Examensarbete i geografisk informationsteknik nr 13

# Integrering av BIM och GIS med spatiala databaser

## – En prestandaanalys

**Martin Fredriksson**

*Civilingenjörsutbildningen i Lantmäteri*

Lunds Tekniska Högskola

Institutionen för Naturgeografi och Ekosystemvetenskap

Lunds Universitet

LUNDS UNIVERSITET
Lunds Tekniska Högskola

# Integrering av BIM och GIS med spatiala databaser

## – En prestandaanalys

---

EXTM05 Master uppsats, 30 hp

*Civilingenjörsutbildningen i Lantmäteri*

Martin Fredriksson

Handledare: Lars Harrie

Institutionen för Naturgeografi och Ekosystemvetenskap

16 januari 2014

Opponent: Nariman Emamian
Examinator: Jonathan Seaquist

Nyckelord: GIS, BIM, spatiala databaser, spatiala indexeringsmetoder, prestandaanalys

Key words: GIS, BIM, spatial databases, spatial index methods, performance analyse



Institutionen för Naturgeografi och Ekosystemvetenskaper
Lunds Universitet
Sölvegatan 12
223 62 Lund

Telefon: 046-222 30 30
Fax: 046-222 03 21
Hemsida: http://www.nateko.lu.se

Examensarbete i geografisk informationsteknik nr 13
Tryckt av E-tryck, E-huset, 2014

# Förord

Denna studie är resultatet av det avslutande examensarbetet för civilingenjörsutbildningen inom Lantmäteri med inriktning geografisk informationsteknik vid Lund tekniska högskola. Arbetet utfördes på institutionen för Naturgeografi och Ekosystemvetenskaper vid Lunds Universitet samt med ett nära sammarbete med Sweco Position i Stockholm.

Jag vill tacka min handledare Anton Sandström på Sweco som har bidragit med viktig kunskap och feedback under arbetets gång och som gjorde detta examensarbete möjligt. Ett tack vill jag även rikta till alla andra personer på Sweco som har tagit sig tid att svara på otaliga frågor och hjälpa mig med deras oumbärliga kunskap och erfarenheter.

Sist men inte minst vill jag tacka min akademiska handledare Lars Harrie som under hela arbetets gång varit ett stort stöd för både akademiska och tekniska frågor.

2014-01-16
Martin Fredriksson

# Förkortningsordlista

ANSI = American National Standards Institute

API = Application Programming Interface

ASCII = American Standard Code for Information Interchange

BIM = Building Information Model

CAD = Computer -aided design

CLR = Common Language Runtime

DDL = Data Definition Language

DML = Data Manipulation Language

FME = Feature Manipulation Engine

GIS = Geographic Information Model

GiST = Generalized Search Tree

GML = Geography Markup Language

HTML = Hypertext Markup Language

IFC = Industry Foundation Classes

ISO = International Organization for Standardization

KML = Keyhole Markup Language

LOD = Level of Details

MBB = Minimum Bounding Box

NF = Normalization Form

OGC = Open Geospatial Consortium

ODL = Object Definition Language

ODMG = Object Data Management Group

OMG = Object Management Group

OQL = Object Query Language

SDL = Storage Definition Language

SLD = Styled Layer Descriptor

SQL = Structured Query Language

UBM = Unified Building Model

UML = Unified Modeling Language

VDL = View Definition Language

WMS = Web Map Service

XML = Extensible Markup Language

# Abstract

The aim of this project is to determine the possibility to store BIM data in spatial databases to establish high performance web based GIS. Building data of Stockholm Arlanda Airport are used to create a web based GIS. The GIS has a tripartite system architecture consisting of spatial database, map server and client. The main part of the project is to determine the impact of the spatial database in the overall performance of the GIS. The project also investigates how databases can be adapted e.g. by altering the spatial index structures to better work with the difficulties of BIM data.

A problem with BIM data is that they are developed by other standards than “ordinary” GIS data. The main problem with the input data is that they are not geographically referenced since they originate from CAD files, drawn in local coordinate system. This results in that the majority of the data are located within the same geographical bounding box. Both Data- and Spatial-driven spatial index structures are dependent of the objects position in the area. If many objects overlap each other the index structure receives a low efficiency which results in low performance of the entire GIS.

The project analyse different spatial index structures to determine which strengths and weaknesses they have for this and similar scenarios. PostGIS and SQL Server implements different spatial index methods and are the two database products analysed and tested in this project. Tests are conducted to determine the performance of the entire GIS by measuring the response time to visualize chosen floors of buildings in the client.

GeoServer is used as map server in the tripartite system. It is default to use a map server when creating GIS. The project shows that GeoServer always generate spatial SQL queries from the WMS statement. When working with BIM data it is not always necessary to use spatial SQL queries to get the correct answer. For e.g. the data objects in this project all have a floor number-attribute which could be used in a non-spatial SQL query to generate the correct answer faster.

The results of the tests show that there is a big difference in performance of tripartite systems depending on which spatial database product that is used. The main object is that the systems have shorter response time when using Data-driven index structures instead of Space-driven. The main reason is that Data-driven structures consist of balanced search trees and thus are more adaptable when working with data which have skew spatial distribution.

The conclusions are that it is possible to create a functional and efficient GIS containing BIM data by using spatial databases. It is important that the input data are geographically referenced since spatial databases need to be able to index the data to maintain high performance. Furthermore, it is recommended to use spatial database products which implements Data-driven spatial index structures since they have a more dynamic design and are adaptable by the location of the objects than Space-driven structures.

## Sammanfattning

Målet med denna studie är att undersöka möjligheten att lagra och hantera BIM-data i spatiala databaser för att upprätta funktionell webbaserad GIS över datamängden. Studien genomförs genom att upprätta ett webbaserat GIS innehållandes byggnadsdata över Stockholm Arlanda Airport. Systemet upprättas med en tredelad systemarkitektur med spatial databas, kartserver och klient. Fokus i studien är att avgöra vilken inverkan den spatiala databasen har på systemet samt avgöra hur databasen kan anpassas, t.ex. med olika indexeringsstrukturer, efter de utmaningar BIM-data medför.

Ett stort problem med BIM-data är att de är uppbyggda efter annan standard än ”vanlig” GIS-data. Det huvudsakliga problemet är att data inte är geografiskt refererade utan är ursprungligen är CAD-filer ritade i lokala koordinatsystem. När alla data läggs in i samma databas är majoriteten av dem belägna inom samma geografiska utbredning. Både objekt- och rumsligt baserade spatiala indexeringsmetoder är beroende av objektens placering i rummet. Om många objekt överlappar varandra påverkar det indexeringsstrukturen negativt och systemen uppvisar dåliga prestanda.

Studien undersöker flera spatiala indexeringsmetoder för att avgöra vilka för- och nackdelar de olika metoderna har i detta och likande scenarier. PostGIS och SQL Server implementerar olika indexeringsmetoder och är de spatiala databasprodukter som testas i studien. Tester utförs för att avgöra systemets fullständiga prestanda genom att mäta svarstider för att visualisera valda byggnadsvåningar i klienten.

GeoServer används som kartserver i det tredelade systemet. Det är standard att använda en kartserver vid upprättande av GIS. Studien visar att GeoServer alltid formulerar spatiala SQL-frågor till databasen vilket vid hantering av BIM-data inte alltid behöver vara lämpligt för att returnera önskad datamängd. Objekten i studien har t.ex. ett våningsattribut som skulle kunna användas i icke-spatial SQL-fråga för att snabbare generera korrekt svar.

Resultaten av prestandatesterna visar att det är stor skillnad i prestanda för system med tredelad systemarkitektur beroende på vilken databasprodukt som används. Ett tydligt resultat är att objektbaserade indexeringsstrukturer genererar kortare svarstider än rumsligt baserade. Den främsta anledningen är att objektbaserade strukturer utgörs av balanserade sökträd och är därmed mer anpassningsbara vid hantering av data med skev spatial fördelning.

Slutsatserna är att det går att skapa funktionella och effektiva GIS över BIM-data med hjälp av spatiala databaser. För att tillhandahålla god prestanda är det av stor vikt att indata är geografiskt refererat eftersom effektiv hantering av spatiala data i databaser kräver att data är möjlig att indexera. Vidare rekommenderas det att använda spatiala databasprodukter som tillämpar objektbaserade indexeringsstrukturer eftersom de har en mer dynamiska utformning samt är mer anpassningsbara efter objektens placering i rummet.

# Innehållsförteckning

# 1 Inledning

## 1.1 Bakgrund

Att få geografisk information visualiserad på ett enkelt och lättillgängligt vis är en självklarhet i dagens samhälle. En tydlig indikation är att en navigeringsapplikation toppar listan över de mest använda applikationerna för smarta telefoner (Mashable 2013). Att informationen ska vara lättillgänglig, korrekt och aktuell ställer höga krav på lagringen och hantering av de geografiska data som visualiseras. Inte bara korrekt och detaljerad information ska visualiseras på ett tydligt och enkelt sätt, systemet ska även tillhandahålla informationen snabbt vilket ställer höga krav på prestandan.

En relativt ny nisch inom kartering är webbaserade inomhuskartor. Gallerior, sjukhus och flygplatser m.fl. har insett de fördelar inomhuskartor inbringar. De kan användas både som navigeringsstöd för besökare och för att skapa GIS som förvaltningsstöd. Inomhuskartor ställer helt nya krav på datainsamlingen. Tekniska hjälpmedel som flygfotogrammetri och luftburen laserscanning är inte tillämpbara. Däremot finns det ofta extremt detaljerade tekniska ritningar som kan användas som stöd vid kartframställningen. Ritningar är ofta upprättade i ett lokalt koordinatsystem utan koppling till något befintligt geografiskt referenssystem. Visserligen behöver inte GIS över specifika och begränsade områden något geografiskt referenssystem men för att möjliggöra interaktion med omvärlden är det nödvändigt.

Det finns idag flera system för att lagra, behandla och dela geografisk data, både kommersiella och system med öppen källkod (eng. *Open Source*). Grunden i dessa system är hur de lagrar och hanterar data i databasen. Utan en väl strukturerad och organiserad databas fallerar lätt hela systemet. Många av de stora databaserna på marknaden följer gemensamma (ISO-) standarder och direktiv, t.ex. SQL, men är ändå olika uppbyggda. Att det finns flera olika system visar att det är svårt att upprätta en lösning som är effektivt för alla potentiella användningsområden. Vid upprättandet av geografiska databassystem är det viktigt att noga analysera vilka behov och krav som ställs på databasen för att på så vis kunna upprätta en, för den aktuella situationen, funktionell och effektiv lösning.

## 1.2 Problemställning

Swedavia, som äger och driver flera flygplatser i Sverige, är intresserade av att skapa inomhuskartor över sina byggnader för att kunna använda dem i deras webbaserade GIS. Det finns i dagsläget inomhusritningar över byggnaderna som är konverterade och inlagda i deras databas (SQL Server 2008). Målet med deras system är i första hand förvaltningsstöd men i framtiden kunna användas som grund till ett inomhusnavigeringssystem, både för anställda och besökare. Tillämpningen av systemet är främst att visualisera samtliga objekt med tillhörande information om dem för fullständiga våningsplan.

Prestandan i det befintliga systemet är så dålig att systemet blir i stort sett obrukbart, d.v.s. svarstiden för att visualisera informationen är väldigt lång. Studiens mål är att öka förståelsen och kunskapen om vilka omständigheter i databassystem som har störst negativ inverkan på prestandan och hur man kan undvika dem vid integrering av BIM och GIS.

Byggnadsdata (BIM-data) skiljer sig ofta från ”vanlig” GIS-data. Ett stort problem är att BIM-data ofta inte är geografiskt refererade utan är endast utritade i lokala koordinatsystem. Vidare består byggnader inte sällan av flera våningar vilket resulterar i att datamängden blir väldigt kompakt i vissa begränsade områden och därmed ofta skevt fördelad i rummet. Databasernas prestanda är direkt beroende på hur data är lagrade och vilka funktioner som ska stödjas. Studien ska undersöka och analysera hur olika spatiala databasprodukter skiljer sig åt och deras styrkor respektive svagheter/brister i liknande scenarior.

## 1.3 Syfte

Denna studies syfte är att undersöka hur BIM-data effektivt kan lagras och hanteras i spatiala databaser för att möjliggöra webbaserad visualisering med hög prestanda. Studien ska främst undersöka den spatiala databasens inverkan på prestandan hos webbaserade GIS med tredelad systemarkitektur.

Studien ska vidare undersöka följande faktorers inverkan på systemets prestanda vid hantering av BIM-data:

- val av indexeringsstruktur
- datamängdens uppbyggnad och fördelning i planet
- parameterinställningar i de spatiala databasprodukterna.

## 1.4 Metod

Projektet delas in i fyra delar: teoretisk studie, valda databassystem skapas och fylls med data, systemens prestanda testas och projektet avslutas med en utförlig utvärdering av systemen.

Del ett innehåller en utförlig litteraturstudie om teknisk bakgrund och förutsättningar. Flera spatiala databassystems uppbyggnad och funktioner granskas och tidigare liknade studier analyseras. Undersökningen ligger till grund för beslutet om vilka databassystem som ska utvärderas och hur utvärderingen ska ske.

Del två innebär att de utvalda databaserna skapas och fylls med aktuella och identiska data. Systemen tillämpar en standard GIS systemarkitektur som kopplar samman klienten med databasen genom användning av kartserver. Genom att använda en extern kartserver testas hela systemets prestanda, från datahantering i databasen till visualisering i klienten.

I del tre utförs och dokumenteras prestandatester och analyser på de spatiala databaserna samt på hela systemarkitekturen. Databaserna inställningar kalibreras efter insikter om prestandaoptimering inhämtade i litteraturstudien.

Under den fjärde och sista delen analyseras och utvärderas resultaten från prestanda-optimerade åtgärderna. Riktlinjer och tips redovisas för val av spatiala databassystem vid integrering av BIM och GIS.

Projektet sker i sammarbete med Sweco Position i Stockholm samt Swedavia som tillhandahåller data.

## 1.5 Disposition

Rapporten inleds med en genomgång av tidigare studier inom det aktuella ämnet. Dels för att undersöka om det finns en färdig applicerbar lösning men även för att undvika att ”uppfinna hjulet igen” om någon redan har gjort en liknande studie. Kapitel 3 beskriver Swedavias systemarkitektur och tillämpningar samt vilka problem som har uppmärksammats och föranlett denna studie.

Kapitel 4 beskriver problematiken som uppstår vid försök av integrering av de två skilda systemen, BIM och GIS. Kapitel 5,6 och 7 presenterar de tekniska förutsättningar som finns tillgänglig med dagens teknik för att lösa det aktuella problemet. En anledning till denna utförliga tekniska beskrivning är att lösningen av det aktuella problemet ska generaliseras och därmed kunna appliceras i liknande scenarier. För att uppnå detta krävs en gedigen kunskap om de tekniska förutsättningarna. Kapitel 8 beskriver de tekniska förutsättningarna för två av marknadens största spatiala databasprodukter. Databasprodukternas tekniska förutsättningar jämföras och analyseras för att se hur de kan användas för att lösa problemet.

Kapitel 9 beskriver problematiken kring utformningen av obehandlade BIM-data och förklarar tillvägagångssättet för den förenklade geografiska refereringen. I kapitlet anges vilka mjukvaror som används för att genomföra testerna. Den grundläggande metodiken av utformningen av prestandatesterna beskrivs och motiveras i detta kapitel.

Kapitel 10 och 11 innehåller teknisk genomgång av prestandatesterna samt resultatet av dessa tester. Prestandatesterna presenteras per databasprodukt för att skapa en tydlig och lättläst struktur av rapporten. I kapitel 12 sammanfattas resultaten från databasprodukterna på ett lättöverskådligt vis.

Rapporten avslutas med att resultatet diskuteras och granskas. Slutsatser, lärdomar och tips gås igenom för att förenkla arbetet med vidare utveckling av liknande scenarier.

# 2 Tidigare studier

Tidigare studier innehållandes prestandaanalyser av spatiala databasprodukter har utförts. De flesta studierna undersöker prestandan hos en eller två databasprodukter vid specifika spatiala frågeställningar eller spatiala sammanslagningar (eng. *join*). Kothuri m.fl. 2002 jämför och anlyserar prestandan för indexeringsmetoderna Quad- och R-träd inom *Oracle Spatial* databasprodukt (Kothuri m.fl 2002). Fokus i deras studie behandlar prestandamätning för flera olika spatiala SQL-frågor, t.ex. *inside*, *equal* och *sdo_nn* (en närmaste granne-fråga). Anledningen till att jämföra prestandan vid olika SQL-frågor är för att indexeringarna bygger på två olika strukturer. Quad-träd är rumsligt baserad och R-träd är objektbaserad. Beroende på vilken spatial fråga som ställs kan det vara fördelaktigt med olika utformade indexeringar.

Ooi m.fl. 1993 jämför en mängd olika spatiala indexeringsmetoder för att avgöra om det är fördelaktigt att använda särskilda indexeringsmetoder vid olika typer av spatiala frågeställningar och sammanslagningar (Ooi m.fl. 1993). Fokus i testerna är att avgöra effektiviteten för de olika indexeringstyperna beroende av tre parametrar: antal objekt per area, objektens storlek och databasens storlek.

Sarda och Subham har gjort en omfattande studie som behandlar en jämförselse av prestandan hos Oracle Spatial 11g och PostGIS 1.5 (Sarda och Subham 2011). Sarda och Subham testar databasprodukternas prestanda utifrån flera spatiala SQL-frågor. De analyserar prestandan i två olika scenarier, s.k. kallt och varmt scenario (eng. *hot phase* och *cold phase*). I det kalla scenariot isoleras testerna och ger inte databasernas frågeoptimerare tillgång till någon statistik. I det varma scenariot har frågeoptimeraren tillgång till fullständig statistik. I denna studie försöker inte Sarda och Subham att optimera några inställningar eller indexeringsstrukturen utan analyserar prestandan vid användning av standard indexeringsstruktur. Sarda och Subham är intresserade av hur databasprodukterna hanterar olika spatiala frågor och vilken databasprodukt som snabbast returnerar korrekt svar.

Jackson 2011 undersöker prestandan hos SQL Server beroende av vilken celldensitet som används vid upprättande av spatiala index (Jacksson 2011). Fokus i hans undersökning är att undersöka vilken celldensitet som genererar bäst indexeringseffektivitet vid lagring av spatiala data. Testerna utförs på datamängder med varierande geografisk utbredning och spatial densitet.

# 3 Swedavia

Swedavia som äger och driver flera flygplatser i Sverige har digitaliserat sitt förvaltningssystem över Stockholm Arlanda Airport i form av ett inomhus-GIS. Grunddata består av CAD-ritningar och flera förvaltningsattribut, som t.ex. ombyggnadsstatus och brandutrustning. Målet är att lagra samtliga data i en gemensam spatial databas. En gemensam spatial databas effektiviserar och underlättar hanteringen av inomhus-GIS och möjliggör utveckling mot inomhusnavigering, vilket är nästa steg i utvecklingen.

## 3.1 Tillämpning

Tillämpningen av Swedavias inomhus-GIS är huvudsakligen förvaltningsstöd. Både för att underlätta och effektivisera hanteringen av digitala data genom att data lagras inom en och samma databas. Men även för att underlätta underhåll och felrapportering av fysiska objekt. Inomhus-GIS kan t.ex. innehålla information vilka företag hyr vilka lokaler samt var och hur länge underhållsarbete utförs.

## 3.2 Befintlig arkitektur

Swedavia använder idag en tredelad webbaserad systemarkitektur med delarna klient, kartserver och databas. Klientdelen exekveras direkt i webbläsaren. Autodesk Infrastrukture Map Service används som kartserver och SQL Server (version 2008 R2) är den spatiala objektrelationella databasen som används.

I SQL Server hanteras och lagras alla data utom de spatiala linjeobjekten. Linjeobjekten lagras inte i databasen eftersom prestandatester uppvisar oacceptabelt långa svarstider vid anrop av en samling linjeobjekt från databasen. Det har resulterat i att linjeobjekten lagras i ett separat filbaserat lagringssystem. Denna tillfälliga lösning medför merarbete vid t.ex. underhåll av digitala data eftersom de är lagrade i olika system.

## 3.3 Problem

Det huvudsakliga problemet är att svarstiderna är för långa och uppfyller inte prestandakraven för delar av data vid användning av ett gemensamt databassystem.

En anledning kan vara att objekten i ritningarna inte är geografiskt refererade och de är godtyckligt placerade i ett lokalt koordinatsystem. Objektens placering och utbredning påverkar indexeringsstrukturen. Även uppbyggnaden av data kan påverka prestandan negativt, alla mått är t.ex. uttryckta i millimeter och vissa objekt har negativa koordinatvärden.

# 4 Integrering av BIM och GIS

BIM (*Building Information Model*) och GIS hanterar båda lägesbundna data. Dock har utvecklingen kring de båda systemen skett var för sig och i olika riktningar. BIM har fokuserat på småskaliga, detaljrika och tekniska ritningar och modeller över byggnader eller våningsplan. BIM kan t.ex. innehålla specifika objekts konstruktion och bärighet. GIS hanterar ofta data med större geografisk utbredning. Det har i modern tid blivit aktuellt att för de olika systemen ta del av varandras data och datahanteringslösningar. Data från BIM blir aktuellt att använda i GIS vid t.ex. skapandet av inomhuskartor. GIS har även tagit steget mot att bygga mer detaljrika och tredimensionella modeller. CityGML är ett verktyg för att modellera tredimensionella geografiskt bundna objekt och används idag för att utveckla tredimensionella system över t.ex. städer (CityGML 2012).

## 4.1 BIM

BIM är ett samlingsbegrepp för att hantera och modellera digitala byggnads- och anläggningsdata. Målet med BIM är att samla alla byggnads- och anläggningsrelaterade data i samma system. Både för att underlätta hanteringen av data men även öka interoperabiliteten för de olika delarna av byggnadsprojekt. System behöver bra struktur för att klassas som ett BIM-system, t.ex. ska BIM-system bestå av objektorienterade modeller med egenskaper och attribut kopplade till objekten, relationer mellan objekt ska stödjas och det ska vara möjligt att producera specifika informationsvyer av modellerna (OpenBIM 2013).

BuildingSMART som är en internationell ideell organisation som arbetar för att utveckla öppna standarder inom BIM (BuildingSMART 2013). Målet med BuildingSMART är att öka interoperabiliteten mellan arkitekt-, ingenjör- och byggnadsbranscherna genom gemensamma standarder. IFC (*Industry Foundation Classes*) är ett mjukvaruoberoende dataformat för hantering och delning av BIM-relaterade data, utvecklad av BuildingSMART. IFC är sedan år 2013 en accepterad ISO-standard (*International Organization for Standardization* (ISO 16739:2013) (BuildingSMART 2013).

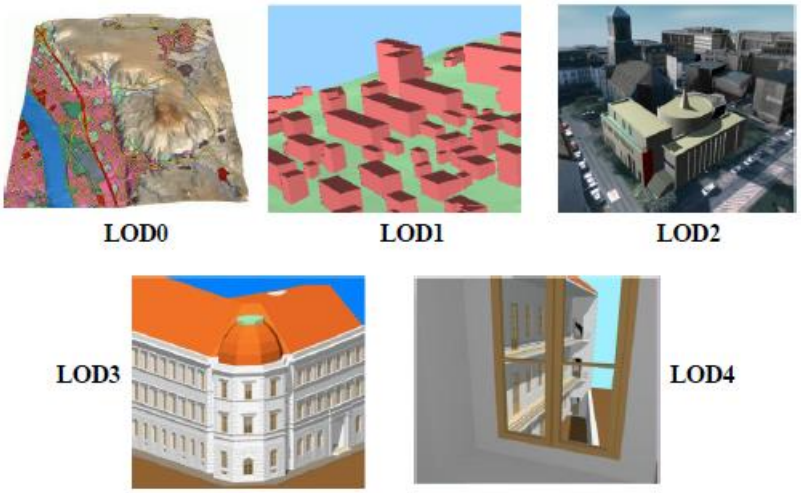
BIM används ofta i planerings- och utförandestadiet för att underlätta hanterningen och öka kontrollen över projektets samtliga delar. Beroende på anläggning innehåller modellen olika data. Det kan vara allt ifrån enkla tvådimensionella ritningar till komplexa tredimensionella modeller/simuleringar.

När byggnationerna är avslutade vidtar det viktiga förvaltningsarbetet vars mål är att på ett kostnadseffektivt vis sköta drift och underhåll. Både av digitala data med framför allt av de fysiska objekten. För att kunna planera och följa upp drift och underhåll för byggnadens alla delar är en digital representation ett lämpligt hjälpmedel. Viktig information vid drift- och underhållsarbete som var och vilket arbete som ska utföras kan då enkelt regleras i databasen för att effektivt kunna planera åtgärderna.

Som planeringsstöd är en karta ofta ett bättre verktyg än en teknisk och detaljerad ritning. För att ändå hantera alla grundläggande detaljrik digitala data är ett databassystem att föredra vid kartframställning framför ett filbaserat lagringssystem med separata ritningsfiler. Databaser kan t.ex. enkelt integreras med en kartserver som i sin tur kan användas vid delning och visualisering av spatiala data.

## 4.2 CityGML

CityGML är en OGC (*Open Geospatial Consortium*) standard för lagring av stadsmodeller. Den är en utvekling av GML (*Geography Markup Language*) som är ett öppet dataformat baserad på XML (*Extensible Markup Language)* (OGC 2012). GML är en accepterad ISO-standard (ISO 19136:2007) för att skapa och hantera geometriska figurer i webbaserade GIS. CityGML används för att utveckla tredimensionella modeller över byggnader och städer. Modellerna upprättas med hjälp av fem detaljnivåer (eng. *Level of Details*(LOD)), från översiktliga storskaliga terrängmodeller till modellering av arkitektuell design, se Figur 4.1. Det är med andra ord möjligt att skapa fullständiga och detaljrika stadsmodeller uppbyggda av korrekta terrängmodeller, vegetation- och vattenytor samt detaljrika byggnader.




