UPPSALA
UNIVERSITET

UPTEC STS 26026
Examensarbete 30 hp
Juni 2026

MySQL och MongoDB i regional ärendehantering

En jämförelse av prestanda, skalbarhet och datamodell

Tilde Adler

Civilingenjörsprogrammet i system i teknik och samhälle

UPPSALA
UNIVERSITET

MySQL and MongoDB in the Context of Regional Case Management: A Comparison of Performance, Scalability and Data Model

Tilde Adler

Abstract

This study compares MySQL and MongoDB regarding performance, scalability, and data model in the context of regional case management. The study was conducted as a comparative experiment using an AI-generated synthetic dataset based on metadata and data volumes from public web registers. The dataset was tested in six sizes, ranging from 21 000 to 790 000 cases, using nine context specific user stories translated into functionally equivalent queries.

The results show that MongoDB had lower execution times for writing operations, while MySQL had lower execution times for simple lookups, filtering, range searches, aggregation and relation-based operations. The study shows that the choice of database management system should be based on query type, data model, and user context. Under the conditions of this study, MySQL appears to be the more suitable choice for regional case management systems where lookups, filtering and case follow-up are central functions.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Anna-Karin Gärdin Ämnesgranskare: Doghonay Arjmand

Examinator: Elísabet Andrésdóttir

Sammanfattning

Ärendehanteringssystem förekommer inom flera sektorer och verksamheter. Offentlig sektor använder ofta ärendehanteringssystem för att lagra inkomna ärenden från medborgare och företag. I systemen görs dagligen sökningar, insättningar och uppdateringar av ärendena som lagras i systemets databas. Idag finns det ingen reglering för hur länge denna data ska lagras och det blir därför en viktig faktor i valet av databashanteringssystem. Datamängden kan komma att påverka prestandan i systemet och en växande datamängd kan ge oväntade utgifter om databasen behöver skalas upp.

Syftet med denna studie var att undersöka hur databashanteringssystemen MySQL och MongoDB skiljer sig åt avseende exekveringstid, skalbarhet och datamodell i kontexten av ett ärendehanteringssystem. I studien användes ett AI-genererat dataset i ett jämförande experiment. Datasetet utformades för att efterlikna strukturen i ett verkligt dataset i ett ärendehanteringssystem, baserat på metadata och datamängder från offentliga webbdiarier. För att undersöka hur databashanteringssystemen påverkades av växande datamängder användes sex olika datasetstorlekar, från 21 000 till 790 000 ärenden. Det gjorde det möjligt att analysera både exekveringstid vid enskilda datamängder och förändringsmönster när datamängden ökade.

Databashanteringssystemen testades med hjälp av nio scenarier kopplade till verksamhetens behov, vilka översattes till databasfrågor i MySQL och MongoDB. Databasfrågorna omfattade enkla uppslag, filtrering, intervallsökning, aggregering, relationsbaserade operationer samt skrivoperationer. Exekveringstiderna mättes vid samtliga datasetstorlekar och databasfrågor, vilket gav underlag för att jämföra prestanda och skalbarhet mellan databashanteringssystemen.

Resultaten visade att MongoDB hade lägre exekveringstider vid studiens skrivoperationer, medan MySQL hade lägre exekveringstider vid enkla uppslag, filtrering, intervallsökning, aggregering och relationsbaserade operationer. Resultaten visar därmed att databashanteringssystemens lämplighet inte endast kan baseras på övergripande prestanda, utan behöver förstås baserat på querytyp, datamodell och användningsområde. I ett regionalt ärendehanteringssystem, där sökning, filtrering och uppföljning av ärenden är centrala funktioner, framstår MySQL som det mest lämpliga alternativet för de databasfrågor som undersöktes i studien.

Resultaten bör därför främst förstås baserat på de testförutsättningar som användes i studien, snarare än som generella resultat för alla ärendehanteringssystem. Studien bidrar därför med kunskap kring hur MySQL och MongoDB skiljer sig åt avseende exekveringstid, prestanda och skalbarhet vid olika typer av databasoperationer baserade på användning i ett ärendehanteringssystem.

Innehållsförteckning

Sammanfattning **0**
Innehållsförteckning **1**
1. Inledning **3**
1.1 Syfte och frågeställningar 4
1.2 Avgränsning 4
2. Bakgrund **5**
2.1 Databassystem och centrala begrepp 5
2.1.1 Queries 5
2.1.2 Relationsdatabasen MySQL 5
2.1.3 NoSQL-databasen MongoDB 6
2.1.4 Transaktioner och ACID 7
2.2 Ärendehanteringssystem och offentlig kontext 8
2.3 Tidigare studier 8
2.3.1 Prestanda och svarstid 8
2.3.2 Skalbarhet, konsistens och transaktioner 9
2.3.3 Datamodell och systemkontext 10
2.3.4 Dataset och experimentella upplägg 10
2.3.5 Identifierad kunskapslucka 11
3. