenschappen, zoals querying, optimalisaties en recovery, ook gebruikt worden bij de implementatie van een *Global Positioning System* (GPS) en bij *Remote Sensing*. Een spatial database systeem is echter absoluut *niet* bedoeld voor eindgebruikers. Het is aan de ontwikkelaars van GIS software en/of andere toepassingen om met behulp van een spatial database systeem een gebruiksvriendelijk eindproduct uit te bouwen dat een passend antwoord biedt op de vragen van de eindgebruiker.

# Lijst van figuren

# Lijst van tabellen

# Lijst van listings

# Bibliografie


[1] Ron Briggs. *Fundamental GIS Concepts*, 2005. University of Texas.

[2] Dare Obasanjo. An Exploration Of Object Oriented DBMS. www.25hoursaday.com/WhyArentYouUsingAnOODBMS.html.

[3] De Maeyer. *Inleiding GIS*, 2004. Universiteit Gent - Vakgroep Geografie.

[4] ESRI. *ESRI Shapefile Technical Description*, July 1998.

[5] Hector Garcia-Molina, Jeffrey D. Ullman and Jennifer Widom. *Database Systems - The Complete Book*. Prentice Hall, 2002.

[6] Hendriks P. and Ottens H. *Geografische Informatie Systemen in ruimtelijk onderzoek* . 1997.

[7] IBM. *DB2 Spatial Extender and Geodetic Extender (Version 8.2) - Users Guide and Reference*.

[8] Information Technology Service. *Introduction to GIS using ArcGIS (Version 6.0)*. Durham University, August 2005.

[9] Jan Paredaens and Bart Kuijpers. Data Models and Query Languages for Spatial Databases. *Data & Knowledge Engineering*, 25:29–53, 1998.

[10] Jo Clarke and Jenny Mitcham. *GIS Preservation Handbook*. Arts and Humanities Data Service, $5^{th}$ edition, June 2005.

[11] Knut Stolze. *SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems*. International Organization for Standardization.

[12] M.Molenaar, O.Kufoniyi and T.Bouloucos. Modelling Topologic Relationships in Vector Maps. In *Proc. Intl. Symp. on Spatial Data Handeling (SDH)*, 1994.

[13] $O_2$. www.o2tech.fr.

[14] Object Store. www.odi.com.

[15] Ozone. www.ozone-db.org.

[16] Open GIS Consortium, Inc. *OpenGIS Simple Features Implementation Specification for OLE/COM (Revision 1.1)*, May 1999.

[17] Open GIS Consortium, Inc. *OpenGIS Simple Features Specification For SQL (Revision 1.1)*, May 1999.

[18] Open GIS Consortium, Inc. *OpenGIS Geography Markup Language (GML) Implementation Specification (Version 3.00)*, January 2003.

[19] Oracle. *Oracle Spatial (Release 10.1) - Users Guide and Reference*, December 2003.

[20] Federal InterAgency Coordinating Committee on Digital Cartography / Federal Geographic Data Committee. www.fgdc.gov.

[21] Ondersteunend Centrum GIS-Vlaanderen. web.gisvlaanderen.be.

[22] International Organization for Standardization. www.iso.org.

[23] Open GIS Consortium, Inc. www.opengeospatial.org.

[24] Tele Atlas. www.teleatlas.com.

[25] Philippe Rigaux, Michel Scholl and Agnes Voisard. *Spatial Databases with Application to GIS*. Morgan Kaufmann Publishers, 2002.

[26] The Arthur H. Robinson Map Library at the University of Wisconsin-Madison. http://www.geography.wisc.edu/maplib/rob_proj.html.

[27] Map Projections poster. http://ioc.unesco.org/oceanteacher/resourcekit /M3/Formats/ProjectionsPoster/Map Projections Poster.htm.

[28] Ralf Hartmut Güting and Markus Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. 1993.

[29] Ravi Kothuri, Albert Godfrind and Euro Beinat. *Pro Oracle Spatial*. APress, 2004.

[30] Scott Crosier, Bob Booth and Andy Mitchell. *Getting Started with ArcGIS*. Redlands, 2002.

[31] ESRI ArcGIS. www.esri.com/software/arcgis.

[32] IBM - DB2 Spatial Extender. www-306.ibm.com/software/data/spatial.

[33] Oracle Spatial. www.oracle.com/technology/products/spatial.

[34] Tele Atlas. *Tele Atlas MultiNet Shapefile 4.1 Format Specifications (Version 1.2)*, September 2003.