**Figur 4.1. Beskrivning av de fem detaljnivåerna (LOD 0 - LOD 4) (OGC 2012).**

Studier pågår för integrering av data från CityGML och BIM i t.ex. IFC format. Hijazi m.fl. 2009 undersöker möjligheten att integrera CityGML och IFC och därmed skapa ett enklare sammarbete inom de två branscherna (Hijazi m.fl. 2009). Tidigare har data transformerats till önskat format, antingen till BIM eller GIS-modeller. I studien undersöker Hijazi m.fl. möjligheten att använda IFC-format för att integrera BIM och GIS.

El-Mekaway m.fl. 2011 undersöker möjligheten att använda en gemensam plattform, UBM (*Unified Building Model*), som ett medium för att konvertera data mellan BIM och GIS (El-Mekaway m.fl. 2011). Oavsett ursprung av data konverteras de först till UBM för att sedan konverteras till önskat format.

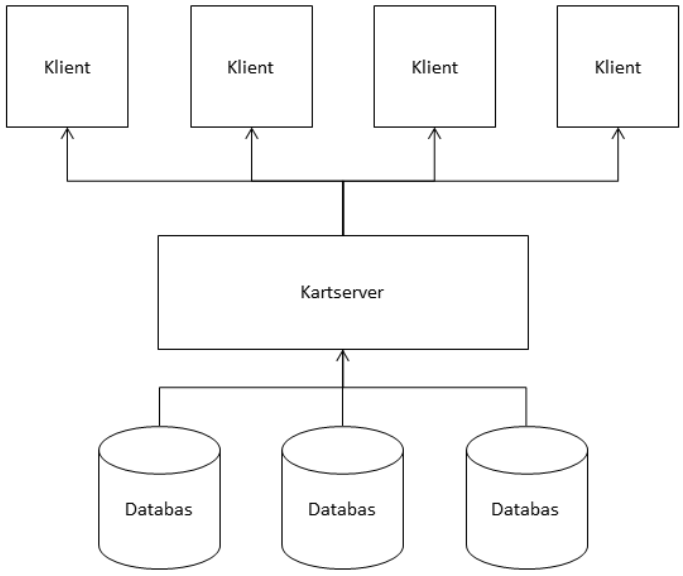
Det har även utförts studier som undersöker möjligheten att integrera BIM och GIS och därmed öka interoperabiliten mellan systemen. Många studier inom ämnet BIM och GIS behandlar möjligheten att visualisera GIS-data tredimensionellt, t.ex. med hjälp av CityGML. Det är inte många studier som behandlar problemet att gå från tredimensionella ritningar till tvådimensionella kartor. I dagens samhälle uppskattas de visuella effekterna som uppkommer vid tredimensionella modeller. Få inser funktionaliteten en ”enkel” karta kan erbjuda.

# 5 Systemarkitektur

Systemarkitekturen är ofta unik för varje system, men viss övergripande och vedertagen arkitektur finns inom olika verksamhetsområden. I det här kapitlet beskrivs den s.k. tredelade systemarkitekturen vilken är den vanligaste för webbaserade GIS. Alternativa lösningar, med fokus på system utan en kartserver som mellanhand, till denna arkitektur diskuteras även i detta kapitel.

## 5.1 Tredelad systemarkitektur

Vanlig systemarkitektur för webbaserade GIS är en tredelad arkitektur som består av klienter, kartserver och spatiala databaser, se Figur 5.1. Klienter är slutanvändarna av systemet och kan ofta använda en vanlig webbläsare i en dator, surfplatta eller en smarttelefon. Kartserver fungerar som ett kommunikationsverktyg för att visualisera spatiala data från databaserna.




**Figur 5.1. Vanlig systemarkitektur för webbaserade GIS.**

### 5.1.1 Klienter

Klienterna används av slutanvändarna av systemet och i dagens ständigt uppkopplade samhälle kan klienterna vara utvecklade även för smarttelefoner. De flesta klienter exekveras direkt i webbläsaren och är utvecklade med främst HTML och JavaScript. Det finns klienter som kräver att extra program installeras (s.k. *plug-ins*), de klienterna är ofta mer kraftfulla.

### 5.1.2 Kartserver

En kartserver fungerar som ett kommunikationsverktyg mellan klient och databas för att möjliggöra webbaserad visualisering av geografiska data. Tabellerna i en (spatial) databas genereras som lager i kartservern. Ett eller flera lager kan anges i WMS-anrop av användaren för att visualiseras i klienten.

#### *5.1.2.1 Web Map Service (WMS)*

De flesta karttjänster stödjer WMS. WMS är en accepterad ISO-standard utvecklad av OGC för webbaserad visualisering av geografiska data (OGC 2006). WMS-anropens syntax är baserad på vanliga HTTP-syntax och returerar den sökta kartbilden i ett bildformat, t.ex. JPEG eller PNG.

För att hämta kartbilder från en kartjänst anges *GetMap* i det obligatoriska parameter *REQUEST* i WMS-anropet (OGC 2006). WMS-anrop innehåller flera olika obligatoriska- eller valfria parametrar beroende av vad som önskas visualiseras. Obligatoriska parametrar förutom *REQUEST* är bl.a. *BBOX*, *LAYERS* och *STYLES*. *BBOX* (*Bouding Box*) innehåller den geografiska utbredningen över området som ska visualiseras. *LAYERS* anger vilka lager som ska visualiseras och *STYLES* anger med vilken stil lager ska visualiseras (OGC 2006). Stilarna för lager kan anges som SLD-filer (*Styled Layer Descriptor*) som uttrycks i XML-format (OGC 2007). Stilar kan vara s.k. dynamiska och ändra utseende efter angivet parametervärde.

#### *5.1.2.2 SQL-anrop*

Som kommunikationsverktyg mellan klient och databas behöver kartservern formatera det standardiserade WMS-anropet till SQL-syntax för att möjliggöra kommunikation med databaserna. Beroende mot vilken databasprodukt *GetMap*-anrop sker så behöver SQL-frågan anpassas efter produktspecifika funktioner. Det medför att alla kartservrar inte kan hantera data från alla databasprodukter. För att möjliggöra visualisering av data från t.ex. SQL Server 2012 med hjälp av GeoServer behövs ett s.k. *plug-in* till GeoServer.

Nedan följer ett exempel av ett autogenererat *GetMap*-anrop från GeoServer och den resulterande spatiala SQL-frågan efter SQL-transformation från GeoServer mot SQL Server. Lagret som önskas visualiseras är en vy av våning 123 från polygonobjektstabellen (*Rumsyta*). Eftersom det är en vy som anropas syns inte vånings id i *WHERE*-klausulen. *WHERE*-klausulen i SQL-frågan använder *Filter*-funktionen som är en produktspecifik *intersects*-liknande funktion för just SQL Server för att avgöra vilka polygonobjekt som befinner sig inom angiven geografisk utbredning.

**GetMap-anrop:**

*/geoserver/arlanda_test/wms?LAYERS=arlanda_test%3Avy_ru*
*msyta_123_georef&STYLES=&FORMAT=image%2Fpng&SERVIC*
*E=WMS&VERSION=1.1.1&REQUEST=GetMap&SRS=EPSG%3A30*
*11&BBOX=146049,6615087,146196,6615139&WIDTH=929&HE*
*IGHT=330*

**SQL-fråga:**

```
SELECT    geometri.STAsBinary() as geometri
FROM      vy_rumsyta_123_georef
WHERE     geometri.Filter(geometry::STGeomFromText(
          'POLYGON ((146049 6615087, 146049 6615140,
          146197 6615140, 146197 6615087,
          146049 6615087))', 3011)) = 1
```

#### *5.1.2.3 GeoServer*

GeoServer är en kartserver utvecklad med öppen källkod som tillåter användaren att editera och dela geografiska data. GeoServer är anpassad efter OGCs öppna standarder inom webbaserad visualisering av geografisk data. (GeoServer 2009)

### 5.1.3 Spatiala databaser

Spatiala databaser används för att lagra spatiala data för att enkelt och effektivt hantera och editera dem. Det finns flera typer av spatiala databaser, de flesta är s.k. objektrelationella databaser och den tekniska beskrivningen av dem kommer att beskrivas i kapitel 6.4. I kapitel 8 beskrivs två av de vanligaste spatiala databasprodukterna på marknaden.

## 5.2 Alternativ systemarkitektur

För att undvika att använda mellanhänder i systemarkitekturen kan förfrågningar ske direkt mot databasen från klienten. Detta medför att samtliga objekt för en specifik SQL-fråga skickas från databasen till klienten. Klienten i sin tur behöver då antingen transformera samtliga objekt till ett bildformat eller visualisera objekten i sin råa form. Det resulterar i att antingen måste klienten vara mer kraftfull (t.ex. *desktop-GIS*) som på kort tid kan transformera och visualisera objekten i bildformat eller så kommer den totala tidsåtgången bli längre än vid användning av kartserver som mellanhand.

# 6 Lagringsmetoder

I dagens informationssamhälle ökar ständigt mängden data och det är väldigt viktigt att data lagras och administreras effektivt. Det finns två grundläggande lagringsmetoder, filbaserade lagringssystem och databassystem.

Filbaserade lagringssystem hanterar data i filer och hierarkiska katalogstrukturer. Kataloger kan innehålla både filer och andra kataloger. Filernas uppbyggnad och utformning beror helt på vilken mjukvara som ska behandla dem, allt ifrån helt ”rå” binära data till produktspecifika data.

Ett databashanteringssystem består av en eller flera databaser som är kopplade till mjukvaruprogram som används för att kommunicera med databasen. Det finns flera olika databashanteringssystem på marknaden för att underlätta lagring och hantering av data, både kommersiella och system med öppen källkod. De flesta databashanteringssystem följer gemensamma standarder och direktiv för att underlätta interoperabilitet och delning av data.

Oberoende av lagringssystem, filbaserat eller databas, så ska ett fungerande datalagringssystem uppfylla följande krav (baserat på Elmasri & Navathe 2011 s.10):

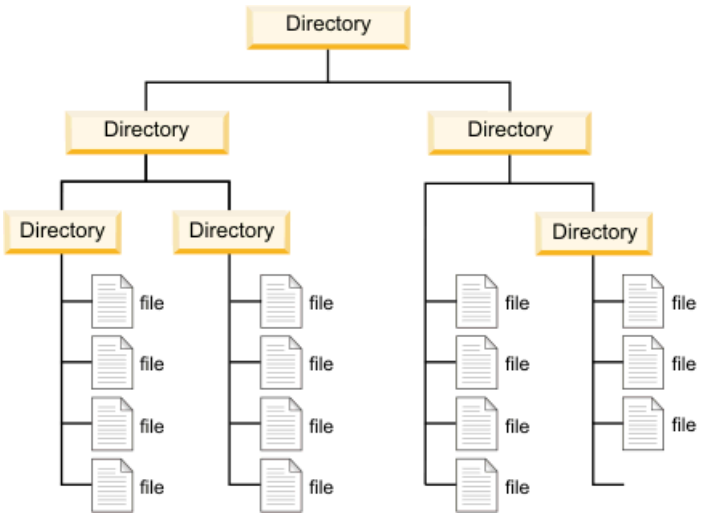
- Metadata – lagrade data ska vara tydligt definierade och innehålla deskriptiva attribut.
- Editera data – möjligt att lägga till, radera och modifiera data.
- Hämta data – möjligt för användare att simultant hämta data.
- Behörighetskontroll – kontrollera användarnas behörighet vid uppdatering och hämtning av data.

## 6.1 Filbaserade lagringssystem

Filbaserade lagringssystem hanterar data i separata filer. Ett kriterium för funktionellt filsystem är att filnamnen är unika. Filnamnet består av filens ”adress” (vilka kataloger filen ligger i). Varje fil har en metadata-fil och en inod-fil kopplat till sig som beskriver filen och dess innehåll (Giampaolo 1999). Metadata-filer innehåller deskriptiva data om filen som t.ex. namn, vem som har skapat filen och vilken data filen innehåller. Inod-filer innehåller information om var, både katalogvis och position i hårddisken, data är lagrade.

För att effektivt hantera filbaserade lagringssystem sorteras och indexeras filerna utifrån ett jämförelsebart attribut i inod-filen (Giampaolo 1999). Indexeringen sker efter det valda attributet i inod-filerna och en referens till var filen och dess inod-fil är lagrad sparas. B-träd och dess varianter (t.ex. $B^+$-träd) samt hashtabeller är de vanligaste indexeringsmetoderna för filbaserade system (Giampaolo 1999).

Den konceptuella strukturen av filbaserade lagringssystem är en hierarkisk lagringsstruktur med kataloger (eng. *directories*) och filer (Giampaolo 1999). Varje katalog kan innehålla både filer och andra kataloger, se Figur 6.1.




**Figur 6.1. Grafisk visualisering över ett filbaserat lagringssystem (IBM 2010).**

För att minimera mängden duplicerade data och samtidigt öka interoperabiliteten kan filer innehålla scenariospecifika attribut (Giampaolo 1999). Attributens utformning och datatyp beror på mjukvaruprogrammet som läser filen och vilket användningsområde det berör. Figur 6.2 är ett exempel av ett filsystem som hanterar personuppgifter. Istället för att behöva ange attributnamnen namn, ålder och kön i varje personfil är de bestämda attribut i mjukvaran. Filens storlek är, för vanliga dataformat (t.ex. ASCII), direkt beroende av antal tecken i filen (Wikipedia 2013c). Användningen av attribut minskar därmed filernas nödvändiga lagringsutrymme.

| |
|---|
| namn Anders Andersson<br>ålder 27<br>kön man |

| |
|---|
| Anders Andersson<br>27<br>man |

**Figur 6.2. Beskrivning av attributanvändning i filbaserade lagringssystem. Den högra figuren använder inte attributnamn.**

## 6.2 Relationsdatabaser

Relationsdatabaser är det mest använda lagringsalternativet för hantering av textuella- och numeriska data. En anledning är det väl etablerade kommunikationsspråket SQL (*Structured Query Language*). Relationsdatabaser är uppbyggda i tabellform. För att uppnå önskad struktur i datalagringen måste minst en kolumn vara en s.k. nyckel (primär eller sekundär). För att säkerställa att databaser är korrekt uppbyggda innehar de fyra viktiga egenskaper (baserat på Brinkhoff & Kresse 2012):

- Tillförlitlighet – databasen ska vara felsäker och tillhandahålla en säker tjänst oavsett om fel i hård- eller mjukvaran inträffar.
- Korrekthet – databasen ska innehålla den data den utger sig för att innehålla.
- Teknisk bevis – databasen ska vara oberoende av vilken hård- och mjukvara som används för tillfället.
- Säkerhet – skriv- och läsrättigheterna ska vara skyddade i databasen.

För att säkerställa effektiv och säker lagring samt för att minimera mängden duplicerade data finns det vissa riktlinjer att följa vid skapandet av en relationsdatabas, så kallade normalformer (eng. *Normalization Form*). Det finns sex normalformer, varav de tre första anses ha hög prioritet. De berör skapandet av de separata tabellerna och anger att: varje cell får endast innehålla ett odelbart värde (1NF), varje tabell ska endast representera en objekttyp (2NF) och alla attribut som inte är nycklar ska vara direkt beroende av primärnyckeln (3NF). De tre sista normalformerna behandlar hur beroenden mellan tabeller ska vara utformade samt hur sammanslagningar (eng. *join*) ska ske. En relationsdatabas måste uppfylla de tidigare normalformerna för att tilldelas en högre normalform. Det räcker alltså inte att endast uppfylla riktlinjerna i 3NF utan även 1NF och 2NF måste vara uppfyllda. (Brinkhoff & Kresse 2012)

En anledning till varför inte relationsdatabaser är tillräckliga vid hantering av spatiala data uppmärksammas redan vid anpassningen efter de första normaliseringsformerna. Ska t.ex. koordinaterna för en väg lagras i en relationsdatabas strider det mot 1NF, att varje cell endast får innehålla ett odelbart värde (inte koordinatpar eller en serie koordinater).

### 6.2.1 SQL

SQL-språket är det mest etablerade standardverktyget för kommunikation mellan användare och relationsdatabaser. Språket utvecklades under 1970-talet av IBM Corporation Inc. och antogs som en standard av ANSI (*American National Standard Institute*) år 1986 och blev året därefter accepterad som en ISO-standard (*International Organization for Standardization*) (Wikipedia 2013a). SQL har utvecklats mycket under tidens gång och stödjer sedan version SQL:1999 (även kallad SQL/MM) OGCs standard *Simple Feature* (se avsnitt 6.5.1) och kan hantera bl.a. spatiala och multimediala data.

SQL är uppbyggt av de fyra språken: *Storage Definition Language* (SDL), *View Definition Language* (VDL), *Data Definition Language*(DDL), och *Data Manipulation Language* (DML) (Elmasri & Navathe 2011). SDL och VDL är kommunikationsverktyg och används internt av mjukvaruprogrammen för att kommunicera med datalagringen i hårdvaran (Elmasri & Navathe 2011). DDL används för att skapa, radera och modifiera definitionerna av tabellerna i databasen så att korrekt och beskrivande attribut lagras enligt definitionen i metadata. Vid skapandet av tabeller anges, med hjälp av DDL, t.ex. vilken datatyp ett attribut får innehålla (t.ex. *char*, *int* eller *double*) samt vilket attribut som är primärnyckel (eng. *Primary Key*) (Chawla & Shekhar 2003). När utformningen av databasen är klar används DML för att fylla, radera eller editera data i databasen samt fråga efter data (Chawla & Shekhar 2003).

Ett viktigt användningsområde inom SQL är frågehantering. Frågehantering är till stor del baserat på relationsalgebra och utvecklat för att göra kommunikationen med databaser lättanvänt och intuitivt (Wikipedia 2013a). SQL-frågor genereras med hjälp av DML och består av sammansättningar av de obligatoriska klausulerna *SELECT* och *FROM*. Frågorna kan kombineras med flera valfria DML-klausulerna, t.ex. *WHERE* och *GROUP BY* för att generera utförligare frågor. SQL kan hantera logiska operatorer (=, <, >, *AND* och *NOT*) vilket möjliggör att anändaren kan ställa avancerade frågor direkt till databasen och beräkningar kan utföras.(Elmasri & Navathe 2011)

SQL har blivit det mest använda verktyget för att hantera relationsdatabaser eftersom det är så kraftfullt men ändå enkelt och intuitivt att använda. Däremot visar det sig vara bristfälligt för att hantera spatiala data, en anledning är att spatiala data kräver speciella datatyper för att lagra t.ex. koordinater och geometriska former. För att möjliggöra hantering av spatiala data med SQL och relationsdatabaser har flera olika tillägg utvecklats för SQL-baserade databaser. Många av dessa tillägg har samlat inspiration från objektorienterade programmeringsspråk som t.ex. Java och C++ och de tillåter användaren t.ex. att definiera och skapa egna datatyper (Chawla & Shekhar 2003). Många av databassystemen som kan hantera spatiala data kallas därför för objektrelationella databaser.

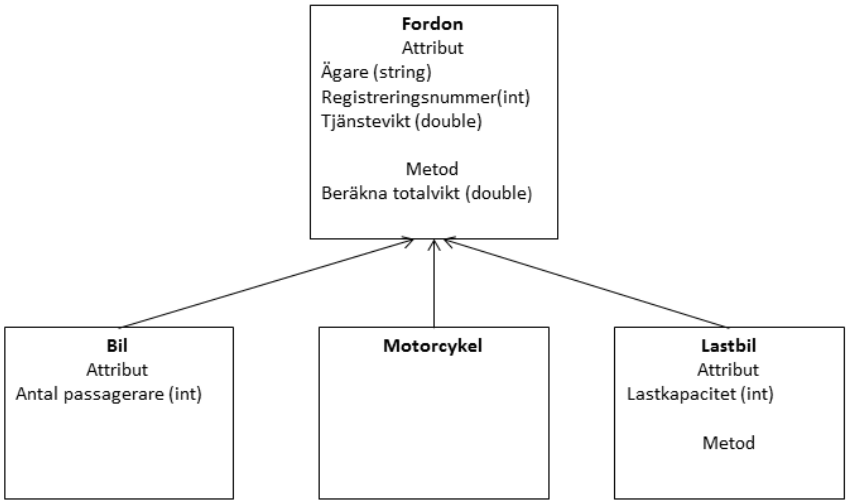
#### *6.2.1.1 Vyer*

Vyer (eng. *Views*) är virtuella tabeller i relationsdatabaser (Microsoft 2013b). Virtuella innebär att tabellerna inte lagras fysiskt i databasen utan de är resultatet av ett SQL-uttryck. SQL-uttryck och frågor kan ställas direkt mot vyer och vyer används ofta för att förenkla långa och komplicerade uttryck. Istället för att skriva komplicerade uttryck bestående av flera sammanslagningar (eng. *joins*) av tabeller med flera olika *WHERE*-klausuler kan situationen förenklas genom att skapa en vy och ställa en följdfråga till den.

## 6.3 Objektorienterade databaser

Objektorienterade databaser hanterar data i objektform. Varje objekt har en unik identifierare (*Object Identifier*) och objekten kategoriseras i klasser (Cattell & Barry 2000). Klasserna är uppbyggda av attribut och metoder. Objektorienterade databaser integreras enkelt med system som är utvecklade med objektorienterade programmeringsspråk som t.ex. Java och C++ (Elmasri & Navathe 2011).

Ett grundläggande och viktigt koncept inom objektorienterade databassystem, som har ärvts från objektorienterad programmering, är arv- och hierarkistrukturer. Arvstruktur, se Figur 6.3, innebär att subklasserna ärver attribut och metoder från superklasserna. Därmed behöver inte attributen och metoderna definieras för varje klass utan bara en gång för superklassen (Elmasri & Navathe 2011). Arv har bara en riktning, t.ex. alla motorcyklar är ett fordon, men alla fordon är inte motorcyklar.




**Figur 6.3. En förenklad arvstruktur. Subklasserna Bil, Motorcykel och Lastbil ärver attributen Ägare, Registreringsnummer och Tjänstevikt och behöver inte definieras varje gång någon av subklasserna skapas.**

Korrekt uppbyggt objektorienterat (klass) system är dynamiskt och förenklar utvecklingen och påbyggnaden av systemet. Det är t.ex. enkelt att i systemet i Figur 6.3 lägga till subklassen Buss eller radera subklassen Motorcykel utan att det påverkar någon av subklasserna eller superklassen.

En stor fördel med objektorienterade databaser är att de kan hantera andra datatyper än de ”primitiva” (t.ex. *int*, *double* och *char*) och tillåter lagring av datatyper som t.ex. mängder och listor (Brinkhoff & Kresse 2012). I en objektorienterad databas kan t.ex. en vägs koordinater lagras som en lista innehållandes punkter (koordinater). Objektorienterade system tillåter även användaren att skapa egendefinierade objektklasser, med tillhörande attribut och metoder, så kallade abstrakta datatyper (Elmasri & Navathe 2011). Abstrakta datatyper kan användas både som attribut- eller objektklass. D.v.s. användaren kan skapa ett bilobjekt med attributen ägare m.m. eller skapa ett ägarobjekt med ett fordonsattribut (som kan sättas till bil). Abstrakta datatyper medför att databasen helt kan skräddarsys efter vilka behov det ställs på databasen. Det resulterar dock i att det är svårt att standardisera klasser och metoder eftersom användare har olika behov.

Object Management Group (OMG) är en internationell, ideell organisation grundad år 1989 och är en utbrytning av Object Data Management Group, som avvecklades år 2001 (ODBMS 2005). Medlemmarna består av regeringsorgan, akademiska institutioner, återförsäljare och användare av objektorienterade databaser (OMG 2013). Organisationen arbetar för att skapa standarder vid upprättandet av objektorienterade system. OMG har bland annat utvecklat två kommunikationsstandarder: *Object Definition Language* (ODL) och *Object Query Language* (OQL) (Elmasri & Navathe 2011). Standarderna är i sin tur utvecklade efter den övergripande ODMG *Object Model*, en standard framtagen av ODMG och är en vedertagen standard för alla typer av objektorienterade system (Cattell & Barry 2000).

### 6.3.1 Object Definition Language (ODL)

ODL är en programmeringsspråkoberoende designstandard som används för att skapa specifikationer för objektklasser (Cattell & Barry 2000). Standarden ska inte förväxlas med ett programmeringsspråk utan den används för att skapa objektklasserstrukturer med arv- och hierarkistrukturer samt upprätta databasdesigner enligt ODMG *Object Model*.

ODL består av *Data Definition Language* (DDL) och *Data Management Language* (DML) (se 6.2.1) fast användaren kan definiera egna datatyper och abstrakta klasser i sann objektorienterad programmeringsanda (Cattell & Barry 2000).

### 6.3.2 Object Query Language (OQL)

OQL är ett standardiserat verktyg för kommunikation med objektorienterade databassystem. OQL är utvecklat efter ODMG *Object Model*-standard för att enkelt integreras med olika objektorienterade programmeringsspråk t.ex. C++ och Java (Cattell & Barry 2000). Utvecklingen av OQL har inspirerats av relationsdatabassystemens kommunikationsspråk SQL och har därför en liknande syntax och funktion (Elmasri & Navathe 2011).

## 6.4 Objektrelationella databaser

Objektrelationella databaser är som namnet antyder en kombination av relationsdatabaser och objektorienterade databaser. Databasen är uppbyggd av tabeller som i relationsdataser men istället för att enbart hantera de strängt begränsade datatyperna från relationsdatabaser kan objektrelationella databaser även hantera abstrakta- och egendefinierade datatyper som objektorienterade databaser.

SQL-språkets utveckling har gått hand i hand med utvecklingen av objektrelationella databaser. SQL:1999 var den första SQL standarden som innehöll objektrelationella definitioner (Brinkhoff & Kresse 2012). Det är numera möjligt att använda SQL för att kommunicera med objektrelationella databaser. Objektrelationella databassystem är dock olika utvecklade men de har ändå SQL som gemensamt kommunikationsverktyg. Utvecklingen av objektrelationella databaser har resulterat i att de flesta relationsdatabaser har utvecklats till objektrelationella databaser. Den stora fördelen med objektrelationella är användarens möjlighet att skapa abstrakta- och egendefinierade datatyper.

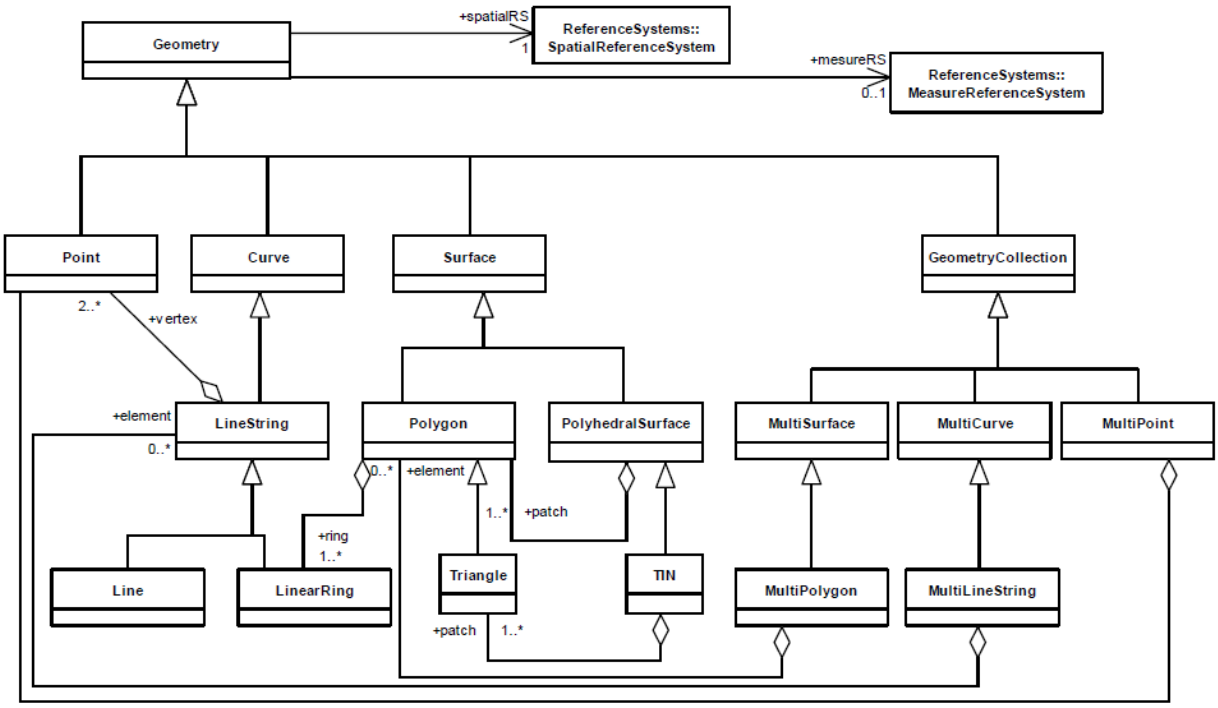
### 6.4.1 Spatiala databaser

Spatiala databassystem är en typ av objektrelationella databaser utvecklade specifikt för att hantera geografiska data. Detta har blivit möjligt genom integreringen av OGCs standard *Simple Feature*. Det finns flera tillvägagångssätt för att skapa funktionella och effektiva system. Att det finns flera olika system visar att det är svårt att upprätta en lösning som är effektivt för alla potentiella användningsområden. Många spatiala databassystem på marknaden är utvecklade av gemensamma standarder och direktiv.

OGC är en internationell organisation, bestående av regeringsorgan, företag och akademiska institut, som utvecklar standarder för geometriska datatyper med öppen källkod, t.ex. *Simple Feature* (OGC 2013).

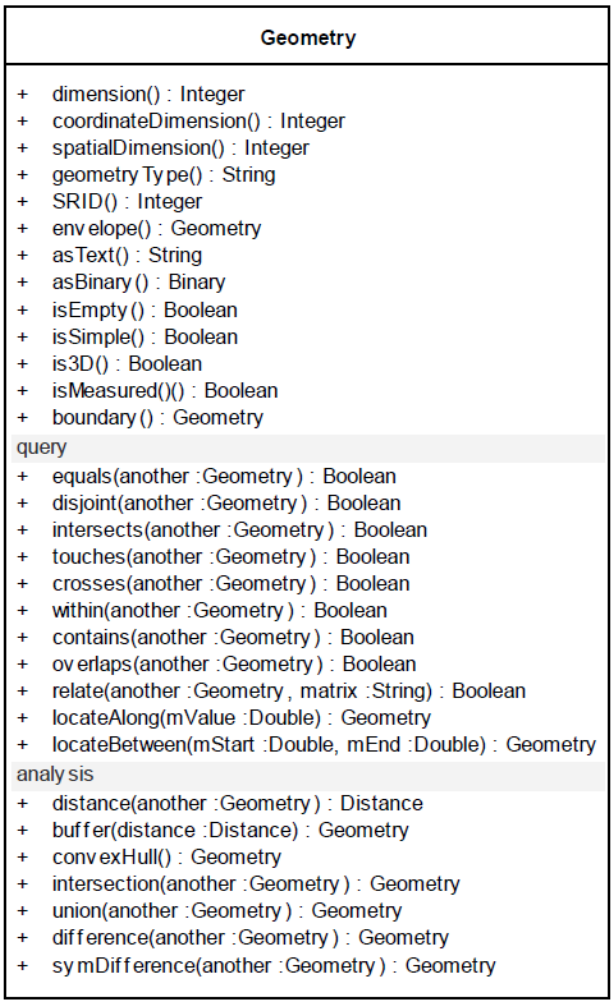
### *6.4.1.1 Simple Feature*

*Simple Feature* är en vedertagen ISO-standard (ISO 19125) som möjliggör hantering av två-dimensionella geografiska objekt (OGC 2011). Figur 6.4 visar den hierarkiska strukturen av *Simple Feature*-standarden i form av ett UML-diagram.




**Figur 6.4. Grafisk visualisering över *Simple Features* hierarkiska struktur (OGC 2011).**

Standarden innehåller även specifikationer av de geometriska- och topologiska operationer som kan utföras på de spatiala. Operationerna kan användas i SQL-uttryck och -frågor eftersom *Simple Feature* är implementerat inom SQL sedan version SQL:1999 (Brinkhoff & Kresse 2012). Figur 6.5 visar superklassen *Geometry* och vilka operationer som är implementerade för denna klass.




**Figur 6.5. *Simple Features* superklass *Geometry* och vilka geometriska- och topologiska operationer som är implementerade (OGC 2011).**

## 6.5 NoSQL-databaser

NoSQL-databaser är en utveckling av klassiska relationsdatabaser men är anpassade för väldigt stora datamängder och snabb hämtning av data (NoSQL 2013). Som namnet antyder använder inte NoSQL-databaser det standardiserade kommunikationsspråket SQL utan tillämpar olika objektorienterade användargränssnitt (eng. API (*Application Programming Interface)*), beroende på vilken NoSQL-databas som används.

Fördelen med NoSQL-databaser framför ”klassiska” objektrelationella databaser är att de är väldigt flexibla vid hanteringa av förändringar i databasens storlek. Det är enkelt vid behov att förstora lagringsutrymmet och datorkraften vid användning av NoSQL-databaser. NoSQL-databaser tillämpar horisontell uppskalning (eng. *Horizontal Scaling*) istället för vertikal uppskalning (eng. *Vertical Scaling*) som är vanligt för objektrelationella databaser (Mongodb 2013a). Horisontell uppskalning lägger till fler servrar parallellt istället för att förstora den befintliga servern.

NoSQL-databaser arbetar inte i strikta och statiska tabeller där alla objekt måste vara utformade på samma vis. NoSQL tillåter användaren att lägga till ”vilken typ” av data som helst. NoSQL-databaser kan vara utvecklade från väldigt enkla nyckel-värde system (eng. *Key-value*) till mer avancerade s.k. dokument databaser. I nyckel-värde system har varje element en nyckel och ett värde kopplat till nyckeln. Dokument databaser kan lagra nyckel-värde par eller- listor. (MongoDB 2013b)

MarkLogic är ett exempel av en NoSQL-databas som tillämpar dokumentstruktur. MarkLogic kan hantera formaten GML, KML och GeoRSS samt flera olika varianter av de vanliga geografiska datatyperna (punkter, linjer och polygoner). Dessa datatyper kan användas för att utföra olika spatiala beräkningar som t.ex. (kortaste) avstånd, ytberäkningar samt topologiska beräkningar som skärningar (eng. *Intersects*) och innehåller (eng. *Contains*). (Marklogic 2012)

# 7 Indexering

Effektiv lagring och hantering av data kräver att datamängden är sorterad efter något jämförelsebart attribut, t.ex. textuellt eller numeriskt. Datamängder innehåller ofta flera attribut men sorteringen kan bara ske enligt ett attribut åt gången. För att upprätta effektiva indexeringsstrukturer krävs det att datamängden kan sorteras och jämföras. Det är i indexeringsstrukturen sökningen efter aktuellt objekt sker och kopplas sedan ihop med resterande data för att generera sökt information om objektet (Elmasri & Navathe 2011).

## 7.1 Linjär- och binärsökning

Tidsåtgången för genomsökningar av datamängder uttrycks ofta som tidskomplexitet: *O(f(n))*, där *f(n)* är en funktion beroende av genomsökningsalgoritm och *n* är antalet element i mängden (Elmasri & Navathe 2011). Tidskomplexiteten beskriver tidsåtgången och uttrycks antingen som ”värsta fall”- eller ”medel fall” scenario.

Sker sökningen i en osorterad datamängd utförs en s.k. linjärsökning, varje element i datamängden behöver i värsta fall traverseras. Är datamängden sorterad kan s.k. binärsökning utföras.

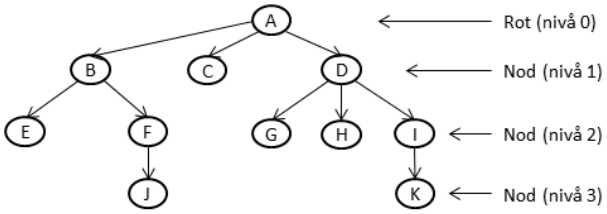
Utförs en linjärsökning traverserar (i värsta fall) varje element i datamängden. Söktiden är linjärt beroende av hur många element datamängden innehåller och tidskomplexiteten beräknas som *O(n)*, där *n* är antalet element i datamängden (Elmasri & Navathe 2011). Exempelvis fördubblas söktiden om antalet element i mängden fördubblas vid linjärsökning.

Binärsökning använder att datamängden är sorterad och tillämpar en s.k. söndra och härska-teknik för att snabbt hitta rätt element. Steg ett är att jämföra datamängdens mellersta element med det sökta samt dela mängden i två delar. Är det sökta värdet större än det mellersta sker en ny likadan sökning i den övre mängden, annars den nedre. I steg två upprepas rutinen från steg ett tills rätt element är hittat. Tidskomplexiteten vid binärsökning beräknas som *O(log(n))* där *n* är antalet element i datamängden (Elmasri & Navathe 2011).

## 7.2 Sökträd

Eftersom indexering ställer höga krav på dynamisk struktur och möjlighet till snabb genomsökning är det inte lämpligt att lagra indexeringsdata i list- eller vektorsystem. Även om listorna eller vektorerna är sorterade är tidskomplexiteten hög vid insättning (på rätt plats) och sökning (Elmasri & Navathe 2011). Den vanligaste lagringsstrukturen av digitala data som istället används är sökträd.

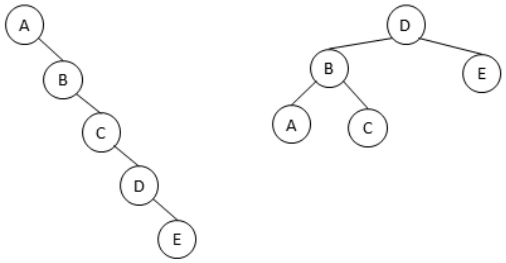
Med hjälp av trädstruktur är det enkelt att hålla reda på var på hårddisken data är lagrade, vilket medför att det går snabbt att hämta eller editera dem. Det finns flera trädmodeller och de har olika för- och nackdelar beroende på datatyp och mängd av data som ska lagras. Alla trädmodeller består av noder, varav noden på nivå noll kallas rot. Noder utan någon nod under sig kallas löv, se Figur 7.1. Oavsett trädstruktur ska alla sökträd vara sorterade för att underlätta administrering av data (Elmasri & Navathe 2011).




**Figur 7.1. Grafisk beskrivning av ett generellt obalanserat sökträd. Noderna C, E, G, H, J och K kallas löv (baserat på Elmasri & Navathe 2011 s.647).**

Sökträdets modell påverkar uppbyggnaden och utseendet på trädet. Beroende på vilken datatyp och mängd som ska lagras kan det vara fördelaktigt med olika trädmodeller. Varje nod innehåller det sorterade attributet samt information om var objektets resterande data är lagrad.

Ett sökträd kallas balanserat om lövens maximala nivåskillnad är mindre eller lika med ett (Elmasri & Navathe 2011). Fördelen med balanserade sökträd är att färre noder behöver sökas igenom och därmed minskar tidskomplexiteten i både medel- och värsta fallet. Jämför de två sökträden i Figur 7.2, att söka igenom det vänstra sökträdet tar i de flesta scenarion längre tid (fler noder behövs traverseras).




**Figur 7.2. Visualisering av två sorterade trädmodeller. Det vänstra är obalanserat och det högra är balaserat.**

Många trädalgoritmer är självbalanserade och sorteras automatiskt. Detta medför att insättning av element har relativt hög tidskomplexitet. Men balanserade och sorterade träd har mycket lägre tidskomplexitet vid hämtning av data, så det är generellt sätt att föredra.

### 7.2.1 B-träd

B-träd är den vanligaste trädalgoritmen vid hantering av numeriska- och textuella data. Dess popularitet beror bl.a. på att den uppfyller följande krav (baserat på Elmasri & Navathe 2011 s.649):

1. Trädet är alltid sorterat och balanserat.
2. Noderna är alltid sorterade.
3. Fyllandsgraden kontrolleras vid insättning av element.
4. Vid radering av element bibehåller alltid noderna minst 50 % fyllnadsgrad.

Punkt 1 och 2 är standard för samtliga trädmodeller. Punkt 3 och 4 är dock de som har bidragit till B-trädets popularitet. Fyllnadsgraden (trädets och nodernas) påverkar trädstrukturens totala förbrukning av lagringsutrymme (även tomma noder allokerar plats i hårddisken). Samtidigt eftersträvas inte 100-procentig fyllnadsgrad eftersom det medför problem vid insättning av nya objekt. Genomsökningstiden påverkas av fyllnadsgraden (trädets och nodernas) och det är en matematisk balansgång att minimera genomsökningstiden beroende av trädets och nodernas fyllnadsgrad.

$B^+$-träd är en utvecklad version av B-träd som har blivit väldigt populär. Till skillnad från B-träd så lagrar $B^+$-träd endast data i löven vilket resulterar i att indexeringsstrukturen uppnår högre kapacitet (Elmasri & Navathe 2011).

## 7.3 Indexeringsnivå

Indexeringsstrukturer delas in i enkel- och multipel indexeringsnivå (eng. *Single-Level* och *Multilevel*). Observera att indexeringsnivå inte är samma sak som trädnivå även om trädstrukturer ofta används för att beskriva indexeringsstrukturer.

### 7.3.1 Enkel indexeringsnivå

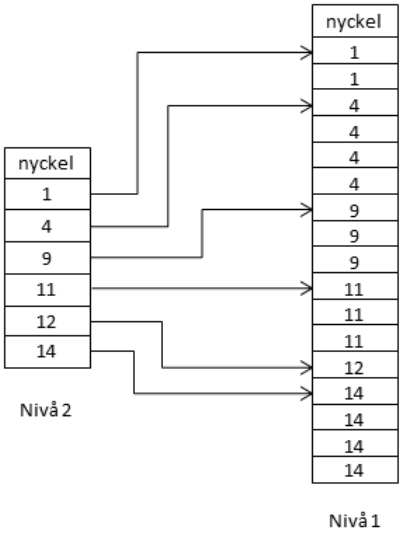
Om en datamängd indexeras utifrån endast ett attribut kallas det för enkel indexering (Elmasri & Navathe 2011). Det är den vanligaste indexeringsnivåstrukturen och binärsökning kan utföras på indexeringsmängden.

Indexeringsstrukturen kan användas för att fysiskt sortera datamängden i hårddisken. Skapas en enkel indexeringsnivå med hjälp av ett sorterbart nyckelattribut kallas det för ett primärindex (eng. *Primary Index*); om attributet är sorterbart men inte är en nyckel (tillåter duplicerade värden) kallas det för klusterindexering (eng. *Clustering Index*) (Elmasri & Navathe 2011). Den fysiska sorteringen i hårddisken snabbar upp hämtningen av data genom att minimera risken att datorn läser och skriver från samma diskområde (eng. *disc space*) samtidigt. Det medför att en tabell enbart kan ha ett primär- eller klusterindex.

Det är möjligt att kombinera primär- eller klusterindexeringar med s.k. sekundära indexeringar. Sekundära indexeringar indexerar data utan att fysiskt ändra ordningen i hårddisken och sekundär indexering kan därmed utföras på icke sorterbara attribut (Elmasri & Navathe 2011).

### 7.3.2 Multipel indexeringsnivå

Vid användning av enkel indexeringsnivå reduceras tidskomplexiten till *O(log n)*. Detta anses vara otillräckligt och skapar långa svarstider för stora datamängder. En lösning som tillämpas är en s.k. multipel indexeringsnivå som indexerar en indexfil (Elmasri & Navathe 2011). Multipel indexeringsnivå ”ser” indexeringsfilen som en sorterad fil som kan indexeras, se Figur 7.3. Varje ny indexeringsfil kan indexeras på nytt och processen kan upprepas obegränsat antal gånger. Eftersom varje ny indexeringsnivå genererar en ny indexeringsfil kan indexeringar på olika nivåer alla vara primär- eller klusterindexeringar även om indexeringar på lägre nivåer är det (Elmasri & Navathe 2011).




**Figur 7.3. Grafisk visualisering av multipel indexeringsstruktur med två nivåer. Nivå 1 (basen) består av en klusterindexering. (baserat på Elmasri & Navathe 2011)**

Indexeringsstrukturer som bygger på multipel indexeringsnivå är mer tidkrävande att skapa än enkel indexeringsnivå. För att undvika att skapa ny indexeringsfil vid varje insättning eller radering av objekt i datamängden fylls inte indexeringsfilen helt vid skapandet. Indexeringsfiler med multipla indexnivåer och med ”tomma platser” kallas för dynamisk multipel indexering (eng. *Dynamic Multilevel Index*) (Elmasri & Navathe 2011).

Tidskomplexiteten för multipel indexeringsnivå är beroende av antalet indexeringsnivåer (indexeringens utbredningsproportion) och blir approximativt $O(\log_{ut} n)$, där *ut* är indexeringens utbredningsproportion och *n* är antalet element (Elmasri & Navathe 2011). Ofta är utbredningsproportionen mycket större än två så multipel indexeringsnivå är i regel mycket snabbare än enkel.

Vid skapandet av indexeringar är det ofta en svår avvägning mellan hur många nivåer indexeringen ska bestå av och hur många tomma platser den ska innehålla. Tomma platser i indexeringsfilen ockuperar lagringsutrymme i hårddisken och ökar indirekt trädets antal nivåer och därmed indexeringsstrukturens totala lagringsbehov.

## 7.4 Spatial indexering

Noderna i spatiala indexeringsstrukturer består ofta av listor. Genomsökningen av spatiala indexeringsstrukturer delas därför in i två steg: hitta rätt nod och hitta rätt objekt i noden. Hitta rätt nod är en första gallring och är den huvudsakliga anledningen till spatiala indexeringar. Genomsökningen för att hitta rätt nod sker med binärsökning och är snabb. När rätt nod är funnen börjar den tidskrävande sökningen efter korrekt objekt. Sökningen efter korrekt objekt i en lista sker med linjärsökning vilket innebär att varje objekt behöver kontrolleras.

Spatiala data är ofta mer problematiskt att sortera än textuella- eller numeriska data. Grundproblemet är att spatiala data inte har någon naturlig sorteringsordning. Val av

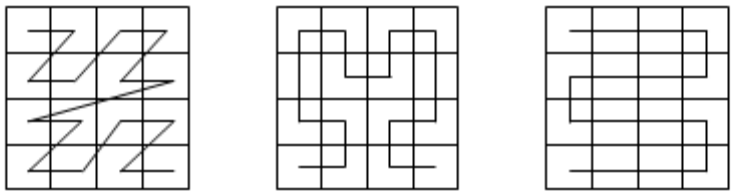
Indexeringsalternativ påverkas även av vilken typ av spatiala data som lagras och vilka funktioner och operationer som ska utföras på datamängden.

Följande grundläggande kriterier behöver uppfyllas för att indexeringsmetoden ska anses vara funktionsduglig och effektiv, oavsett vilka typer av spatiala data databasen innehåller (baserad på Brinkhoff & Kresse 2012 s.88):

1. Punkter, ytor- och kurvapproximationer ska vara organiserade.
2. Grundläggande spatiala frågeställningar måste stödjas.
3. Rumsligt närliggande data ska med hög sannolikhet sorteras nära varandra i indexeringsstrukturen.
4. Dynamisk insättning, modifiering och radering av objekt ska stödjas.
5. Rimligt utnyttjande av lagringsutrymmet ska garanteras.
6. Indexstrukturen ska vara robust, lagring av olikformad data ska inte signifikant påverka svarstiden.

Punkt 1,2,4 och 5 kan ses som självklara för alla typer av databassystem och är inte unikt för hantering av spatiala data. Objekten som hanteras i databassystemet ska givetvis vara organiserade och grundläggande (spatiala) SQL-frågor ska kunna utföras på dem. Att objekten ska kunna modifieras utan att lagringsstrukturen signifikant försämras och påverkar lagringsutrymmet negativt råder det inga tvivel om även vid hantering av icke-spatiala data.

Punkt 3 är däremot ett för spatiala data unikt kriterium och kan vid första anblick låta både självklart och enkelt. Men vid närmare eftertanke förstår man att det snabbt blir ganska komplicerat att avgöra vilka objekt som befinner sig nära varandra. Ett sätt att skapa denna närhetsegenskap är att använda en rumsligt fördelad kurva (eng. *Space-Filling Curve*), se Figur 7.4. Rumsligt fördelade kurvor anger ordningen av hur ett begränsat område ska traverseras (Rigaux m.fl. 2002). Traveseringen avgör insättningsordningen av data i indexeringsstrukturen. Objekt inom samma cell kan t.ex. lagras i samma nod i indexeringsstrukturen.



**Figur 7.1. Visualisering av tre olika algoritmer (f.v. Z-kurva, Hilbert-kurva och Horisontal-kurva) för att skapa tre rumsligt fördelade kurvor (baserat på Rigaux m.fl. 2002).**

Även punkt 6 ställer höga krav på lagrings- och indexeringsstrukturen. Det är t.ex. stor skillnad på ett noll-dimensionellt punkobjekt och ett två dimensionellt polygonobjekt, jämfört med numeriska- eller textuella data som ofta lagrar likformade objekt i databasen.

För att förenkla indexeringen brukar approximationer av de geometriska figurer som ska indexeras skapas. Den vanligaste approximationsmetoden är att beräkna figurens minsta rektangulära geometriska utbredning (eng. *Minimum Bounding Box*) (Rigaux m.fl. 2002).

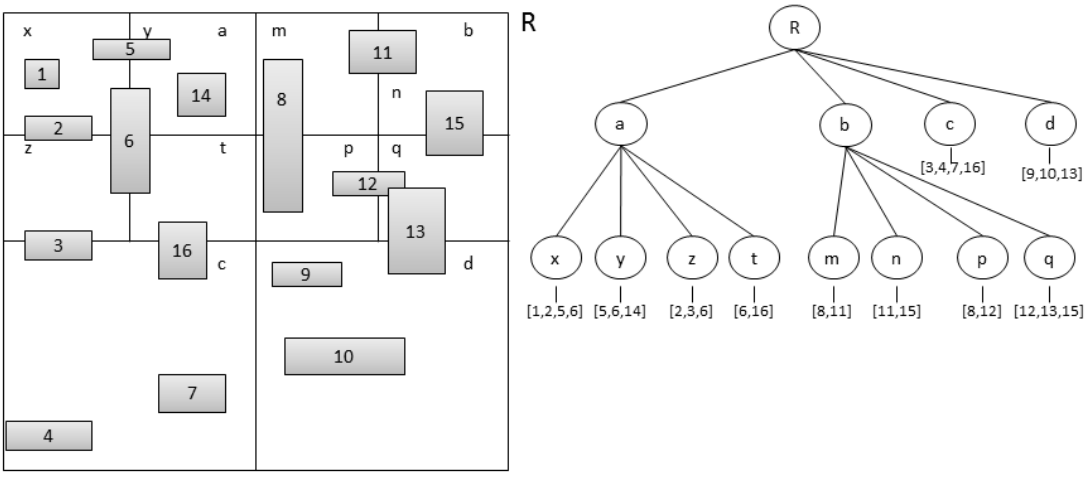
De flesta spatiala indexeringsalgoritmer kan kategoriseras som objektbaserade (eng. *Data-Driven*) eller rumsligt baserade (eng. *Space-Driven*). Objektbaserade indexeringsstrukturer är alltid balanserade och påverkas av hur objekten är fördelade i rummet (Rigaux m.fl. 2002). Rumsligt baserade strukturer påverkas inte av objektens fördelning i rummet utan indexeringen konstruerar cellerna rekursivt enbart efter en förbestämd storleksfördelning av rummet, vilket medför att indexeringen inte sällan blir obalanserad (Aref & Ilyas 2001).

### 7.4.1 Rumsligt baserad indexering

Även om rumsligt baserade indexeringsmetoder genererar obalanserade indexerings-strukturer används de frekvent. Den främsta anledningen är dess ofta enkla implementering och det är enkelt att i förväg uppskatta hur stort lagringsutrymme som krävs för indexeringsstrukturen (Rigaux m.fl. 2002). Är objekten jämt fördelade i rummet är ofta rumsligt baserade indexeringsstrukturer väl fungerande. Quad-träd är en av många rumsligt baserade indexeringsstrukturer.

#### *7.4.1.1 Quad-träd*

Quad-träd är en dynamisk rumsligt baserad indexeringsalgoritm (Rigaux m.fl. 2002). Den är dynamisk i den meningen att algoritmen rekursivt skapar nya indexeringsceller beroende av antal objekt i cellen. Överstiger antalet objekt i cellen antalet platser i nodernas listor (eng. *array*) sönderdelas cellen i fyra mindre celler och antalet objekt kontrolleras i de nybildade cellerna, se Figur 7.5.




**Figur 7.5. Den vänstra figuren beskriver geometriska objekts minsta utbredning (1-16) fördelade i olika celler (a-z) beroende på hur många placeras inom varje cell. Den högra figuren visualiserar indexeringsstrukturens trädmodell. (Baserat på Rigaux m.fl. 2002 s.221)**

Objekt i Quad-träd kan tillhöra flera celler, se t.ex. objekt 6 i Figur 7.5. Det beror på att cellernas utformning inte påverkas av objektens placering. Det medför att duplicerade data lagras i indexeringsstrukturen.

### 7.4.2 Objektbaserad indexering

Objektbaserade indexeringsalgoritmer genererar ofta fullständigt balanserade indexeringsstrukturer, d.v.s. löven i trädmodellen är alla på samma nivå. Objektbaserade indexeringar anpassar cellernas storlek och antal efter objektens fördelning i rummet (Rigaux m.fl. 2002). Objektbaserade indexeringar är ofta mer komplicerade att implementera och mer tidskrävande att generera än rumsligt baserade. Är objekten ojämnt fördelade i rummet är det ofta fördelaktigt med objektbaserad indexering framför rumsligt baserad.

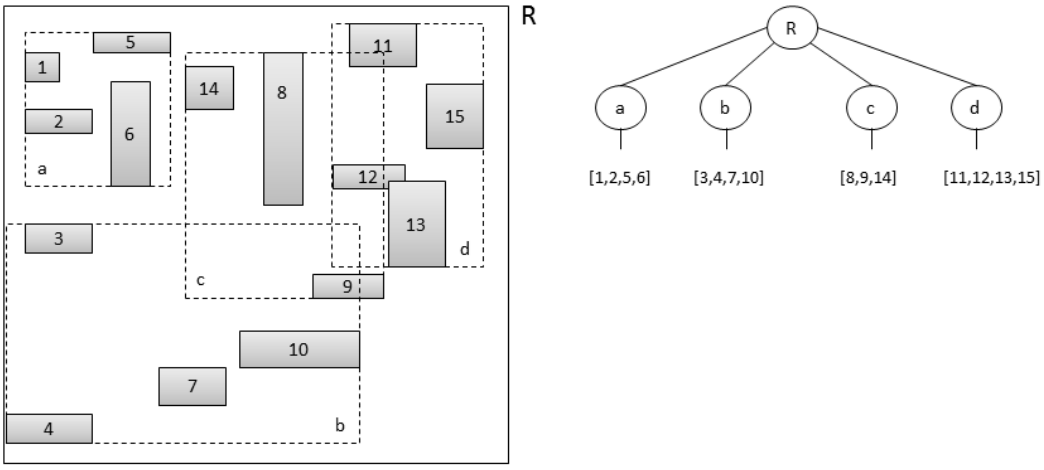
#### *7.4.2.1 Generalized Search Tree (GiST)*

GiST är ett ramverk innehållande flera objektsbaserade och fullständigt balanserade trädstrukturer, t.ex. B-träd, $B^+$-träd, R-träd och R*-träd (Hellerstein m.fl. 1995). GiST är utvecklad så att användaren själv kan välja vilken indexeringsmetod som ska tillämpas på datamängden. Det medför att användaren kan skräddarsy indexeringsmetoden efter vilka behov som ställs i olika scenarier (Hellerstein m.fl. 1995).

#### *7.4.2.2 R-träd*

R-träd är en objektbaserad indexeringsstruktur som genererar fullständigt balanserade träd (Rigaux m.fl. 2002). Indexeringsmetoden genererar storleksoberoende indexeringsceller som helt innesluter ett visst antal objekts minsta rektangulära geografiska utbredning (eng. *Minimum Bounding Box*). Indexeringscellerna genereras utifrån objektens placering i rummet. Indexeringsceller skapas kring de objekt som befinner sig längst ifrån varandra först (Rigaux m.fl. 2002). I Figur 7.6 skapas t.ex. indexeringsceller för objekt 1 och 10 först.

Trädets löv innehåller alla data och består av listor (eng. *arrays*). Listorna innehåller information om vilka objekt som tillhör vilka celler och var i hårddisken objekten är lagrade. Eftersom listorna bara innehåller objekt som är helt inneslutna av indexeringsblocken lagras inte duplicerad data, jämför med Quad-träd (Rigaux m.fl. 2002).




**Figur 7.6. Den vänstra figuren beskriver de geometriska objektens minsta rektangulära utbredning (1-15) fördelade i fyra celler (a-d). Figuren till höger är en grafisk illustration över indexeringsstrukturens trädmodell. (Baserat på Rigaux m.fl. 2002 s.239)**

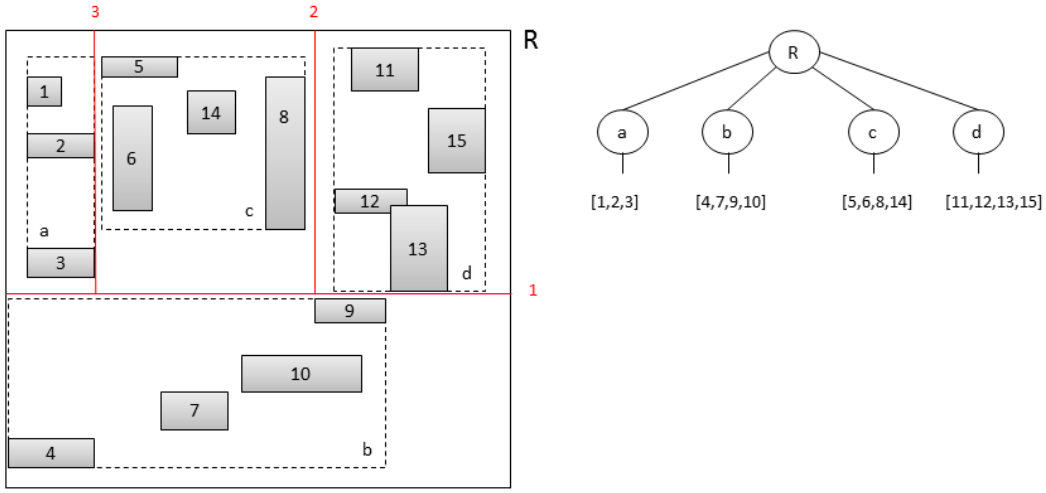
Storleken av, och mängden element i indexeringscellerna (a-d i Figur 7.6) påverkas av antalet objekt och deras fördelning i rummet. Mängden element (*M*) i en lista påverkas av lagringsutrymmets kapacitet (eng. *Disc Page Capacity*) (*P*) och antalet objekt i rummet (*E*) enligt förhållandet *M = P/E* (Rigaux m.fl. 2002 s.239). R-trädets prestanda är proportionellt mot datamängdens minsta geografiska utbredning (eng. *Minimum Bounding Box*), ju större utbredning desto sämre prestanda (Oracle 2013b).

En svaghet med R-trädet är att algoritmen för insättning av element är känslig för dålig insättningsordning. Anledningen är att beräkningen av indexeringscellernas utformning är tidskrävande och R-trädet inte minimerar cellernas överlappning. Sökningen efter korrekt objekt sker i varje indexeringscell som påverkas, är det många celler som överlappar sker sökningen i många subträd. Det har utvecklats flera varianter av R-trädet för hantering av överlappningsproblemet bl.a. R*-träd. (Rigaux m.fl. 2002)

### *7.4.2.3 R*-träd*

R*-träd består av samma struktur som det ursprungliga R-trädet och är också alltid fullständigt balanserat (Rigaux m.fl. 2002). R*-träd tar hänsyn till indexeringscellernas utformning och försöker att minimera överlappningen genom att i första hand dela upp indexeringsblocken i x- och y-led (Rigaux m.fl. 2002). Observera att överlappning är fortfarande tillåtet men mängden är minimerad.

Figur 7.7 beskriver ett R*-träd över samma datamängd som i Figur 7.6. R*-trädet undersöker om det går att dela upp objekten i rummet utan överlapp och börjar med linje 1. Antalet element är för stort i den övre delen (>4) och delas upp på nytt (linje 2 och 3) tills tillåtet antal element är godkänt. R*-trädet har lyckats att indexera mängden utan någon överlappning mellan indexeringscellerna.




**Figur 7.7. R*-trädets indexeringsstruktur lyckas att indexera objekten utan någon överlappning, jämför med Figur 7.6 (baserat på Rigaux m.fl. s.252).**

### 7.4.3 Utvärdering av spatial indexering

För att objektivt och rättvisade jämföra spatiala indexeringsstrukturers prestanda och funktionalitet utvärderas de vanligtvis efter tre faktorer: korrekthet, effektivitet och tidskomplexitet.

Korrekthet mäter hur ofta en spatial fråga returnerar korrekt svar. Vanliga frågor för bedömning av korrekthet är avståndsfrågor. En fråga kan t.ex. vara ”vilket objekt är närmast objekt 9 i Figur 7.7”. Korrekt svar är objekt 13, men eftersom indexeringsstrukturen har sorterat objekt 9 och objekt 13 i olika indexeringsceller är risken stor att avståndet mellan objekt 9 och 13 inte beräknas och svaret blir felaktigen objekt 10. Anledningen är att indexeringen används som ett första filter för att minska antalet utförda avstånds-beräkningar. Indexeringen använder ofta någon av de rumsfördelande kurvorna som presenteras i Figur 7.1 för att avgöra objektens närhet till varandra och därmed placering i indexeringsstrukturen. Det är förstås bara aktuellt att utföra kortaste-avståndsberäkningar på de objekt som befinner sig nära varandra.

Indexeringsstrukturer är ofta indelade i två segment: ett primärfilter och ett inre filter. Det primära filtret utgör en första gallring och det inre filtret är ofta en mer kostsam kontroll för att hitta korrekt objekt. Målet är att minimera antalet anrop av det inre filtret för att uppnå en effektiv indexering.

Det primära filtrets effektivitet beräknas enligt Ekvation 7.1 (baserat på Technet 2013d)

$$\textit{Primärfilter effektivitet (\%)} = \frac{antal\ korrekta\ objekt}{antal\ objekt\ selekterade\ av\ primärfilter} \quad \text{Ekvation 7.1.}$$

*Antal korrekta objekt* är de som besvarar frågan korrekt. *Antal objekt selekterade av primärfilter* är alla objekt som är ”tillfrågade” efter att indexeringen har gjort sin gallring. Hög procentsats betyder att indexeringen inte ”släpper igenom” så många felaktiga objekt och färre objekt behöver matchas direkt mot frågan (vilket ofta är tidskrävande).

Det inre filtrets effektivitet beräknas enligt Ekvation 7.2 (baserat på Technet 2013d)

$$\textit{Inre filtrets effektivitet (\%)} = \frac{antal\ objekt\ selekterade\ av\ inre\ filter}{antal\ korrekta\ objekt} \qquad \text{Ekvation 7.2.}$$

Där *antalet objekt selekterade av inre filter = antal rader selekterade av primärfiltret – antal gånger det inre filtret anropas*. *Antal objekt selekterade av inre filter* kan med andra ord aldrig vara mindre än *antal korrekta objekt*.

Ett avgörande attribut för tidsåtgången vid användning av indexering är *antalet anrop det av inre filteret*. Detta värde anges ofta som en kvot, se Ekvation 7.3. (baserat på Technet 2013d)

$$P\ (\%) = \frac{antal\ objekt\ selekterade\ av\ inre\ filter}{antal\ objekt\ selekerade\ av\ primärfilter} \qquad \text{Ekvation 7.3.}$$

Där *antalet rader selekterade av inre filtret = antal rader selekterade av primärfiltret – antal gånger det inre filtret anropas*. Hög effektivitet hos primärfiltret innebär färre antal fel objekt släpps igenom. Vilket resulterar i minskat antal anrop av det inre filtret.

Tidkomplexiteten mäter tidsåtgången som krävs för att returnera (korrekt) svar på SQL-frågor. Det är i stort sett omöjligt att ange en matematisk formel för generell tidskomplexitet eftersom det är många faktorer som påverkar den. Påverkande faktorer är t.ex. SQL-frågans utformning och indexeringsstrukturens effektivitet.

Det är en väldigt svårt att för många olika typer av spatiala frågor skapa en indexerings-struktur som snabbt returnerar korrekta svar. Är indexeringsstrukturens effektivitet hög innebär det att den är väldigt noggrann i sitt första urval, t.ex. kan listorna i noderna vara väldigt små. Det skulle i sin tur innebära att det finns väldigt många noder att söka igenom för att hitta rätt nod. Men när rätt nod väl är funnen behöver få objekt kontrolleras. Beroende på hur avancerad spatial SQL-fråga som är ställd kan det vara fördelaktigt med olika hög indexeringseffektivitet.

# 8 Spatiala databasprodukter

Det finns flera databasprodukter på markanden för att lagra och hantera spatiala data effektivt. I det här kapitlet beskrivs två av de vanligaste och mest använda spatiala databasprodukterna PostGIS och SQL Server. De har båda kommunikationsspråket SQL och den vedertagna spatiala påbyggnaden *Simple Feature* gemensamt. Dock skiljer de sig kraftigt i andra avseenden, t.ex. vid hantering och upprättande av spatial indexering.

## 8.1 PostgreSQL

PostgreSQL är en objektrelationell databas utvecklat av Berkeley Computer Science Department vid University of California med öppen källkod. Sedan PostgreSQL släpptes år 1987 har databasen genomgått åtskilliga uppdateringar och använder sedan år 1996 SQL. I skrivande stund stödjer PostgreSQL SQL:2008 vilket medför att databasen kan hantera de standardiserade datatyperna som igår i SQL:2008. (PostgreSQL 2013)

### 8.1.1 Spatiala data

Det finns flera tilläggsprogram till PostgreSQL för hantering av spatiala data, bl.a. PostGIS och pgSphere (Obe & Hsu 2011). PostGIS är det vanligaste och mest använda, för hantering av två-dimensionella spatiala data. pgSphere används för att hantera tredimensionella spatiala data.

#### *8.1.1.1 PostGIS*

PostGIS är ett tillägg till PostgreSQL, utvecklat av OpenGeo med öppen källkod och släpptes år 2001 (Obe & Hsu 2011). PostGIS möjliggör lagring och hantering av tvådimensionella spatiala data genom den abstrakta klassen *Geometry* (OpenGEO 2013).

*Geometry* är utvecklad efter OGCs ISO-standard *Simple Feature* (ISO 19125) och kan skapa de nödvända geometriska objekten t.ex. punkter, linjer och polygoner. *Geometry*-klassen innehåller även en subklass kallad *Geometry Collection*, som möjliggör skapandet av objektlistor. *Geometry Collection* kan hantera både heterogena (t.ex. *Multipoints* och *Multipolygons*) och homogena (kombinationer av objekttyper, t.o.m. andra *Geometry Collection*-objekt) objektlistor (Obe & Hsu 2011).

#### *8.1.1.2 pgSphere*

pgSphere är ett tillägg som används för att hantera tredimensionella spatiala data med sfäriska koordinater. Koordinaterna utryckts antingen i decimalform eller i grader, minuter och sekunder. pgSphere kan utföra spatiala operationer på de tredimensionella objekten, t.ex. transformera koordinaterna mellan longitud-latitud och grader, minuter och sekunder, beräkna geodetiska koordinater (X, Y, Z) samt beräkna areor på ytor (pgSphere 2013).

De sfäriska datatyperna som pgSphere kan hantera är punkter, Euler transformation, linjer, linjesegment (eng. *linestrings*) och geometriska figurer (cirklar, ellipser och polygoner). Alla datatyper utom Euler transformationen kan användas vid skapandet av spatiala

indexeringar. pgSphere tillämpar skalstrukturen GiST vid upprättande av spatiala indexeringar (pgSphere 2013).

### 8.1.2 Spatial indexering

PostgreSQLs standard indexeringsalgoritm är B-träd, d.v.s. det är den algoritm som används om ingen annan specifik algoritm anges vid skapandet av indexering (PostgreSQL 2013). För textuella- och numeriska data är B-träd fullt funktionsdugligt men för att upprätta spatial indexering krävs andra indexeringsalgoritmer. Vilken algoritm som används är beroende av vilket tillägg som används för hantering av spatiala data. Tilläggsprogrammen PostGIS och pgSphere kan båda implementera GiST-indexering (Obe & Hsu 2011).

PostGIS implementerar en egenutvecklad version av den ”allmänna” GiST-indexeringen. GiST-indexeringen bygger på R-trädet (se kapitel 7.4.2.2) men är lite mer flexibelt. PostGIS GiST-indexering kan t.ex. hantera stora geometrier och tillåter geometrier med värdet *null* (PostGIS 2013).

## 8.2 SQL server

SQL Server är en objektrelationell databas utveckad av Microsoft Corporation. SQL Server är sedan version 2005 integrerat med CLR (*Common Language Runtime*) som är uppbyggt av .NET-ramvärket (Technet 2013b).

### 8.2.1 Spatiala data

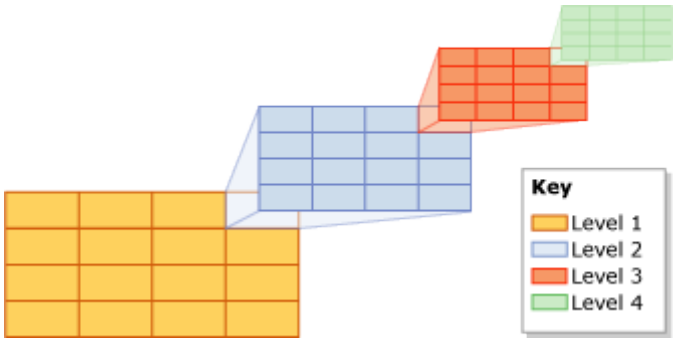
SQL Server kan sedan version 2008 hantera spatiala data med hjälp av de två spatiala datatyper *Geometry* och *Geography* (Technet 2013c). *Geometry* används för att representera två-dimensionella data i ett Euklidiskt koordinatsystem. *Geography* används vid hantering av tredimensionella data och kan hantera både jordcentrerat kartesiskt koordinatsystem (X,Y, Z) och geografiskt koordinatsystem (longitud och latitud). De båda spatiala datatyperna är baserade på *Simple Features* (ISO 19125) datatyp *Geometry* (Technet 2013c). Figur 6.4 visar UML-diagram över *Simple Feature* och Figur 6.5 visar uppbyggnaden av *Geometry*-klassen.

SQL Server delar in de spatiala datatyperna i två klasser: *Simple types* och *Collection types* (Technet 2013c). *Simple types* består av en objekttyp, t.ex. punkt- eller polygonobjekt. *Collection types* består av en lista med flera objekt av samma typ, t.ex. punktlista (eng. *Multipoint*) eller *GeometryCollection*. *GeometryCollection* är en vektorklass som kan innehålla flera olika typer av geometriska objekt i samma vektor medan t.ex. punktlistor bara kan innehålla punktobjekt (Technet 2013c).

### 8.2.2 Spatial indexering

SQL Server använder en rumsligt baserad hierarkisk indexeringsmetod, kallad *Tessellation*, för att indexera två-dimensionella spatiala objekt. *Tessellation* består av åtta indexeringsnivåer, benämns från nivå 1 till nivå 8 (vid manuell genererad indexering används endast fyra nivåer). Varje nivå består av en kvadratisk mängd celler (4x4, 8x8 eller 16x16). (Technet 2013a)

Antal celler, s.k. celldensiteten, avgör hur noggrann indexeringen är. Cellerna medför att endast objekt i ”rätt” cell behöver undersökas. Hög celldensitet är viktigt när datamängden är kompakt, d.v.s. många objekt på liten yta. Indexeringensmetoden lagrar alla data i ett B-träd för att möjliggöra tideffektiv hämtning och sökning av data. Varje cell sönderdelas därefter i valt antal nya celler (olika nivåer behöver inte innehålla samma antal celler), se Figur 8.1. För att numrerar cellerna samt tillämpa ”regeln” om spatial närhet används Hilbert-kurvan. (Technet 2013a)




**Figur 8.1. Beskrivning av indexeringsmetoden *Tessellation* (manuell), varje nivå består av 4x4 celler (Technet 2013b).**

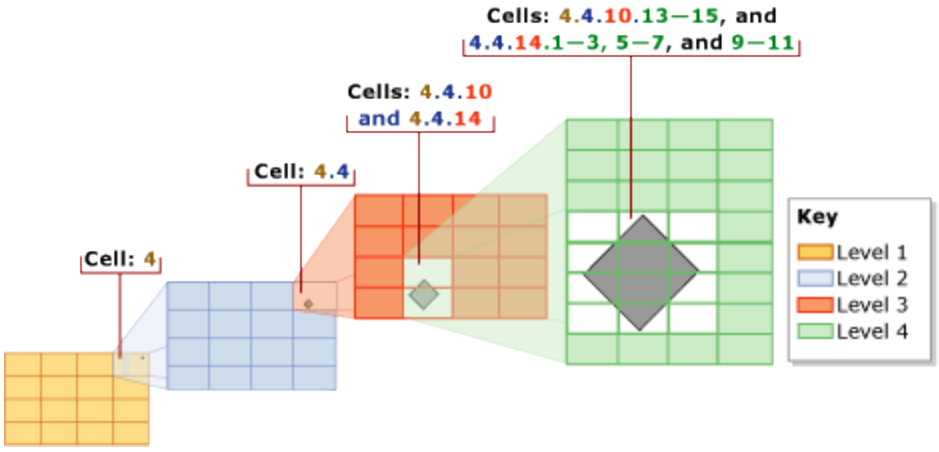
*Tessellation* använder approximationsmetoden minsta geografiska utbredning (eng. *Minimum Bounding Box*) för att avgöra inom vilka celler ett objekt befinner sig. För att minimera mängden data som lagras med hjälp av *Tessellation*-algoritmen är algoritmen uppbyggd efter följande regler (baserad på Technet 2013a):

- fullständig täckning
- antal celler per objekt
- högsta möjliga nivå.

Regeln om fullständig täckning hanterar polygoner som helt täcker indexeringsceller. Täcker ett objekt fullständigt en cell sönderdelas inte den specifika cellen ytterligare, det vore onödigt eftersom nästa cellnivå kan ses som en inzoomad version av en tidigare nivå, t.ex. cell 4.4.14.6 och 4.4.14.10 i Figur 8.2.

Antal celler per objekt-regeln begränsar antalet celler som lagras i indexeringen för ett objekt. Uppnås antalet celler sönderdelas inte cellen som innehåller objektet ytterligare. Standardvärde för regeln är 16, men användaren kan ange ett värde mellan 1-8192.

Regeln om högsta möjliga nivå begränsar mängden data som indexeras genom indexeringens hierarkiska uppbyggnad vilket avspeglas i cellernas namnsättning. Indexeringen som utförs i Figur 8.2 behöver inte lagra de tre första nivåerna eftersom de indirekt är inkluderade i den fjärde och högsta nivån.




**Figur 8.2. Ett exempel av *Tessellation*-algoritmen, varje nivå består av 4x4 celler och den minsta geografiska utbredningen (vit rektangel) används för att avgöra vilka celler som berörs. I detta förenklade exempel används inte Hilbert-kurvan för att numrera cellerna utan en radvis numrering nyttjas. (Technet 2013a)**

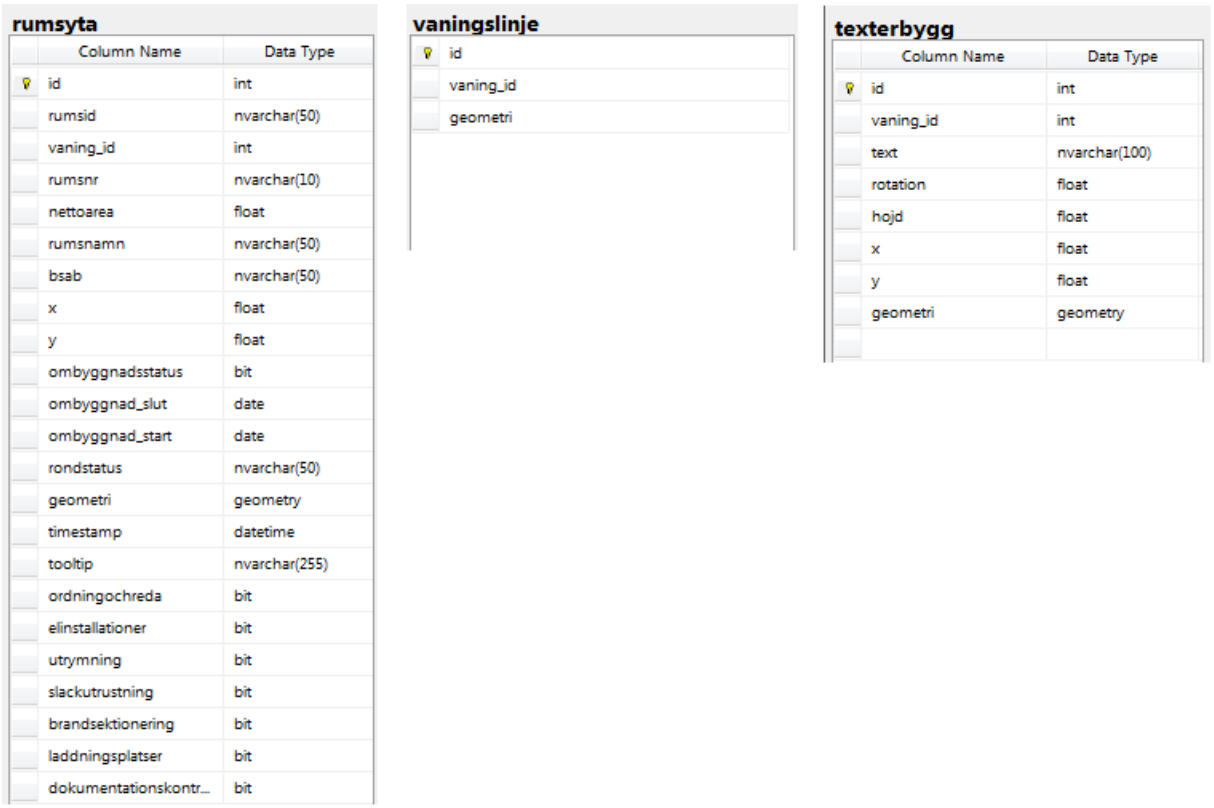
# 9 Fallstudie och data

Målet med projektet är att finna effektivt datahanteringssystem vid integrering av BIM-data och GIS. Problemlösningen delas i två delar: integrering av BIM-data i GIS och prestandaoptimering av databasprodukter. Integrering av BIM-data behandlar problemet med att BIM-data sällan är geografiskt refererade och är uppbyggd enligt andra standarder än de i GIS.

Del två innebär att prestandaoptimerade åtgärder utförs på de spatiala databasprodukterna med målet att finna vilka åtgärder som har störst påverkan. Två databasprodukter testas och jämförs och alla prestandaoptimerade åtgärder dokumenteras och beskrivs. För att möjliggöra tester av prestandaoptimerade åtgärder krävs det att indata konverteras till lämpliga format och att de spatiala objekten blir geografiskt refererade. Den geografiska refereringen av de spatiala objekten är viktig eftersom objektens placering i planet har en stor inverkan på spatiala indexeringsstrukturers inställningar och utformning.

## 9.1 Ursprungliga data

Indata för studien är Swedavias data över Stockholm Arlanda Airport i Sverige. Alla data i detta skede är endast konverterade till korrekt databasformat och därmed helt obehandlade. Indata består av flera tabeller, både med och utan spatiala data. Tabellerna innehållandes de spatiala objekten är de som är av intresse i det här projektet, se Figur 9.1.



**rumsyta**

| Column Name | Data Type |
|---|---|
| id | int |
| rumsid | nvarchar(50) |
| vaning_id | int |
| rumsnr | nvarchar(10) |
| nettoarea | float |
| rumsnamn | nvarchar(50) |
| bsab | nvarchar(50) |
| x | float |
| y | float |
| ombyggnadsstatus | bit |
| ombyggnad_slut | date |
| ombyggnad_start | date |
| rondstatus | nvarchar(50) |
| geometri | geometry |
| timestamp | datetime |
| tooltip | nvarchar(255) |
| ordningochreda | bit |
| elinstallationer | bit |
| utrymning | bit |
| slackutrustning | bit |
| brandsektionering | bit |
| laddningsplatser | bit |
| dokumentationskontr... | bit |

**vaningslinje**

| id |
|---|
| vaning_id |
| geometri |

**texterbygg**

| Column Name | Data Type |
|---|---|
| id | int |
| vaning_id | int |
| text | nvarchar(100) |
| rotation | float |
| hojd | float |
| x | float |
| y | float |
| geometri | geometry |

**Figur 9.1. De tre tabellerna i datamängden som innehåller spatiala data. Attributet innehållandes spatiala data är *geometri, id* är primärnycklar i samtliga tabeller.**

Swedavias byggnadsstruktur är hierarkiskt uppbyggd. Tabellen *Rumsyta* består av cirka 16 000 objekt fördelade över 280 våningar. Tabellen beskriver byggnadernas rumsfördelning i form av polygoner och innehåller bl.a. information om vilken våning (*vaning_id*) rummet tillhör. Varje rum innehåller även rumsspecifika data t.ex. rumsstorlek (*nettoarea*), om rummet är under ombyggnad (*ombyggnadsstatus*) och om rummet innehåller släckutrustning (*slackutrustning*). Våningstabellen (*Vaning*) utgör en icke-spatial tabell som är kopplad till byggnadstabellen (*Byggnad*) som beskriver byggnadstypen.

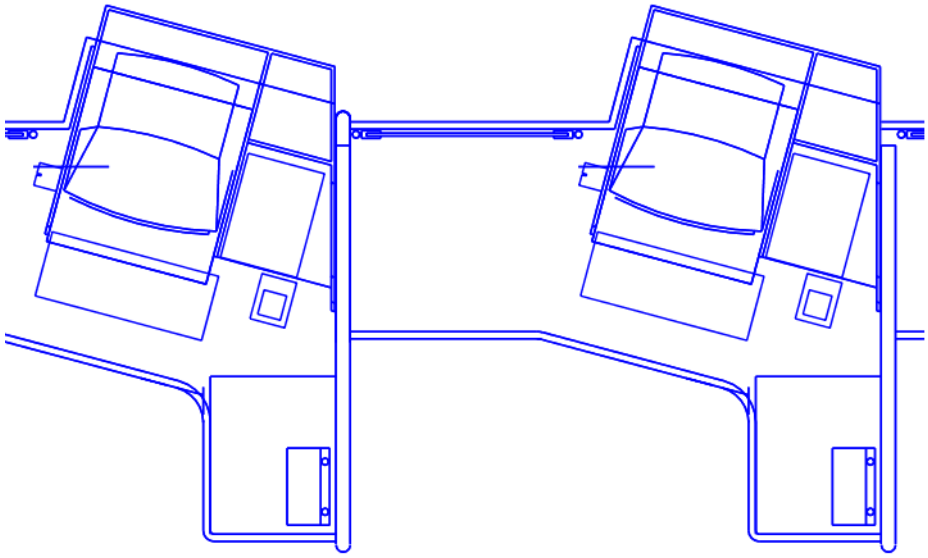
Våningslinjetabellen (*Vaningslinje*) består av cirka 1 580 000 linjeobjekt fördelade över de 280 våningarna. Tabellen innehåller varje linjeobjekt som beskrivs i de ursprungliga CAD-ritningarna. Texttabellen (*Texterbygg*) innehåller cirka 18 000 textobjekt med storlek (*hojd*) och lutning (*rotation*) och beskriver vilken typ av byggnad och rum som syns i kartan.

Fördelningen av objekten är bland våningarna är väldigt skev, se Tabell 9.1; det beror främst på den stora variationen av våningsstorlek. Våningsstorleken varierar från 6,6 $m^2$ till 23 910 $m^2$ med en medelstorlek på 2 888 $m^2$. Större våningar innehåller generellt fler objekt, t.ex. innehåller den största våningen (våning 130) flest polygon-, linje- och punktobjekt.

**Tabell 9.1. Fördelningen av spatiala objekt per tabell. Tabellen redovisar det lägsta (min), största (max) och medelvärdet av antal objekt för någon våning.**

| tabellnamn | min | max | medelvärde |
|---|---|---|---|
| *Rumsyta* | 1 | 595 | 61,5 |
| *Vaningslinjer* | 16 | 120 918 | 6 817,1 |
| *Texterbygg* | 1 | 588 | 70,8 |

Eftersom datamängden härstammar från tekniska ritningar är detaljrikedomen och noggrannheten mycket stor, se Figur 9.2. Den spatiala felsäkerheten uppgår till 4 mm för inomhusritningarna. Det innebär att måtten i ritningarna är väntevärden med standardavvikelse på 4 mm (97,5% säkerhet). Dessutom är ritningarna upprättade i separata CAD-filer i lokala och geografiskt referensoberoende koordinatsystem. Det medför att våningsplanen inte har någon rumslig referens till varandra och många är belägna inom samma utbredning kring origo, t.ex. är 230 av de 280 våningar belägna innanför våning 130s minsta geografiska utbredning vilket motsvarar ca 1 250 000 linjeobjekt inom utrymmet.



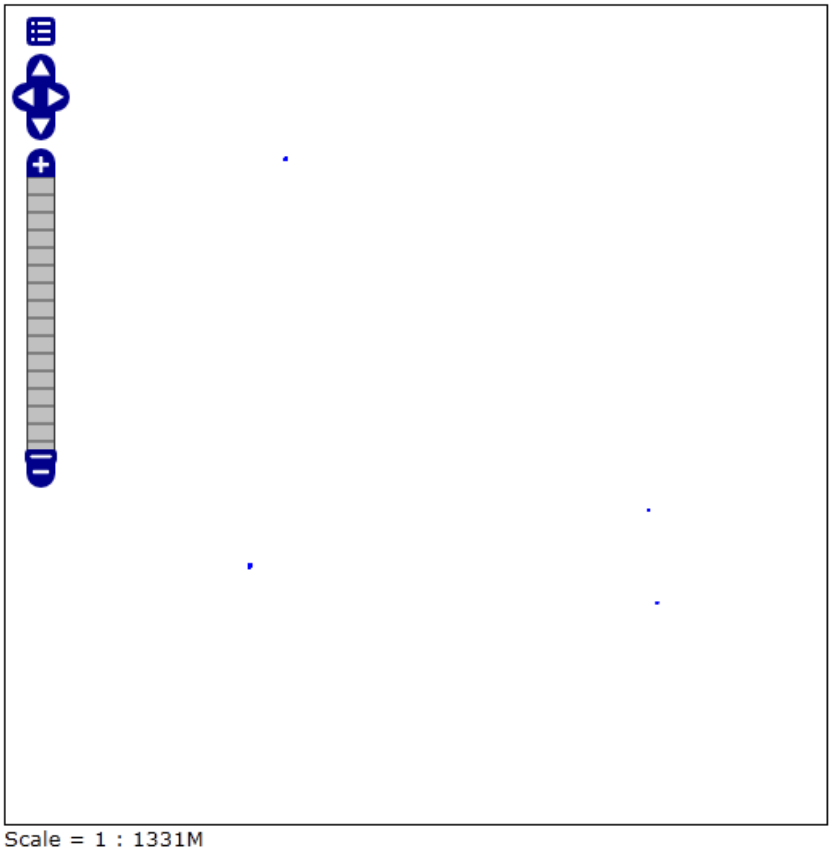
**Figur 9.2. Exempelbild över detaljrikedomen i våningslinjetabellen. Figuren visar en incheckningsdisk med två datorplatser.**

Eftersom indata ursprungligen är CAD-ritningar är alla mått uttryckta i millimeter och vissa CAD-ritningar innehåller negativa koordinatvärden, se Tabell 9.2 som beskriver den minsta geografiska utbredningen för våning 130.

**Tabell 9.2. Den geografiska utbredningen för våningslinjerna för våning 130. Beräknad i GeoServer med hjälp av *Compute from data*. Koordinaterna är uttryckta i ett lokalt koordinatsystem och enheten är millimeter.**

| **Minsta geografiska utbredning** | |
|---|---|
| Min X | -105 844 |
| Min Y | -45 650 |
| Max X | 362 618 |
| Max Y | 265 604 |

Alla data är dock inte centrerad kring origo utan vissa våningsplan är placerade väldigt långt ifrån varandra vilket medför att den totala utbredningen för datamängden är enorm, se Figur 9.3.

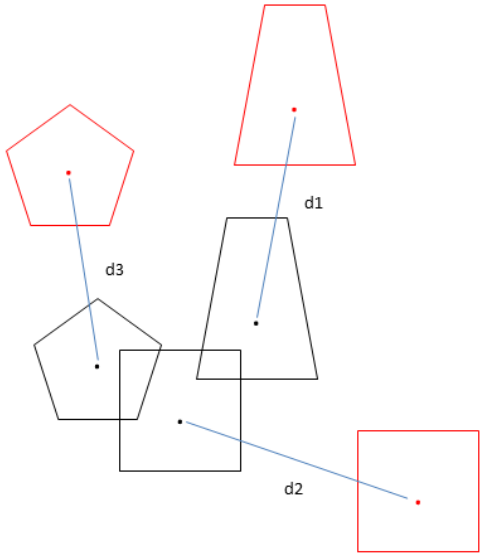



**Figur 9.3. Översikt över den obehandlade datamängdens utbredning. Majoriteten av objekten är lokaliserade i den nedre vänstra delen vilket även motsvarar origo. Skalan är 1:1 331 000 000 vilket visar att utbredningen är enorm.**

## 9.2 Behandling av data

Indata i ursprungligt format innebär att några prestandaoptimerade åtgärder i de spatiala databasprodukterna inte kan utföras. Det huvudsakliga problemet är att spatiala BIM-data inte är geografiskt refererade utan bara är godtyckligt placerade i planet. Tester visar att även om de spatiala objekten indexeras korrekt avgör databasprodukternas frågeoptimerare att frågan besvaras snabbare utan att använda den spatiala indexeringen.

En förenklad geografisk referering utförs med hjälp av mjukvaran FME. Samtliga objekt i tabellerna *Texterbygg*, *Rumsyta* och *Vaningslinje* tilldelas ett byggnadsnummerattribut som anger vilken byggnad objektet tillhör. Information finns om byggnadernas korrekta geografiska positioner och den geografiska tyngdpunkten för varje byggnad beräknas. Den spatiala noggrannheten i Swedavias utomhuskartor är 100 mm. D.v.s. byggnadens korrekta position är ett väntevärde med 100 mm i standardavvikelse (97,5% säkerhet). Därefter beräknas skillnaden för de korrekt placerade byggnadernas tyngdpunkt med tyngdpunkten för byggnaderna avgiva i databasen (benämns d1, d2 och d3 i Figur 9.4). Varje spatialt objekt flyttas sedan den beräknade sträckan, se Figur 9.4.

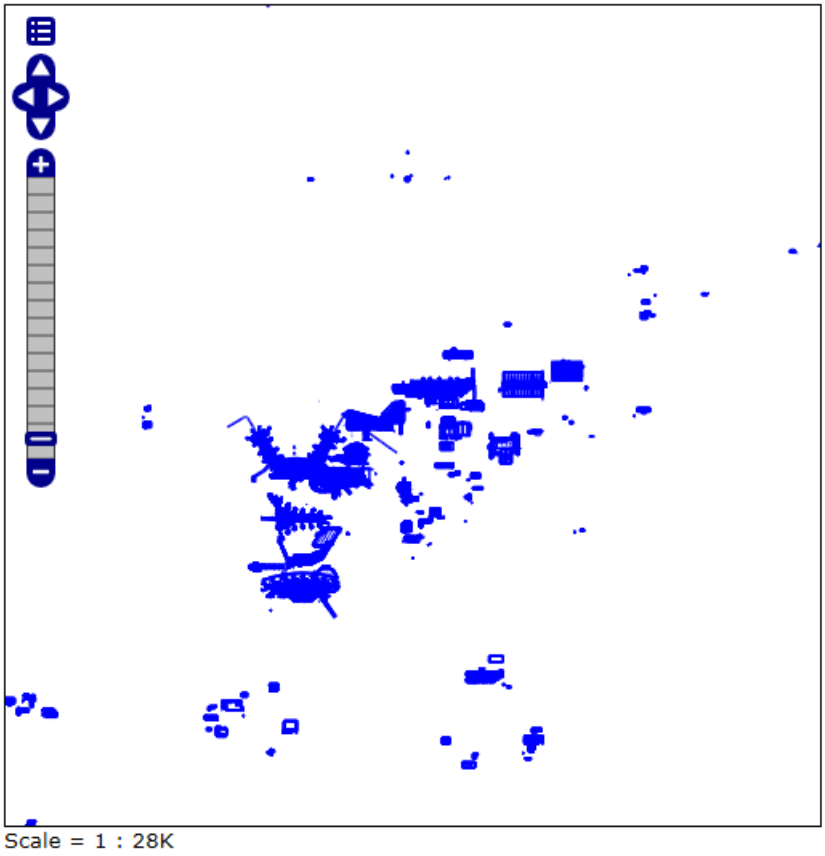



**Figur 9.4. Förenklad illustration över geografisk referering. De svarta objekten är byggnadernas positioner i databasen och de röd är byggnadernas korrekta geografiska positioner. Byggnadernas tyngdpunkter beräknas varpå alla spatiala objekt i databasen flyttas den beräknade sträckan.**

I samband med geografisk referering av de spatiala objekten utförs även en konvertering av objektens metriska enheter. Objektens måttenheter är ursprungligen uttryckta i millimeter. Konverteringen resulterar i att alla mått uttrycks i meter vilket är standard vid användning av geografiska koordinatsystem.

Resultatet av den geografiska refereringen syns i Figur 9.5. Byggnaderna är nu placerade vid sin geografiskt korrekta position. Området är i stort sett kvadratiskt med den totala arean på nästan 16 000 $m^2$. Fördelningen av data är fortfarande väldigt skev, stor del av datamängden återfinns på en liten yta i mitten av området. Dessutom har flera av byggnaderna i området fler än en våning (byggnaderna består av upp till 10 våningar) vilket ökar den redan kompakta datamängden i dessa områden.

Den spatiala säkerheten av den förenklade geografiska refereringen är väldigt låg. T.ex. har ingen vridning av byggnaderna utförts utan byggnaderna har endast flyttats till närheten av sin korrekta geografiska position med hjälp av tyngdpunktsberäkningar. Består byggnaden av flera våningar används medelvärdet av våningarnas tyngdpunker. Det rekommenderas inte att utföra geografiska refereringar enligt denna metod om byggnadernas korrekta position är viktig för tillämpningen. I aktuellt fall anses däremot den förenklade geografiska refereringen tillräcklig eftersom fokus i studien är att analysera spatiala indexeringsmetoder och det viktiga är att objekten är ”naturligt” fördelade i planet och inte centrerade kring en punkt.




**Figur 9.5. Utbredningen av indata efter att objekten blivit geografiskt refererade, jämför med Figur 9.3. Skalan är nu 1: 28 000 vilket är rimligt för datamängden.**

Tabell 9.3 visar den minsta geografiska utbredningen för våning 130, nu är enheterna meter i stället för millimeter (jämför med Tabell 9.2). Efter denna behandling av indata kan prestandaoptimerade åtgärder korrekt utföras på datamängden.

**Tabell 9.3. Den minsta geografiska utbredningen för våning 130. Beräknad i GeoServer med hjälp av *Compute from data*. Koordinaterna är uttryckta i koordinatsystemet SWEREF 99 18 00 (EPSG:3011) och enheten är nu meter.**

| **Minsta geografiska utbredning** | |
|---|---|
| Min X | 145 672 |
| Min Y | 6 614 965 |
| Max X | 146 140 |
| Max Y | 6 615 276 |

## 9.3 Mjukvaror

De spatiala databasprodukterna som analyseras och utvärderas är:

1. SQL Server 2012 – SQL Server Express v. 11.0.2100.60
2. PostgreSQL – PostGIS pgAdmin v. 1.14.3

Beslutet togs eftersom de är bland de databasprodukterna med flest användare samt att de har olika indexeringsmetoder. Databasprodukterna är också kompatibla med GeoServer som används som kartserver i projektet.

Användningen av expressversionen av SQL Server 2012 anses inte ha någon inverkan på prestandaanalyserna. SQL Server Express begränsar bl.a. antalet värddatorer till en, databasens maximala lagringsutrymme till 10 GB och användning av datorns minne till 1 GB (Microsoft 2013a). Övriga begränsningar för SQL Server 2012 Express berör säkerheten, möjligheten med online säkerhetskopiering och molnhantering (eng*. Cloud Management*) vilket inte påverkar studiens prestandatester.

GeoServer (v. 2.4.0) används som kartserver för att generara de önskade kartbilderna i webbläsaren. Tidsåtgången för kartgenereringen (mottagningen av kartbilden) mäts i millisekunder med Internet Explorers (v. 9.0.8112.16421) inbyggda nätverkstrafikregistrering (F-12 utvecklingsverktyg, nätverk). GeoServer används för att kunna mäta prestandan på hela det tredelade systemet. Det är av intresse eftersom de flesta GIS idag tillämpar en sådan systemarkitektur.

GeoServer valdes att användas som kartserver av flera anledningar, bl.a. är den utvecklad med öppen källkod vilket inte kräver några extra licenser. GeoServer är även kompatibel med de båda spatiala databasprodukterna som ska utvärderas. Vidare har GeoServer en *Layer Preview*-funktion som kan användas som en förenklad klient i prestandatesterna. Eftersom målet med studien är att undersöka databasernas inverkan på systemets prestanda ansågs val av kartserver sekundärt och eftersom GeoServer uppfyller samtliga krav för att möjliggöra prestandatester utsågs GeoServer som studiens kartserver.

## 9.4 Prestandaoptimering

Prestandaoptimerade åtgärderna utförs på Swedavias data efter behandlingen av indata. Prestandatesterna sker separat för de två databasprodukterna eftersom databaserna har olika optimeringsmöjligheter. Objekttyperna (linjer, polygoner och punkter) testas och jämförs separat inom varje databasprodukt. Prestandatesterna utförs i två olika system: ett system med tredelad systemarkitektur där prestandan för hela kedjan utvärderas samt ett system som förbiser mellanhanden och prestandatesterna utförs direkt i databasen.

Prestandatesterna för den tredelade systemarkitekturen utförs genom att mäta svarstiden vid *GetMap*-anrop för att rendera alla objekt för valt våningsplan. Prestandatesterna som utförs direkt i databasen är för att tydligt isolera databasernas inverkan på systemet samt undvika användningen av spatiala SQL-frågor som kartservrar automatiskt genererar vid visualisering av spatiala data.

För våningslinjetabellen mäts svarstiderna för våning 28 och våning 299. Våning 28 är en av de våningarna med flest linjeobjekt (ca 92 000). Från början var det planerat att mäta svarstiden från den största våningen som också har flest linjeobjekt (våning 130). Tester visade att frågeoptimerare i de spatiala databasprodukterna bortser från den spatiala

indexeringen och utför endast linjärsökning i datamängden vid *GetMap*-anrop av våning 130. Därför beslutades det att testa och analysera *GetMap*-anrop av våning 28 istället. Våning 299 är den mest kompakta våningen (flest linjeobjekt per $m^2$) och liknar även ”medelvärde”-våningen i antal linjeobjekt, ca 6400 linjeobjekt.

Prestandatesterna för polygonobjekt utförs på våningarna 130 och 123. Våning 130 är den geografiskt största våningen och innehåller flest polygonobjekt (595 st.). Våning 123 är lik ”medelvåningen” för polygonobjekt med 61 polygoner.

Punktobjektens prestandatester utförs på våning 130 och våning 148. Våning 130 den geografisk störta våningen och innehåller flest punktobjekt (588 st.). Våning 148 är mest lik ”medelvåningen” för punkobjekt med 72 punkter.

Optimeringsförsöken jämförs med databasernas standard indexeringsinställningar för varje objekttyp för att ge en tydlig referenspunkt. För varje prestandatest kopieras originaltabellen och kopian tilldelas en unik indexering varpå en vy för önskad våning generas på den kopierade tabellen. Vyn läggs till som ett lager i GeoServer och svarstiderna för tio *GetMap*-anrop mäts och medelvärdet redovisas.

Prestandatester utförs även på indata som inte blivit geografiskt refererade för att tydligt markera skillnaden i prestanda. Dessa prestandatest utförs endast på linjeobjektstabellen på vyer av våningarna 28 och 299. Testerna utförs både via GeoServer och som isolerade SQL-frågor.

### 9.4.1 GetMap-anrop

*GetMap*-anrop kan ske mot en vy genererad över önskad våning skapad i databasen. Ett annat alternativ är att ställa anropet mot ”hela” tabellen i databasen där endast objekten för önskad våning är synliga med hjälp av ett filter i SLD-filen. Skillnaden i *SELECT*-klausulen för en vy eller ”hela” tabellen syns i Figur 9.6. Ett filter i SLD-filen medför att *GetMap*-anropet bifogar *vaning_id*=130 som ett kriterium i *WHERE*-klausulen istället för att kriteriet sker när vyn skapas.

**SLD-fliter**

```
SELECT "id","vaning_id","geometri".STAsBinary() as "geometri" FROM "lines_only_auto_16" WHERE
("geometri".Filter(geometry::STGeomFromText('POLYGON  ((-343736.3671875 -204772.078125, -
343736.3671875 424725.078125, 600509.3671875 424725.078125, 600509.3671875 -204772.078125, -
343736.3671875 -204772.078125))', 3785)) = 1 AND "vaning_id" = 130)
```

**Vy**

```
SELECT "geometri".STAsBinary() as "geometri" FROM "vy_lines_only_auto_16" WHERE
"geometri".Filter(geometry::STGeomFromText('POLYGON  ((-343735.4195784598 -204771.53941560988,
-343735.4195784598 424725.5951503299, 600510.2822704399 424725.5951503299,
600510.2822704399 -204771.53941560988, -343735.4195784598 -204771.53941560988))', 3785)) = 1
```

**Figur 9.6. Skillnaden i *GetMap*-anropet för en vy eller ett SLD-filter för att begränsa att endast våning 130 visas. Sker *GetMap*-anropet med ett SLD-filter bifogas våningsnumret (*vaning_id*) direkt i *WHERE*-klausulen.**

Tester visar att svarstiden är i stort sett oberoende av om *GetMap*-anropet sker mot en vy eller med hjälp av ett SLD-filter. Samtliga prestandaoptimerade tester utförs mot en vy skapad i databasen.

Varje *GetMap*-anrop sker 10 gånger och medelvärdet redovisas i rapporten. Beroende av vilken databasprodukt som anropas är SQL-uttrycken olika utförda, se Figur 9.7. Det som skiljer SQL-uttrycken är urformningen av *WHERE*-klausulen där produktspecifika funktioner används (funktionerna kallas *Filter* och *&&* för de olika databasprodukterna).

**SQL Server 2012**

```
SELECT "geometri".STAsBinary() as "geometri" FROM "vy_lines_only_high"
WHERE  "geometri".Filter(geometry::STGeomFromText('POLYGON  ((-
343735.4195784598 -204771.53941560988, -343735.4195784598
424725.5951503299, 600510.2822704399 424725.5951503299,
600510.2822704399 -204771.53941560988, -343735.4195784598 -
204771.53941560988))', 3785)) = 1
```

**PostGIS 9.1**

```
SELECT encode(ST_AsBinary(ST_Force_2D("geom")),'base64') as "geom" FROM
"public"."lines_only_van130" WHERE  "geom" && ST_GeomFromText('POLYGON
((-343735.4195784598 -204771.53941560988, -343735.4195784598
424725.5951503299, 600510.2822704399 424725.5951503299,
600510.2822704399 -204771.53941560988, -343735.4195784598 -
204771.53941560988))', 97589)
```

**Figur 9.7. SQL-utrycket som skickas till databaserna (hämtat från geoserver.log, Service info - DEBUG).**

# 10 Prestandaoptimering PostGIS 9.1

PostGIS standard indexeringsmetod för spatiala data är GiST. Det finns flera möjligheter för användaren att anpassa indexeringsstrukturen efter just sina behov. Flera olika inställningar kombineras och har testats för att uppnå bästa resultat. Varje test genomfördes genom att mäta svarstiden för tio *GetMap*-anrop, medelvärdet redovisas nedan under resultat.

## 10.1 Metod

Nedan beskrivna optimeringsmetoder utförs på de tre olika objekttyperna (punkter, linjer och polygoner). För varje objekttyp testas optimeringsmetoderna på olika våningar beroende av våningarnas storlek och hur många objekt våningen innehåller, allt för att göra testerna så relevanta och aktuella som möjligt utan att behöva testa samtliga våningsplan. Inställningarna gäller för samtliga objekttyper.

### 10.1.1 Standard indexering

Standard indexeringsmetod för spatiala data är GiST med fyllnadsgrad (eng. *fill factor*) på 90%.

### 10.1.2 Primär spatial indexering

Standard inställningar för GiST-indexeringsstruktur tillämpades. Den spatiala GiST-indexeringen användes för att sortera datamängden fysiskt i hårddisken med SQL-kommandot:

*cluster* my_table *using* gist_index.

GiST-indexeringen blev därmed en s.k. primär indexering.

### 10.1.3 Primär spatial indexering med sekundär icke-spatial indexering

Standard inställningar för GiST-indexeringsstruktur tillämpades. Den spatiala GiST-indexeringen användes för att sortera datamängden fysiskt i hårddisken (ett s.k. primärindex).

En sekundär icke-spatial indexeringsstruktur skapades efter vånings id (*vaning_id*); ett klassikt B-träd användes.

### 10.1.4 Primär icke-spatial indexering med sekundär spatial indexering

Klassiskt B-träd används för att skapa en icke-spatial primär indexering över vånings id (*vaning_id*). En sekundär spatial GiST-indexering skapas också.

### 10.1.5 SQL-frågor direkt i PostGIS

Detta test utförs genom att isolera databasen från klient och kartserver, därmed tydligt avgöra vilken inverkan inställningarna för de olika spatiala indexeringsmetoderna har. Endast våningslinjetabellen (*Vaningslinje*) har tillräckligt många objekt för att denna analys ska vara aktuell. Därför utförs testerna endast på denna tabell.

Tillämpningen av Swedavias GIS är att visualisera byggnadsobjekt för hela våningsplan. Samtliga objekt har en våningstillhörighet (attribut *vaning_id*). Varpå tester utförs för att undersöka om det är snabbare att hitta alla sökta objekt utan att generera spatiala SQL-frågor (vilket många kartservrar automatiskt gör). D.v.s. Svarstiderna jämförs för SQL-anrop direkt i databasen vid hämtning av en vånings samtliga linjeobjekt. Uppvisad bästa spatiala indexeringsmetod jämförs mot icke-spatial fråga och indexering.

## 10.2 Resultat

I det här avsnittet presenteras resultatet för samtliga prestandaoptimerade åtgärder som har utförts i PostGIS 9.1. Resultaten redovisas per objekttyp.

### 10.2.1 Punktobjekt

Datamängden punktobjekt är relativt liten med cirka 18 000 objekt. Det resulterar i att indexeringstesterna uppvisar snarlika värden oavsett indexeringsjusteringar. Tabell 10.1 visar medelvärdet för svarstiderna för samtliga prestandaoptimerade försök utförda på en vy skapad över våning 130 från den geografiskt refererade punktobjekts-tabellen (*Texterbygg*) i PostGIS 9.1.

**Tabell 10.1. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 130 från punktobjektstabellen (*Texterbygg*). Dubbel indexering betyder att tabellen har två indexeringar, spatial och icke-spatial.**

| Indexeringsstruktur | Svarstid (ms) |
|---|---|
| **Enkel indexering:** | |
| GiST (standard) | 63 |
| GiST (primär indexering) | 66 |
| **Dubbel indexering:** | |
| GiST och B-träd | 61 |
| GiST (primär indexering) och B-träd | 63 |
| B-träd (primär indexering) och GiST | 62 |

Tabell 10.2 visar medelvärdet för svarstiderna för samtliga prestandaoptimerade försök utförda på en vy skapad över våning 148 från den geografiskt refererade punktobjekts-tabellen (*Texterbygg*) i PostGIS 9.1.

**Tabell 10.2. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 148 från punktobjektstabellen (*Texterbygg*). Dubbel indexering betyder att tabellen har två indexeringar, spatial och icke-spatial.**

| Indexeringsstruktur | Svarstid (ms) |
|---|---|
| **Enkel indexering:** | |
| GiST (standard) | 39 |
| GiST (primär indexering) | 41 |
| **Dubbel indexering:** | |
| GiST och B-träd | 41 |
| GiST (primär indexering) och B-träd | 43 |
| B-träd (primär indexering) och GiST | 40 |

### 10.2.2 Linjeobjekt

Tabell 10.3 visar medelvärdet för svarstiderna för samtliga prestandaoptimerade försök utförda på en vy skapad över våning 28 från den geografiskt refererade linjeobjektstabellen (*Vaningslinje*) i PostGIS 9.1. Det är tydligt att det är fördelaktigt att tillämpa dubbel indexering när datamängden ökar. Svarstiderna minskar med ca 5,5 – 7,1% vid tillämpning av sekundär icke-spatial indexering för linjeobjektstabellen. Resultatet visar även att det är stor skillnad om datamängden är geografiskt refererad eller inte.

**Tabell 10.3. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 28 från linjeobjektstabellen (*Vaningslinje*). Dubbel indexering betyder att tabellen har två indexeringar, spatial och icke-spatial.**

| Indexeringsstruktur | Svarstid (ms) |
|---|---|
| **Enkel indexering:** | |
| GiST (standard) | 1 049 |
| GiST (primär indexering) | 1 050 |
| GiST (icke-georefererade) | 1 474 |
| **Dubbel indexering:** | |
| GiST och B-träd | 995 |
| GiST (primär indexering) B-träd | 1 014 |
| B-träd (primär indexering) och GiST | 994 |
| GiST och Hash | 1 036 |
| Gist (primär indexering) och Hash | 1 034 |

Tabell 10.4 visar medelvärdet för svarstiderna för samtliga prestandaoptimerade försök utförda på en vy skapad över våning 299 från den geografiskt refererade linjeobjektstabellen (*Vaningslinje*) i PostGIS 9.1.

**Tabell 10.4. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 299 från linjeobjektstabellen (*Vaningslinje*). Dubbel indexering betyder att tabellen har två indexeringar, spatial och icke-spatial.**

| Indexeringsstruktur | Svarstid (ms) |
|---|---|
| **Enkel indexering:** | |
| GiST (standard) | 120 |
| GiST (primär indexering) | 125 |
| GiST (icke-georefererade) | 141 |
| **Dubbel indexering:** | |
| GiST och B-träd | 115 |
| GiST (primär indexering) och B-träd | 112 |
| B-träd (primär indexering) och GiST | 115 |
| GiST och Hash | 123 |
| GiST (primär indexering) och Hash | 120 |

### 10.2.3 Polygonobjekt

Tabell 10.5 visar medelvärdet för svarstiderna för samtliga prestandaoptimerade försök utförda på en vy skapad över våning 130 från den geografiskt refererade polygonobjekts-tabellen (*Rumsyta*) i PostGIS.

**Tabell 10.5. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 130 från polygonobjektstabellen (*Rumsyta*). Dubbel indexering betyder att tabellen har två indexeringar, spatial och icke-spatial.**

| Indexeringsstruktur | Svarstid (ms) |
|---|---|
| **Enkel indexering:** | |
| GiST (standard) | 98 |
| GiST (primär indexering) | 95 |
| **Dubbel indexering:** | |
| GiST och B-träd | 98 |
| GiST (primär indexering) och B-träd | 94 |
| B-träd (primär indexering) och GiST | 103 |
| GiST och Hash | 95 |
| GiST (primär indexering) och Hash | 97 |

Tabell 10.6 visar medelvärdet för svarstiderna för samtliga prestandaoptimerade försök utförda på en vy skapad över våning 148 från den geografiskt refererade polygonobjekts-tabellen (*Rumsyta*) i PostGIS.

**Tabell 10.6. Medelvärdet av svarstiderna från 10 GetMap-anrop mot en vy av våning 148 från polygonobjektstabellen (*Rumsyta*). Dubbel indexering betyder att tabellen har två indexeringar, spatial och icke-spatial.**

| Indexeringsstruktur | Svarstid (ms) |
|---|---|
| **Enkel indexering:** | |
| GiST (standard) | 78 |
| GiST (primär indexering) | 78 |
| **Dubbel indexering:** | |
| GiST och B-träd | 78 |
| GiST (primär indexering) och B-träd | 74 |
| B-träd (primär indexering) och GiST | 78 |
| GiST och Hash | 78 |
| GiST (primär indexering) och Hash | 78 |

### 10.2.4 SQL-frågor direkt i PostGIS

Tabell 10.7 visar svarstiderna för SQL-frågorna som utfördes direkt i PostGIS utan inblandning av varken klient eller kartserver. Frågorna ställs mot en vy av våning 28 av linjeobjektstabellen (*Vaningslinje*) direkt i PostGIS. Det framgår tydligt att icke-spatiala SQL-frågor returnerar korrekt svar snabbast. Det är även stor skillnad i svarstiderna beroende på om datamängden är geografiskt refererad eller inte, geografisk refererad datamängd genererar cirka 45% kortare svarstid.

**Tabell 10.7. Svarstiderna för SQL-frågorna direkt mot en vy av våning 28 av linjeobjektstabellen (*Vaningslinje*) i PostGIS. Tabellen visar svarstiderna för både spatiala och icke-spatiala SQL-frågor.**

| **Spatial SQL-fråga** | **Svarstid (ms)** |
|---|---|
| Indexeringsstruktur: | |
| GiST (standard) | 140 |
| Gist (primär indexering) och B-träd | 103 |
| GiST (icke-georefererade) | 243 |
| **Icke-spatial SQL-fråga** | |
| Indexeringsstruktur: | |
| Gist (primär indexering) och B-träd | 70 |
| B-träd | 70 |

Tabell 10.8 visar svarstiderna för visar svarstiderna för SQL-frågorna som utfördes direkt i PostGIS utan inblandning av varken klient eller kartserver. Frågorna ställs mot en vy av våning 299 av linjeobjektstabellen (*Vaningslinje*) direkt i PostGIS.

**Tabell 10.8. Svarstiderna för SQL-frågorna direkt mot en vy av våning 299 av linjeobjektstabellen (*Vaningslinje*) i PostGIS. Tabellen visar svarstiderna för både spatiala och icke-spatiala SQL-frågor.**

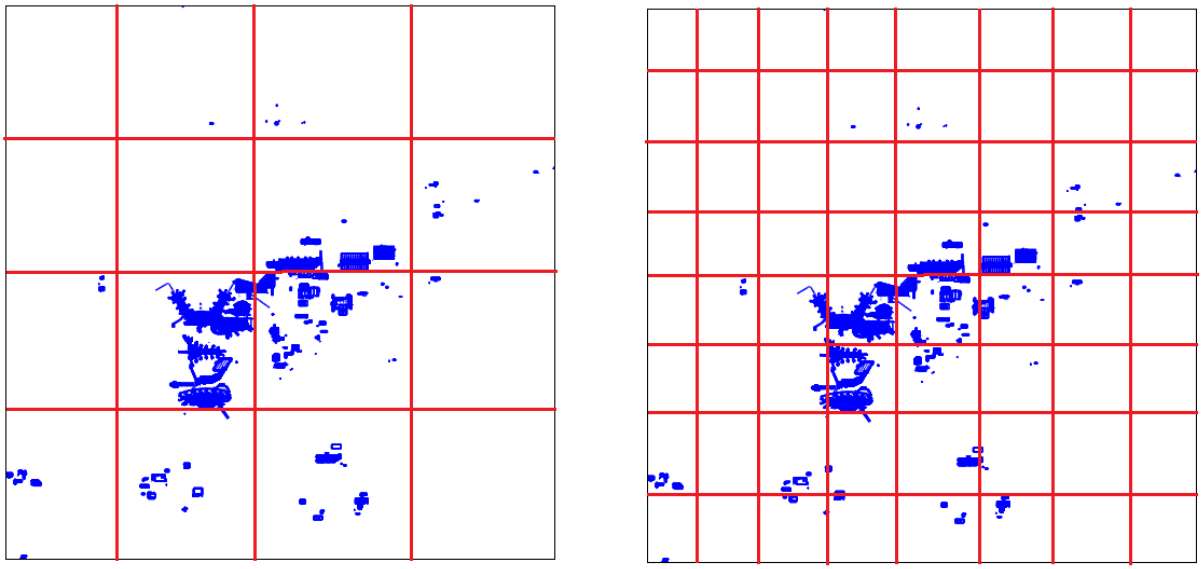
| **Spatial SQL-fråga** | **Svarstid (ms)** |
|---|---|
| Indexeringsstruktur: | |
| GiST (standard) | 39 |
| GiST (primär indexering) och B-träd | 28 |
| GiST (icke-georefererade) | 62 |
| **Icke-spatial SQL-fråga** | |
| Indexeringsstruktur: | |
| GiST (primär indexering) och B-träd | 11 |
| B-träd | 11 |

# 11 Prestandaoptimering SQL Server 2012

SQL Server 2012 tillhandahåller användaren många möjligheter att justera inställningarna för att anpassa dem efter specifika datamängder. Flera olika inställningar kombineras och testas för att uppnå bästa resultat. Varje test genomfördes genom att mäta svarstiden för tio *GetMap*-anrop, medelvärdet redovisas i nedan under resultat.

## 11.1 Metod

Nedan beskriva optimeringsmetoder utförs på de tre olika objekttyperna (punkter, linjer och polygoner). SQL Server 2012 använder den rumsligt baserade indexeringsmetoden *Tesselation* vilken delar in området i kvadratiska celler (4x4, 8x8 eller 16x16). Figur 11.1 visar indelningen av Arlandaområdet för 4x4 och 8x8 celler. Som synes är fördelningen av data väldigt skev och centrerad till vissa av cellerna vilket försvårar inställningarna av indexeringen.



**Figur 11.1. Visualisering av celldensiteten på indexeringsnivå 1 *låg* (4x4) och *medel* (8x8) över Arlanda-området.**

För samtliga testade indexeringsinställningar mäts även indexeringens effektivitet för linje- och polygonobjektstabellerna med hjälp av SQL Server 2012 funktion *sp_help_spatial_geometry_index*.

### 11.1.1 Standard indexering

Indexeringsmetoden (*Tessellation scheme*) vid standardinställningar är *Geometry auto grid* vilken anger indexeringsnivå 1 till *hög* (16x16 celler) och resterande sju nivåer till *låg* (4x4 celler) (Beauchemin 2010). Antalet celler per objekt är enligt standard 16. Den minsta geografiska utbredningen som används för indexeringen beräknas i GeoServer (*Compute from data*).

### 11.1.2 Celldensitet

Vid manuellt skapad indexering anger användaren själv celldensiteten för de fyra indexeringsnivåerna. Celldensiteten som testas för samtliga objekttyper är *hög* (16x16), *medel* (8x8) och *låg* (4x4) uniformt på samtliga indexeringsnivåer. Enligt teoristudien ska det vara fördelaktigt med hög celldensitet vid kompakt datamängd. Indata i aktuellt fall måste ses som gles, dock är den skevt fördelad och väldigt kompakt inom vissa områden, se Figur 11.1. Detta gör val av celldensitet mer komplicerat.

Enligt tidigare studier (Jackson 2011) är det fördelaktigt att använda lika stort totalt antal celler som högsta spatiala objektdensiteten. Därför testas även celldensitets-kombinationen *medel*, *låg*, *låg* och *låg* (*MLLL*) för samtliga objekttyper. Antal celler per objekt vid samtliga tester är standardvärdet 16.

### 11.1.3 Antal celler per objekt

Antalet celler per objekt är enligt standard 16. Värdet är dels beroende av objektens utbredning i rummet och dels celldensiteten. Är celldensiteten låg blir antalet berörda celler per automatik lägre än om celldensiteten är hög (förutsatt samma datamängd). Antalet celler per objekt påverkas även av objekttyp. Punktobjekt t.ex. kan endast tillhöra en cell per indexeringsnivå. Polygonobjekt har generellt större utbredning än linjeobjekt och därmed större antal celler per objekt. Prestandatesterna utförs på indexeringsstruktur *MLLL* celldensitet. *MLLL* används eftersom den uppvisade bäst resultat vid tidigare prestandatester.

Antal celler per objekt är obetydligt för punktobjekt, så länge det överstiger antalet indexeringsnivåer (4), därför testas inte denna parameter för punktobjekt. För linje- och polygonobjekt testas 4, 16, 32, 128, 256 och 8192 celler per objekt.

### 11.1.4 Spatialt anpassad primärnyckel

Tidigare studier (Katibah m.fl. 2012) påvisar förbättrad indexeringsprestanda för punktobjekt om primärnyckeln är spatialt anpassad. Det genom att skapa en sammanslagen primärnyckel (eng. *Clustered Primary Key*) av punktens koordinater och en unik identifierare. Denna teori testas i första hand på punkttabellen men försök görs även på linje- och polygonobjektstabellerna. Den spatiala indexeringen skapas med celldensitet *MLLL* och 256 celler per objekt.

För linjeobjekt skapas en tom tabell med nya kolumner för x- och y-koordinater, se Figur 11.2. Funktionen som beräknar mittpunkten beräknar den numeriska mittpunkten och inte den geografiska mittpunkten. De två nya kolumnerna används för att generera en sammanslagen primärnyckel tillsammans med id (för att garantera en unik identifierare).

```
create table vaningslinje_georef_MLLL_256 (
id int not null,
vaning_id int not null,
geometri geometry not null,
x as geometri.STPointN(geometri.STNumPoints()/2).STX  persisted not null,
y as geometri.STPointN(geometri.STNumPoints()/2).STY  persisted not null,
Primary key clustered( x asc, y asc, id asc));
```

**Figur 11.2. SQL-kod som används för att skapa en ny tabell med x- och y-koordinater för linjeobjekts mittpunkt.**

Därefter fylls tabellen med data från våningslinjetabellen. Eftersom x-och y-kolumnerna är beräknade värden från geometrikolumnen kan den nya tabellen fyllas direkt från våningslinjetabellen. Även fast den nya tabellen har fem kolumner medan linjeobjekts-tabellen endast har tre.

Genereringen av en sammanslagen primärnyckel för polygonobjektens sker analogt fast funktionen *STCentroid()* används för att beräkna mittpunkten istället. Punktobjekt sker analogt och har bara ett koordinatpar varav tillvägagångssättet blir väldigt enkelt och beskrivs inte här.

### 11.1.5 Primär spatial indexering med sekundär icke-spatial indexering

Primär spatial indexering kombineras med en sekundär icke-spatial indexering. Anledningen är att avgöra om genomsökningen efter den spatiala gallringen blir snabbare med en extra indexering. För samtliga objekttyper används den spatiala indexeringsstrukturen som hittills gett bäst resultat som primär indexering. Den sekundära indexeringen skapas utifrån objektens våningstillhörighet (*vaning_id*).

### 11.1.6 Indexeringsstruktur effektivitet

För samtliga testade indexeringsinställningar mäts även indexeringens effektivitet med hjälp av SQL Server 2012 funktion *sp_help_spatial_geometry_index*. Funktionen beräknar indexeringens effektivitet genom att testa vald datamängd (tabell) mot en *intersects*-liknande spatial fråga. Indexeringsstrukturerna testas genom att använda den aktuella vånings minsta geografiska utbredning som polygon som indata till *sp_help_spatial_geometry_index*.

*sp_help_spatial_geometry_index*-funktionen returnerar många mer eller mindre intressanta värden. De som är av intresse för att avgöra indexeringens effektivitet i aktuellt scenario är indexeringens primära filter effektivitet (*Primary Filter Efficency*) (*PFE*), det interna filtrets effektivitet (*Internal Filter Effiency*) (*IFE*) och andel rader i primärfiltret som anropas i det interna filtret (*Percentage of Primary Filter Rows Selected by Internal filter*) (*PoP*), se Ekvation 11.1, Ekvation 11.2 och Ekvation 11.3.

$$PFE\ (\%) = \frac{antal\ korrekta\ objekt}{antal\ objekt\ selekterade\ av\ primärfilter} \qquad \text{Ekvation 11.1}$$

$$IFE\ (\%) = \frac{antal\ objekt\ selekterade\ av\ inre\ filter}{antal\ korrekta\ objekt} \qquad \text{Ekvation 11.2}$$

$$PoP\ (\%) = \frac{antal\ objekt\ selekterade\ av\ inre\ filter}{antal\ objekt\ selekerade\ av\ primärfilter} \qquad \text{Ekvation 11.3}$$

### 11.1.7 SQL-frågor direkt i SQL Server

Detta test utförs genom att isolera databasen från klient och kartserver, därmed tydligt avgöra vilken inverkan inställningarna för de olika spatiala indexeringsmetoderna har. Endast linjeobjektstabellen (*Vaningslinje*) har tillräckligt många objekt för att denna analys ska vara aktuell. Därför utförs testerna endast på denna tabell.

Tillämpningen av Swedavias GIS är att visualisera byggnadsobjekt för hela våningsplan. Samtliga objekt har en våningstillhörighet (attributet *vaning_id*) varpå tester utförs för att undersöka om det är snabbare att hitta alla sökta objekt utan att generera spatiala SQL-frågor (vilket många kartservrar automatiskt gör). D.v.s. Svarstiderna jämförs för SQL-anrop direkt i databasen vid hämtning av en vånings samtliga linjeobjekt. Uppvisad bästa spatiala indexeringsmetod jämförs mot icke-spatial fråga och indexering.

## 11.2 Resultat

I det här avsnittet presenteras resultatet för de prestandaoptimerade åtgärderna som har utförts i SQL Server 2012. Resultaten redovisas per objekttyp. Resultatet av SQL-frågorna exekverade direkt i SQL Server redovisas sist i kapitlet.

### 11.2.1 Punktobjekt

Tabell 11.1 visar medelvärdet för svarstiderna för samtliga prestandaoptimeradeförsök utförda på en vy skapad över våning 130 från den geografiskt refererade punktobjekts-tabellen (*Texterbygg*) i SQL Server 2012. Indexeringsstrukturen *MLLL* som är utvecklad efter objektens spatiala densitet uppvisar bäst prestanda.

**Tabell 11.1 Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 130 från punktobjektstabellen (*Texterbygg*).**

| **Parameterinställning** | **Medelvärde (ms)** |
|---|---|
| Standard | 91 |
| **Celldensitet:** | |
| *Hög* (16x16) | 93 |
| *Medel* (8x8) | 94 |
| *Låg* (4x4) | 88 |
| *MLLL* ((8x8)(4x4)(4x4)(4x4)) | 84 |
| **Övriga tester:** | |
| Sekundär icke-spatial indexering | 93 |
| Spatialt anpassad primärnyckel (*MLLL*) | 94 |

Tabell 11.2 visar medelvärdet för svarstiderna för samtliga prestandaoptimeradeförsök utförda på en vy skapad över våning 148 från den geografiskt refererade punktobjekts-tabellen (*Texterbygg*) i SQL Server 2012.

**Tabell 11.2. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 148 från punktobjektstabellen (*Texterbygg*).**

| **Parameterinställning** | **Medelvärde (ms)** |
|---|---|
| Standard | 56 |
| **Celldensitet:** | |
| *Hög* (16x16) | 55 |
| *Medel* (8x8) | 49 |
| *Låg* (4x4) | 47 |
| *MLLL* ((8x8)(4x4)(4x4)(4x4)) | 47 |
| **Övriga tester:** | |
| Sekundär icke-spatial indexering | 47 |
| Spatialt anpassad primärnyckel (*MLLL*) | 47 |

### 11.2.2 Linjeobjekt

Tabell 11.3 visar medelvärdet för svarstiderna för samtliga prestandaoptimeradeförsök utförda på en vy skapad över våning 28 från den geografiskt refererade linjeobjektstabellen (*Vaningslinje*) i SQL Server 2012. Indexeringsstrukturen *MLLL* (oavsett antal celler per objekt) uppvisar mycket kortare svarstider än standard indexeringsstruktur (upp till 18,3% lägre vid hantering av linjeobjekt).

**Tabell 11.3. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 28 från linjeobjektstabellen (*Vaningslinje*).**

| **Parameterinställning** | **Medelvärde (ms)** |
|---|---|
| Standard | 4 001 |
| **Celldensitet:** | |
| *Hög* (16x16) | 3 871 |
| *Medel* (8x8) | 3 596 |
| *Låg* (4x4) | 3 551 |
| *MLLL* ((8x8)(4x4)(4x4)(4x4)) | 3 535 |
| **Celler per objekt:** | |
| 4 | 3 527 |
| 16 | 3 535 |
| 32 | 3 519 |
| 128 | 3 548 |
| 256 | 3 548 |
| 8192 | 3 692 |
| **Övriga tester:** | |
| Spatialt anpassad primärnyckel | 3 970 |
| Sekundär icke-spatial indexering | 3 956 |
| Icke-georefererade (*MLLL*, 256) | 15 560 |

Tabell 11.4 visar medelvärdet för svarstiderna för samtliga prestandaoptimeradeförsök utförda på en vy skapad över våning 299 från den geografiskt refererade linjeobjektstabellen (*Vaningslinje*) i SQL Server 2012.

**Tabell 11.4. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 299 från linjeobjektstabellen (*Vaningslinje*).**

| **Parameterinställning** | **Medelvärde (ms)** |
|---|---|
| Standard | 477 |
| **Celldensitet:** | |
| *Hög* (16x16) | 444 |
| *Medel* (8x8) | 396 |
| *Låg* (4x4) | 382 |
| *MLLL* ((8x8)(4x4)(4x4)(4x4)) | 381 |
| **Celler per objekt:** | |
| 4 | 382 |
| 16 | 381 |
| 32 | 390 |
| 128 | 378 |
| 256 | 379 |
| 8192 | 384 |
| **Övriga tester:** | |
| Spatialt anpassad primärnyckel | 445 |
| Sekundär icke-spatial indexering | 616 |
| Icke-georefererade (*MLLL*, 256) | 987 |

Tabell 11.5 och Tabell 11.6 visar indexeringarnas effektivitet för linjeobjekt. Effektiviteten testas med *sp_help_spatial_geometry_index()*-funktion med polygoner som motsvarar våning 28s och våning 299s minsta geografiska utbredning. Primärfiltrets effektivitet anger hur stor andel korrekta objekt är selekterade av indexeringens första ”urval”. Andel av primärfilteret som selekteras av det inre filtret anger noggrannheten av primärfiltret och det inre filtret. Vid tillämpning av högre värde än 128 för *antal celler per objekt* uppvisar indexeringsstrukturen *MLLL* mycket hög effektivitet vid hantering av linjeobjekt.

**Tabell 11.5. Indexeringsstrukturernas effektivitet beräknat med SQL Servers inbyggda funktion, en polygon motsvarande våning 28 minsta geografiska utredning används. Indexeringsstrukturen beskriver celldensiteten och antal celler per objekt.**

| Indexeringsstruktur | Primärfilter (%) | Inre filter (%) | Andel av primärfilter valt av inre filter (%) |
|---|---|---|---|
| Standard, 16 | 85,8 | 33,5 | 28,7 |
| *Hög* (16x16), 16 | 49,4 | 0 | 0 |
| *Hög* (16x16), 2000 | 97,5 | 90,7 | 88,4 |
| *Medel* (8x8), 16 | 85,8 | 33,5 | 28,7 |
| *Låg* (4x4), 16 | 85,8 | 33,5 | 28,7 |
| *MLLL*, 4 | 66,2 | 0,0 | 0,0 |
| *MLLL*, 16 | 76,2 | 11,1 | 8,5 |
| *MLLL*, 32 | 92,3 | 22,9 | 21,2 |
| *MLLL*, 128 | 92,7 | 93,9 | 87,0 |
| *MLLL*, 256 | 99,3 | 96,8 | 96,1 |
| *MLLL*, 8192 | 99,3 | 97,2 | 96,5 |

**Tabell 11.6. Indexeringsstrukturernas effektivitet beräknat med SQL Servers inbyggda funktion, en polygon motsvarande våning 299 minsta geografiska utredning används. Indexeringsstrukturen beskriver celldensiteten och antal celler per objekt.**

| Indexeringsstruktur | Primärfilter (%) | Inre filter (%) | Andel av primärfilter valt av inre filter (%) |
|---|---|---|---|
| Standard, 16 | 90,3 | 85,2 | 76,9 |
| *Hög* (16x16), 16 | 56,0 | 0,0 | 0,0 |
| *Hög* (16x16), 2000 | 96,7 | 93,3 | 90,2 |
| *Medel* (8x8), 16 | 90,3 | 85,2 | 76,9 |
| *Låg* (4x4), 16 | 90,3 | 85,2 | 76,9 |
| *MLLL*, 4 | 76,3 | 0,0 | 0,0 |
| *MLLL*, 16 | 91,7 | 2,4 | 2,2 |
| *MLLL*, 32 | 86,0 | 38,3 | 33,0 |
| *MLLL*, 128 | 92,6 | 93,0 | 86,1 |
| *MLLL*, 256 | 96,4 | 96,5 | 93,0 |
| *MLLL*, 8192 | 96,4 | 96,5 | 93,0 |

### 11.2.3 Polygonobjekt

Tabell 11.7 visar medelvärdet för svarstiderna för samtliga prestandaoptimeradeförsök utförda på en vy skapad över våning 130 från den geografiskt refererade polygonsobjekts-tabellen (*Rumsyta*) i SQL Server 2012. Återigen har standard indexeringsstruktur längst svarstid och indexeringsstrukturen *MLLL* uppvisar bäst prestanda.

**Tabell 11.7. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 130 från polygonobjektstabellen (*Rumsyta*).**

| **Parameterinställning** | **Medelvärde (ms)** |
|---|---|
| Standard | 136 |
| **Celldensitet:** | |
| *Hög* (16x16) | 126 |
| *Medel* (8x8) | 131 |
| *Låg* (4x4) | 116 |
| *MLLL* ((8x8)(4x4)(4x4)(4x4)) | 119 |
| **Celler per objekt:** | |
| 4 | 121 |
| 16 | 119 |
| 32 | 125 |
| 128 | 119 |
| 256 | 120 |
| 8192 | 117 |
| **Övriga tester:** | |
| Sekundär icke-spatial indexering | 119 |
| Spatialt anpassad primärnyckel | 125 |

Tabell 11.8 visar medelvärdet för svarstiderna för samtliga prestandaoptimeradeförsök utförda på en vy skapad över våning 123 från den geografiskt refererade polygonsobjekts-tabellen (*Rumsyta*) i SQL Server 2012.

**Tabell 11.8. Medelvärdet av svarstiderna från 10 *GetMap*-anrop mot en vy av våning 123 från polygonobjektstabellen (*Rumsyta*).**

| **Parameterinställning** | **Medelvärde (ms)** |
|---|---|
| Standard | 89 |
| **Celldensitet:** | |
| Hög (16x6) | 94 |
| Medel (8x8) | 86 |
| Låg (4x4) | 78 |
| *MLLL* ((8x8)(4x4)(4x4)(4x4)) | 75 |
| **Celler per objekt:** | |
| 4 | 78 |
| 16 | 75 |
| 32 | 80 |
| 128 | 78 |
| 256 | 78 |
| 8192 | 78 |
| **Övriga tester:** | |
| Sekundär icke-spatial indexering | 83 |
| Spatialt anpassad primärnyckel | 78 |

Tabell 11.9 och Tabell 11.10 visar indexeringarnas effektivitet för polygonobjekt. Effektiviteten testas med *sp_help_spatial_geometry_index()*-funktion med polygoner som motsvarar våning 130s och våning 123s minsta geografiska utbredning. Primärfiltrets effektivitet anger hur stor andel korrekta objekt är selekterade av indexeringens första ”urval”. Andel av primärfilteret som selekteras av det inre filtret anger noggrannheten av primärfiltret och det inre filtret. Vid tillämpning av högre värde än 128 för *antal celler per objekt* uppvisar indexeringsstrukturen *MLLL* mycket hög effektivitet vid hantering av linjeobjekt vid alla tre effektivitetsmätningar.

**Tabell 11.9. Indexeringsstrukturernas effektivitet beräknat med SQL Servers inbyggda funktion, en polygon motsvarande våning 130 minsta geografiska utredning används. Indexeringsstrukturen beskriver celldensiteten och antal celler per objekt.**

| Indexeringsstruktur | Primärfilter (%) | Inre filter (%) | Andel av primärfilter valt av inre filter (%) |
|---|---|---|---|
| Auto 16 | 93,7 | 15,0 | 14,1 |
| Hög (16x16), 16 | 64,7 | 0,0 | 0,0 |
| Medel (8x8), 16 | 45,3 | 15,0 | 6,8 |
| Låg (4x4) 16 | 93,7 | 15,0 | 14,0 |
| *MLLL*, 4 | 32,1 | 0,0 | 0,0 |
| *MLLL*, 16 | 89,0 | 46,7 | 41,5 |
| *MLLL*, 32 | 90,2 | 65,3 | 58,9 |
| *MLLL*, 128 | 96,9 | 97,2 | 94,2 |
| *MLLL*, 256 | 99,8 | 97,7 | 97,5 |
| *MLLL*, 8192 | 99,8 | 99,2 | 99,0 |

**Tabell 11.10. Indexeringsstrukturernas effektivitet beräknat med SQL Servers inbyggda funktion, en polygon motsvarande våning 123 minsta geografiska utredning används. Indexeringsstrukturen beskriver celldensiteten och antal celler per objekt.**

| Indexeringsstruktur | Primärfilter (%) | Inre filter (%) | Andel av primärfilter valt av inre filter (%) |
|---|---|---|---|
| Auto, 16 | 93,7 | 15,0 | 14,1 |
| *Hög* (16x16), 16 | 64,7 | 0,0 | 0,0 |
| Medel (8x8), 16 | 45,3 | 15,0 | 6,8 |
| *Låg* (4x4), 16 | 93,7 | 15,0 | 14,0 |
| *MLLL*, 4 | 32,1 | 0,0 | 0,0 |
| *MLLL*, 16 | 89,0 | 46,7 | 41,5 |
| *MLLL*, 32 | 90,2 | 65,3 | 58,9 |
| *MLLL*, 128 | 96,9 | 97,2 | 94,2 |
| *MLLL*, 256 | 99,8 | 97,7 | 97,5 |
| *MLLL*, 8192 | 99,8 | 99,2 | 99,0 |

### 11.3.4 SQL-frågor direkt i SQL Server

Tabell 111.11 visar svarstiderna för SQL-frågorna som utfördes direkt i SQL Server utan inblandning av varken klient eller kartserver. Frågorna ställs mot en vy av våning 28 av linjeobjektstabellen (*Vaningslinje*) direkt i SQL Server. Det framgår att icke-spatiala SQL-frågor returnerar korrekt svar överlägset snabbast. Det är även väldigt stor skillnad i svarstiderna beroende på om datamängden är geografiskt refererad eller inte, speciellt vid hantering av våningsplan bestående av många objekt (t.ex. våning 28).

**Tabell 11.11 Svarstiderna för SQL-frågorna direkt mot en vy av våning 28 av linjeobjektstabellen (*Vaningslinje*) i SQL Server. Tabellen visar svarstiderna för både spatiala och icke-spatiala SQL-frågor.**

| **Spatial SQL-fråga** | **Svarstid (ms)** |
|---|---|
| Indexeringsstruktur: | |
| Standard | 1 318 |
| *MLLL* 256 | 771 |
| *MLLL 256* och B-träd | 1 399 |
| Icke-georefererade (*MLLL*, 256) | 9 866 |
| **Icke-spatial SQL-fråga** | |
| Indexeringsstruktur: | |
| *MLLL* 256 och B-träd | 5 |
| B-träd | 5 |

Tabell 11.12 visar svarstiderna för SQL-frågorna som utfördes direkt i SQL Server utan inblandning av varken klient eller kartserver. Frågorna ställs mot en vy av våning 299 av linjeobjektstabellen (*Vaningslinje*) direkt i SQL Server.

**Tabell 11.12. Svarstiderna för SQL-frågorna direkt mot en vy av våning 299 av linjeobjektstabellen (*Vaningslinje*) i SQL Server. Tabellen visar svarstiderna för både spatiala och icke-spatiala SQL-frågor.**

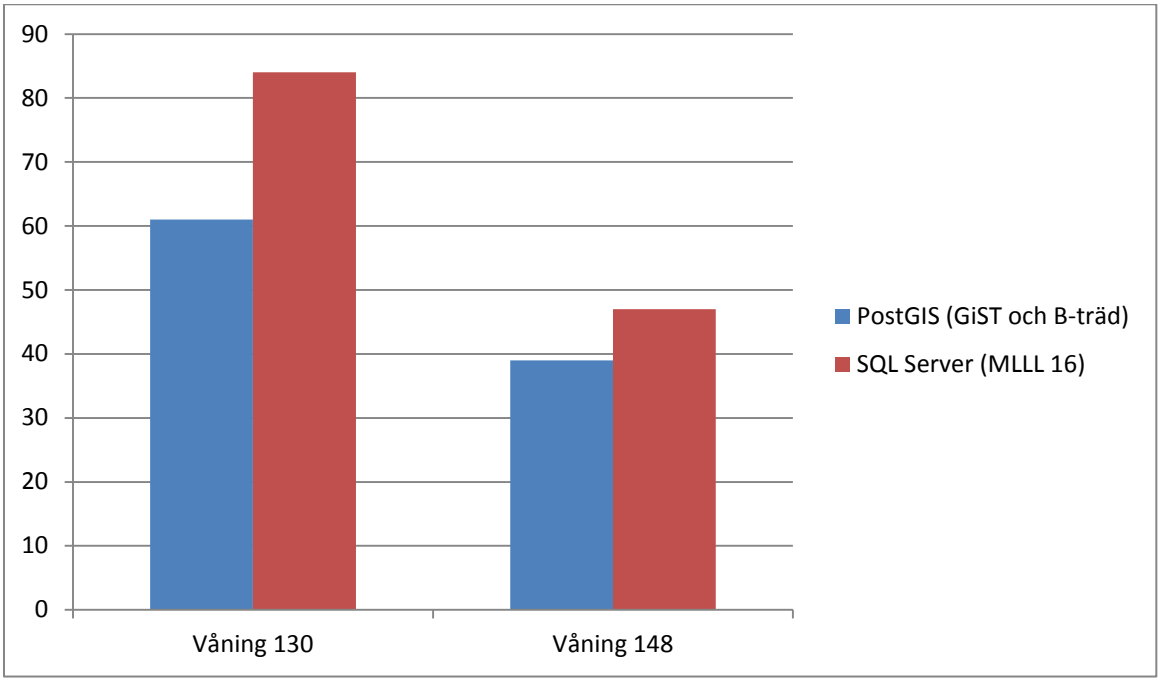
| **Spatial SQL-fråga** | **Svarstid (ms)** |
|---|---|
| Indexeringsstruktur: | |
| Standard | 273 |
| *MLLL* 256 | 141 |
| *MLLL* 256 och B-träd | 357 |
| Icke-georefererade (*MLLL*, 256) | 384 |
| **Icke-spatial SQL-fråga** | |
| Indexeringsstruktur: | |
| *MLLL 256 och B-träd* | 1 |
| B-träd | 1 |

# 12 Sammanfattning av resultat

I det här kapitlet redovisas resultaten av de prestandaoptimerade åtgärderna för de två databasprodukterna. Redovisningen sker separat för de tre objekttyperna. Indexeringsstrukturen som redovisas med tillhörande svarstid är de som har uppvisat bäst prestanda i tidigare test.

## 12.1 Punktobjekt

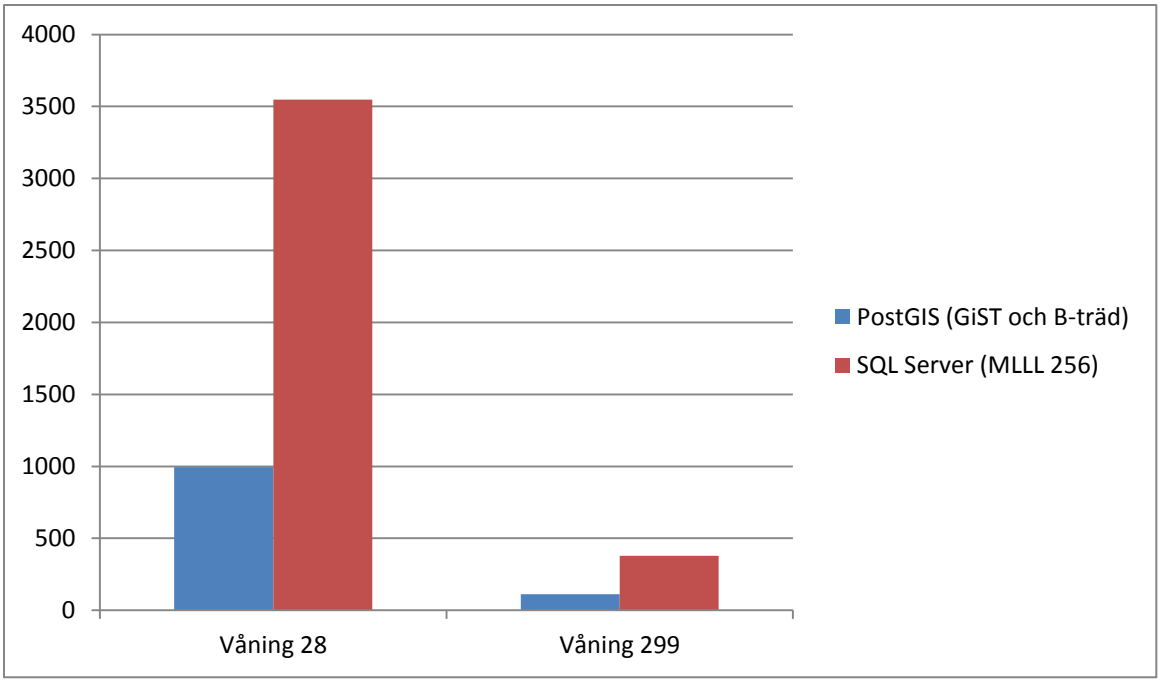
Figur 12.1 visar de lägsta svarstiderna för varje databasprodukt för respektive *GetMap*-anrop mot vy av våning 130 och våning 148 i punktobjektstabellen (*Texterbygg*). Tillämpad indexeringsstruktur anges inom parentesen. PostGIS (blå stapel) uppvisar kortast svarstid och skillnaden är större vid hantering av den större våningen (våning 130). För punktobjekt har PostGIS upp mot 17,0% lägre svarstider.




**Figur 12.1. De lägsta svarstiderna för punktobjektstabellerna för respektive databasprodukt per analyserad våning. Tillämpad indexeringsstruktur anges inom parentesen.**

## 12.2 Linjeobjekt

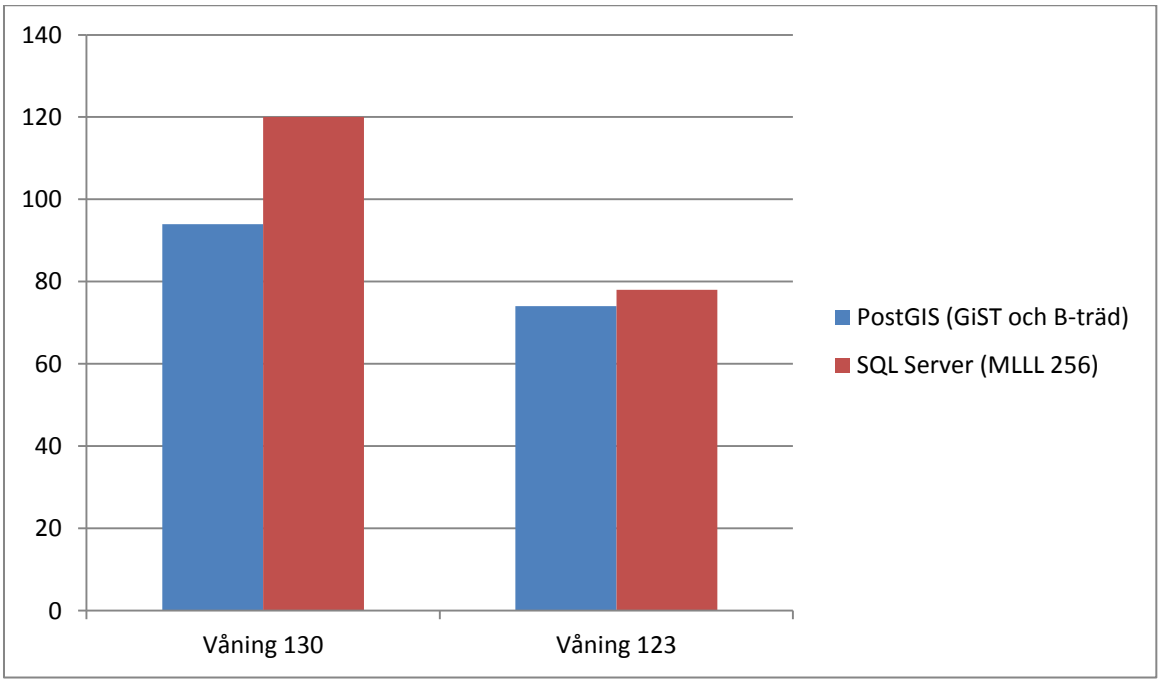
Figur 12.2 visar de lägsta svarstiderna för varje databasprodukt för respektive *GetMap*-anrop mot vy av våning 28 och våning 299 i linjeobjektstabellen (*Vaningslinje*). Tillämpad indexeringsstruktur anges inom parentesen. PostGIS (blå stapel) uppvisar cirka 71,0% kortare svarstid än SQL Server.




**Figur 12.2. De lägsta svarstiderna för linjeobjektstabellen för respektive databasprodukt per analyserad våning. Tillämpad indexeringsstruktur anges inom parentesen.**

## 12.3 Polygonobjekt

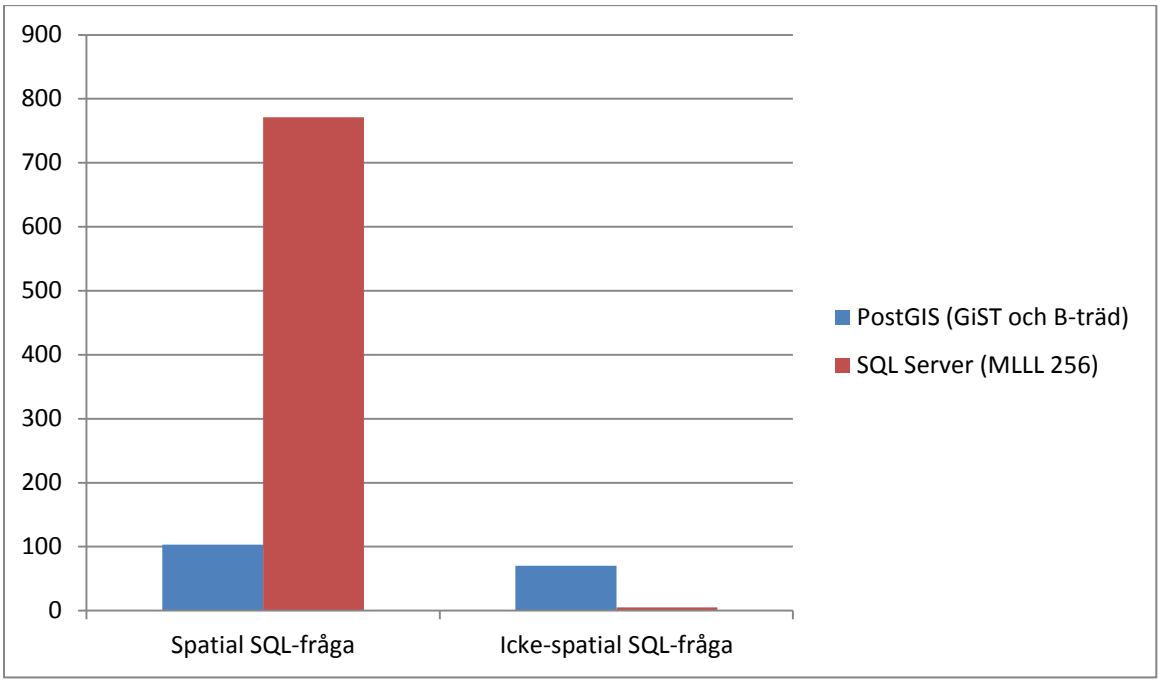
Figur 12.3 visar de lägsta svarstiderna för varje databasprodukt för respektive *GetMap*-anrop mot vy av våning 123 och våning 130 i polygonobjektstabellen (*Rumsyta*). ). Tillämpad indexeringsstruktur anges inom parentesen. PostGIS (blå stapel) uppvisar kortast svarstid och skillnaden är större vid hantering av den större våningen (våning 130). Svarstiderna för PostGIS är cirka 21,7% lägre vid tester av våning 130).



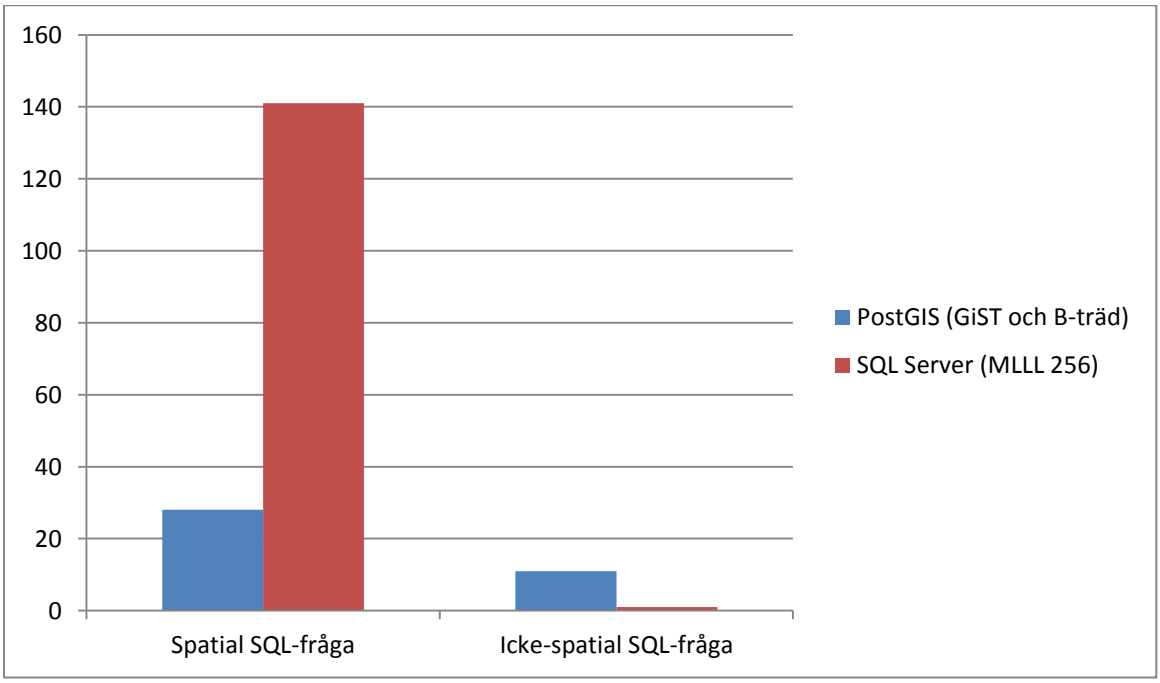

**Figur 12.3. De lägsta svarstiderna för polygonobjektstabellen för respektive databasprodukt per analyserad våning. Tillämpad indexeringsstruktur anges inom parentesen.**

## 12.4 SQL-frågor direkt i databaserna

Figur 12.4 och Figur 12.5 visar svarstiderna för SQL-frågorna som utfördes direkt databaserna utan inblandning av varken klient eller kartserver. Frågorna ställs mot vyer av våning 28 (Figur 12.4) och våning 299 (Figur 12.5) av linjeobjektstabellen (*Vaningslinje*) direkt i databaserna. Det framgår att icke-spatiala SQL-frågor returnerar korrekt svar mycket snabbare än spatiala. Vid hantering av spatiala SQL-frågor är PostGIS (blå stapel) snabbast.




**Figur 12.4. Respektive databasprodukts lägsta svarstider för SQL-frågorna uttryckta direkt mot en vy av våning 28 direkt i databasen. Tillämpad indexeringsmetod anges inom parentesen.**




**Figur 12.5. Respektive databasprodukts lägsta svarstider för SQL-frågorna uttryckta direkt mot en vy av våning 299 direkt i databasen. Tillämpad indexeringsmetod anges inom parentesen.**

# 13 Diskussion

Indata till denna studie var BIM-data i form av CAD-ritningar. Det visade sig i tidigt skede att data i obehandlat format var direkt olämplig att använda i GIS och därmed olämpligt att utföra prestandatester på. Både för tester på hela systemet och isolerade SQL-frågor. Den huvudsakligen anledningen är att samtliga spatiala indexeringsmetoder är beroende av objektens placering i rummet. I ursprungligt format var det aktuella områdets geografiska utbredning enorm samtidigt som majoriteten av objekten var belägna inom små begränsade områden. Särskilt tydligt blev detta vid analys av SQL Server 2012 som tillämpar en rumsligt baserad indexeringsstruktur. Därför utfördes en förenklad geografisk referering. Den geografiska refereringen placerade varje spatialt objekt nära den geografiskt korrekta positionen dock utan någon vridning, vilket för ändamålet ansågs vara tillräckligt. Att geografiskt referera data är även viktigt för att möjliggöra integration med andra system eller kartor.

Geografiskt refererade data hade fortfarande en komplicerad utformning. Det aktuella områdets var nu av en hanterbar storlek och alla mått var uttryckta i meter. Dock var den rumsliga fördelningen fortfarande väldigt skev, stor del av objekten är belägna inom vissa begränsade områden. Dessutom är byggnaderna av varierande storlek, våningarna i det aktuella fallet varierar från 6,6 $m^2$ till ca 20 000 $m^2$. Detta är några av de problem som uppstår vid hantering av byggnadsdata. Det är naturligt att byggnader är olika stora och består av flera våningar samt att byggnaderna är ojämnt fördelande över det aktuella geografiska området. Det aktuella fallet är inte unikt i sin utformning vilket gör det mycket intressant att jämföra hur olika spatiala databasprodukter hanterar situationer som denna.

I denna studie testades och analyserades främst prestandan av hela systemarkitekturen från databas, via kartserver till karttjänst (klient). Prestandan testades i ett förenklat system där *Layer Preview* i GeoServer agerade karttjänst. Fokus lades främst på lagring och hantering av data i två olika spatiala databasprodukter: PostGIS och SQL Server. Dessa två valdes eftersom de är bland de mest använda spatiala databasprodukterna på marknaden samt de tillämpar olika indexeringsmetoder. PostGIS använder GiST-indexering som implementerar R-trädstrukturen. R-träd är en objektbaserad indexeringsstruktur och är därmed mer flexibel än *Tesselation*-algoritmen som är en variant av rumsligt baserad indexeringsstruktur. Även om SQL Server erbjuder användaren många möjligheter till justering av indexerings-parametrarna är den dock underlägsen objektbaserade indexeringsstrukturer i detta och liknande scenarier.

I likhet med tidigare studier (t.ex. Ooi m.fl. 1993) visar prestandatesterna utförda i denna studie att objektbaserade indexeringsstrukturer är mer flexibla och hanterar skevt fördelad data bättre än rumsligt baserade indexeringsstrukturer. Ooi m.fl. konstaterar att en huvudsaklig anledning är att objektbaserade indexeringsstrukturer alltid är uppbyggda av balanserade sökträd medan sökträden för rumsligt baserade indexeringsmetoder riskerar att

bli väldigt obalanserade om datamängden är skevt fördelad i planet. Även testerna Kothuri m.fl. 2002 utför visar att objektbaserade indexeringsstrukturer överlag uppvisar bättre prestanda än rumsligt baserade.

Testerna som utfördes direkt i databaserna exekveras genom att räkna (*count()*-funktion användes) antalet returnerade objekt. Det medförde att frågan endast returnerade en siffra istället för en lista över samtliga objekt, vilket kräver en kortare tidsåtgång. Dock är det inte en perfekt metod eftersom tester visar stor skillnad i tidsåtgång för PostGIS och SQL Server för denna ”enkla” fråga (svarstiden varierar från 1 ms för SQL Server till 11 ms för PostGIS). Testet uppfyller dock sin funktion: visa att det är väldigt stor skillnad i tidsåtgång för spatiala och icke-spatiala frågor. Testerna visar också en tydlig förbättring i svarstider efter indexeringsoptimering. Svarstider i SQL Server minskade från 1 318 ms till 771 ms (våning 28) och från 273 ms till 141 ms (våning 299). Svarstiderna i PostGIS minskade från 140 ms till 103 ms (våning 28) och från 39 ms till 28 ms (våning 299).

Snabbast möjliga indexeringsstruktur för SQL Server uppmätte *primärfiltereffektivitet* på 96,4-99,3 %, *inre filtrets effektivitet* till 96,5-96,8 % och *andel av primärfilter selekterat av det inre filter* 93,0-96,1 % för linjeobjektstabellen (*Vaningslinje*). Motsvarande värden för polygonobjektstabellen (*Rumsyta*)är 99,8 %, 97,5 % och 97,5 %. Jämfört med tidigare studier, t.ex. Beauchemin 2008 är uppvisar denna indexeringsstruktur (*MLLL* 256) väldigt hög effektivitet. Det styrker resonemanget att rumsligt baserade indexeringsstrukturer är sämre än objektbaserade indexeringsstrukturer vid hantering av BIM-data. Eftersom svarstiderna för tester i SQL Server är mycket längre än svarstider för tester utförda i PostGIS (som tillämpar en objektbaserad indexeringsstruktur), trots bra värden för indexeringarnas effektivitet.

En överraskning med SQL Server var att dess standard indexeringsstruktur (*Geometry Auto Grid*) var i stort sett den långsammaste indexeringsmetoden och uppvisade dålig effektivitet. Den låga effektiviteten beror antagligen främst på det låga tillåtna *antalet celler per objekt*, vilket är 16 enligt standard inställningarna. Överstiges denna siffra av något objekt går inte indexeringen djupare i strukturen (för det specifika objektet) vilket resulterar i dålig noggrannhet och effektivitet. Det är förvånande att SQL Server anger en sådan låg gräns för *antal celler per objekt*. Det är inte unikt för just BIM-data att beröra många celler, många tidigare studier har också reflekterat över detta.

PostGIS uppvisade direkt snabba svarstider utan behov av avancerade indexeringin-ställningar som SQL Server. Den huvudsakliga anledningen är att PostGIS tillämpar en objektbaserad indexeringsstruktur. Objektbaserade indexeringsstrukturer är mer dynamiska och anpassningsbara efter objektens placering i rummet. Som teoristudien visar består objektbaserade indexeringsstrukturer alltid av balanserade sökträd, vilket minimerar tidskomplexiteten i de flesta scenarier. En potentiell nackdel med PostGIS kan vara att det är svårt för användaren att justera många indexeringsinställningar. Det blir snabbt väldigt avancerat och behandlar t.ex. hur data är lagrat fysiskt i hårddisken för att optimera

simultan läsning och skrivning av data från hårddisken. Detta faller inte inom ramarna för denna studie och undersöks inte närmare.

En stor fördel med PostGIS är den väl utvecklade frågeoptimerare som finns tillgänglig. Det var bl.a. med hjälp av frågeoptimeraren som problemet med de obehandlade indata uppmärksammades. Oavsett indexeringsinställningar ansåg frågeoptimeraren att det var fördelaktigt att söka efter korrekt data med hjälp av linjärsökning istället för att använda den tillgängliga spatiala indexeringen för våningar med stor geografisk utbredning. PostGIS frågeoptimerare gör det fördelaktigt att skapa flera indexeringsstrukturer till samma datamängd då frågeoptimeraren snabbt kan avgöra vilken indexeringsstruktur som är mest fördelaktig att använda. Lika bra utvecklad är inte SQL Servers frågeoptimerare. I de utförda testerna ökar istället svarstiden vid användning av flera indexeringsstrukturer för samma tabell.

De två spatiala databashanterarna uppvisar störst skillnader i prestanda vid hantering av stora våningar innehållandes många objekt. En anledning kan vara att objektbaserade indexeringsstrukturer är mer robusta och inte lika känsliga för stora datamängder eftersom sökträden som för objektbaserade indexeringsstrukturer är balanserade.

GeoServer användes som kartserver i studien. Kartservrar används i nästan alla situationer vid användning av webbaserade GIS som kommunikationsverktyg och mellanhand mellan klient och databas. Dels översätter den WMS-anrop till SQL-syntax och men framför allt returnerar kartservern den efterfrågade informationen i bildformat. Bildformatet är mer lätthanterligt för klienterna än en samling geografiska objekt. Speciellt i aktuellt fall där flera av de våningarna består av ca 100 000 linjeobjekt.

En väsentlig nackdel med GeoServer (och andra kartservrar) är att det GeoServer alltid översätter klientens WMS-anrop till spatiala SQL-frågor. I många praktiska tillämpningar av GIS är det nödvändigt att formulera spatiala SQL-frågor för att mottaga korrekt svar. Men det finns tillfällen då spatiala frågeställningar inte är nödvändigt och enbart försämrar prestandan hos systemet. I det aktuella fallet har varje spatialt objekt ett våningsattribut och tillämpningen av systemet är att visualisera samtliga objekt för fullständiga våningar. Detta medför att formuleringen av en spatial SQL-fråga är direkt olämpligt eftersom en icke-spatial SQL-fråga som enbart söker objekt med ett visst våningsattribut besvaras många gånger snabbare. Detta syns tydligt i testerna med de isolerade SQL-frågorna.

Trots denna nackdel anses det ändå vara fördelaktigt att använda kartserver som kommunikationsverktyg mellan klient och databas. Främst för att den avlastar klienterna det betungande arbetet att hantera objekten som returneras från databaserna. Det gör att klienterna behöver vara mindre kraftfulla och kan utgöras av standardiserade webbläsare i såväl datorer som surfplattor eller smarttelefoner utan någon from av s.k. *plug-in*.

Det som skiljer denna studie från tidigare är främst att denna studie undersöker prestandan hos ett komplett system med tredelad systemarkitektur. Många tidigare studier analyserar prestandan hos specifika indexeringsalgoritmer vid särskilda frågeställningar. I denna studie utvecklas resultat och lärdomar från tidigare undersökningar för att lösa ett problem som aktualiseras i dagens samhälle, nämligen behovet av inomhuskartor och -navigering Denna studie belyser nödvändiga åtgärder för att systemet ska uppnå acceptabel prestanda samt utgör en grund för vidare utveckling ämnet.

## 13.1 Vidareutveckling

Jag anser att utvecklingen att integrera BIM och GIS med hjälp av spatiala databaser är bra och nödvändig för de två systemen har mycket att lära av varandra. Informationsflödet i världen ökar hela tiden och det är viktigt att utveckla data- hantering och lagring. Integrering av BIM och GIS kan medföra att mängden duplicerade data minskar, informationen behöver inte finnas i båda systemen om det finns ett fungerande system för båda tillämpningarna. För att integreringen av BIM och GIS ska bli en självklarhet behöver i dagsläget visst arbete utföras. Främst handlar det om att upprätta gemensamma standarder, en viktig del är byggnadsritningar behöver bli geografiskt refererade direkt vid skapandet.

Denna studie analyserade främst databasernas roll för den totala prestandan i system med tredelad systemarkitektur. Databaser är en väldigt viktig del i sådan systemarkitektur men det skulle vara intressant att även undersöka kartservers inverkan. Det finns flera kartservrar på marknaden och de har alla olika för- och nackdelar på samma vis som (spatiala) databaser. Särskilt intressant hade det varit att undersöka möjligheten till att formulera icke-spatiala SQL-frågor i kartservern.

Vidare skulle det vara intressant att undersöka vilka egenskaper hos sammasatta spatiala objekt som har störst inverkan på prestandan. T.ex. är det våningens totala geografiska utbredning eller är det våningens antal objekt som har störst inverkan på prestandan? Detta blir av intresse vid upprättande av inomhuskartor som baseras på ritningar. Inomhuskartor ska inte vara lika detaljrika som ritningar och det hade vart intressant att hitta ett riktmärke för hur många objekt olika stora våningar kan innehålla för att ändå uppvisa god prestanda.

# 14 Slutsatser

Studien visar tydligt att det är möjligt att integrera BIM-data i GIS med hjälp av spatiala databaser. Det finns dock flera viktiga aspekter att uppmärksamma för att det ska resultera i system med hög prestanda.

Det har framgått av denna studie att data som ska användas i (webbaserade) GIS med tredelad systemarkitektur behöver vara geografiskt refererade för att funktionellt kunna användas med god prestanda. Främst eftersom samtliga spatiala indexeringsmetoder är beroende av objektens placering i rummet. Även om objektbaserade indexeringsmetoder är mindre känsliga är de fortfarande inte tillräckligt effektiva om för många objekts minsta geografiska utbredning överlappar varandra. Vilket är en överhändande risk om objekten inte är geografiskt refererade.

Önskas systemet användas utan någon koppling till andra system eller omvärlden och med hjälp av kraftfullare klienter är det inte nödvändigt att geografiskt referera objekten. Istället kan systemet utvecklas utan användning av kartserver som mellanhand för att undvika generering av spatiala SQL-frågor.

Av de tester som utförts i studien framgår det att PostGIS är att föredra framför SQL Server. Främst eftersom PostGIS tillämpar en objektbaserad indexeringsmetod som är mycket mer dynamisk och anpassningsbar samt kräver inte lika avancerade inställningar för att uppnå godtagbara svarstider. PostGIS har även en bättre frågeoptimerare som effektivt kan använda egenskapen att den genomsökta tabellen har mer än en indexeringsstruktur för att svara på sammanslagna frågor. Tester utförda på SQL Server visar däremot att genomsökningen blir långsammare om tabellen har flera indexeringsstrukturer när sammanslagna frågor exekveras.

Av studien har det framkommit att kartservrar alltid genererar spatiala SQL-frågor. I de flesta GIS-tillämpningarna är det nödvändigt med spatiala frågeställningen men i aktuellt och liknande scenarier där inget spatialt urval är nödvändigt hade det vart fördelaktigt att kunna generera icke-spatiala SQL-frågor. Dock överväger inte användningen av icke-spatiala SQL-frågor de många fördelar kartservrar bidrar med.

# Referenser


Aref W. och Ilyas I. 2001. *A Framework for Supporting the Class of Space Partitioning Trees*, Department of Computer Science, Purdune University, West Lafayette IN 47907-1398, finns tillgänglig på: http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=2499&context=cstech, hämtat 2013-10-28

Atkinson M., Banchilhon F., DeWitt D., Dittrich K., Maier D. och Zdonik S. 1995. *The Object-Oriented Database System Manifesto*, http://www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto/, hämtat 2013-10-28

Beauchemin B. 2008. SQL Server Blog, http://www.sqlskills.com/blogs/bobb/spatial-index-diagnostic-procs-filter-output/, hämtat 2013-10-07

Beauchemin B. 2010. SQL Server Blog, http://www.sqlskills.com/blogs/bobb/sql-server-denali-the-new-autogrid-spatial-index/, hämtat 2013-10-07

Brinkhoff T. och Kresse W. 2012. Databases s. 61-106, *Springer Handbook of Geographic Information*, red. Kresse W. och Danko D. Springer-Verlag, Berlin, ISBN: 978-3-5403-72678-4 finns tillgänglig på: http://link.springer.com/content/pdf/bfm%3A978-3-540-72680-7%2F1.pdf

BuildingSMART 2013. Industry Foundation Classes (IFC) data model, http://www.buildingsmart.org/standards/ifc/model-industry-foundation-classes-ifc, hämtat 2013-09-30

Cattell R. och Barry D. 2000. *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann Publishers, San Francisco, California, finns tillgänglig på: http://www.xtec.cat/~iguixa/materialsGenerics/ODMG30.pdf, hämtat 2013-09-18

Chawla S. och Shekhar S. 2003. *Spatial Databases - a tour*, Pearson Education Inc, Upper Saddle River, New Jersey 07458, ISBN: 0-13-017480-7

CityGML 2012. What is CityGML, http://www.citygml.org/index.php?id=1523, hämtat 2013-11-24

Elmasri R. och Navathe S. 2011. *Fundaments of Database System*, sjätte upplagan, Addison-Weasley, Boston, ISBN 10: 0-136-08620-9

El-Mekaway M., Östman A. och Shahzad K. 2011. *Towards Interoperating CityGML and IFC Building Models: A Unified Model Based Approach*, Department of Computer and System Sciences, Royal Institute of Technology (KTH), Sverige

GeoServer 2009. What is GeoServer, http://geoserver.org/display/GEOS/What+is+GeoServer, hämtat 2013-09-26

Giampaolo D. 1999. *Practical File System Design*, Morgan Kaufmann Publishers, Inc., San Francisco, California, ISBN: 1-55860-497-9, finns tillgänglig på: http://www.nobius.org/~dbg/practical-file-system-design.pdf, hämtat 2013-10-28

Hellerstein J., Naughton J. och Pfeffer A.1995. *Generalized Search Trees for Database Systems*, finns tillgängligt på: http://db.cs.berkeley.edu//papers/UW-CS-TR-1274.pdf, hämtat 2013-10-28

Hijazi I., Ehlers M., Zlatanova S. och Isikdag U. 2009. *IFC to CityGML Transformation Framework for Geo-Analysis: AWater Utility Network Case*, Institute for Geoinformatics and Remote Sensing, University of Osnabruck, Osnabruck, Germany och OTB, Research Institute for Housing, Urban and Mobility Studies, Delft University of Technology, Delft the Netherlands, finns tillgänglig på: http://www.gdmc.nl/publications/2009/IFC_to_CityGML.pdf, hämtat 2013-11-24

IBM 2010. z/OS UNIX file system, http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zconcepts/zconcepts_177.htm, hämtat 2013-09-18

Jackson T. 2011. The Black Art of Spatial Index Tuning in SQL Server, http://boomphisto.blogspot.se/2011/04/black-art-of-spatial-index-tuning-in.html, hämtat: 2013-11-24

Katibah E., Stojic M., Rys M. och Dritsas N. 2012. Tuning Spatial Point Data Queries in SQL Server 2012, http://social.technet.microsoft.com/wiki/contents/articles/9694.tuning-spatial-point-data-queries-in-sql-server-2012.aspx, hämtat 2013-09-22

Kothuri R., Ravada S. och Abugov D. 2002. *Quadtree and R-tree Indexes in Oracle Spatial: A Comparison using GIS Data*, Oracle Corporation, Nashua New Hampshire 03062, finns tillgänglig på: http://www.dpi.inpe.br/cursos/ser303/oracle_r_tree.pdf, hämtat 2013-11-01

Marklogic 2012. MarkLogic Geospatial Search, http://www.marklogic.com/resources/marklogic-geospatial-search/resource_download/datasheets/, hämtat 2013-12-16

Mashable 2013. Mashable, http://mashable.com/2013/08/05/most-used-smartphone-apps/, hämtat 2013-09-05

Microsoft 2013a. Features Supported by the Editions of SQL Server 2012, http://msdn.microsoft.com/en-us/library/cc645993(v=SQL.110).aspx, hämtat 2013-10-02

Microsoft 2013b. SQL Views, http://technet.microsoft.com/en-us/library/aa214068(v=sql.80).aspx, hämtat 2013-10-07

MongoDB 2013a. Sharding Introduction, http://docs.mongodb.org/manual/core/sharding-introduction/, hämtat 2013-12-16

MongoDB 2013b. NoSQL databases Explained, http://www.mongodb.com/learn/nosql, hämtat 2013-12-16

Obe R. och Hsu L. 2011. *PostGIS in Action*, Manning Publications Co., 180 Broad Street, Stamford, Connecticut 06901, ISBN: 9781935182269

ODBMS 2005. Object Data Management Group, http://www.odbms.org/ODMG/, hämtat 2013-09-17

OGC 2006. *OpenGIS Web Map Server Implementation Specification*, Open Geospatial Consortium Inc., version 1.3.0, red. Beaujardiere J., finns tillgänglig på: http://www.opengeospatial.org/standards/wms, hämtat 2013-10-31

OGC 2007. *Styled Layer Descriptor profil of the Web Map Service, Implementation Specification*, version 1.1.0, Open Geospatial Consortium Inc., red. Lupp M., finns tillgänglig på: http://www.opengeospatial.org/standards/sld, hämtat 2013-10-31

OGC 2011. *OpenGIS Implementation Standard for Geographic information – Simple feature access – Part 1: Common architecture*, version 1.2.1, red. Herring J., finns tillgänglig på: http://www.opengeospatial.org/standards/sfa, hämtat 2013-10-28

OGC 2012. *OGC City Geography Markup Language (CityGML) Encoding Standard*, version 2.0, Open Geospatial Consortium, red. Gröger G., Kolbe T., Nagel C. och Häfele K., finns tillgänglig på: http://www.opengis.net/spec/citygml/2.0, hämtat 2013-11-25

OGC 2013. About OGC, http://www.opengeospatial.org/ogc, hämtat 2013-09-04

OMG 2013. Object Management Group, http://www.omg.org/gettingstarted/gettingstartedindex.htm, hämtat 2013-09-03

Ooi B., Sack-Davis R. och Han J. 1993. *Indexing in Spatial Databases*, Dept. Of Inf. Sys. And Comp. Sci., National U. of Singapore, Collaborative Inf. Tech. Res. Inst. R.M.I.T. and The U. of Melbourne och School of Computing Simon Fraser U. Canada, finns tillgängligt på: http://www.comp.nus.edu.sg/~ooibc/spatialsurvey.pdf, hämtat 2013-11-24

OpenBIM 2013. Definition BIM, http://www.openbim.se/om_openbim/definition_bim, hämtat 2013-10-28

OpenGEO 2013. Introduction to PostGIS, http://workshops.opengeo.org/postgis-intro/introduction.html, hämtat 2013-09-10

Oracle 2013b. *Oracle Spatial and Graph: Developer's Guide 12c Release 1(12.1)*, primär författare Murray C., Oracle Corporation, 500 Oracle Parkway, Redwood Shores, California 94065, http://docs.oracle.com/cd/E16655_01/appdev.121/e17896.pdf, hämtat 2013-10-28

pgSphere 2013. pgSphere 1.1, http://pgsphere.projects.pgfoundry.org/index.html, hämtat 2013-09-10

PostGIS 2013. *PostGIS 2.1.1 dev Manual*, The PostGIS Development Group, finns tillgänglig på: http://postgis.net/stuff/postgis-2.1.1dev.pdf, hämtat: 2013-10-28

PostgreSQL 2013. *PostgreSQL 9.2.4 Documentation*, version 9.2.4, The PostgreSQL Global Development Group, finns tillgänglig på: http://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-A4.pdf, hämtat 2013-10-28

Rigaux P., Scholl M. och Voisard A. 2002. *Spatial Databases with application to GIS*, Morgan Kaufmann Publishers, 340 Pine Street, San Francisco, California 94104-3205, ISBN: 1-55860-588-6

Technet 2013a. Spatial Indexing Overview, http://technet.microsoft.com/en-us/library/bb895265.aspx, hämtat 2013-09-06

Technet 2013b. Common Language Runtime, http://technet.microsoft.com/en-us/library/ms131102.aspx, hämtat 2013-09-10

Technet 2013c. Spatial Data Types Overview, http://technet.microsoft.com/en-us/library/bb964711.aspx, hämtat 2013-09-10

Technet 2013d. Arguments and Properties of Spatial Index Stored Procedures, http://technet.microsoft.com/en-us/library/cc627425.aspx, hämtat 2013-11-08

Wikipedia 2013a. SQL, http://en.wikipedia.org/wiki/SQL#cite_note-ISO.2FIEC-12, hämtat 2013-08-22

Wikipedia 2013b. Inode, http://en.wikipedia.org/wiki/Inode, hämtat 2013-09-18

Wikipedia 2013c. Ascii, http://en.wikipedia.org/wiki/ASCII hämtat 2013-09-18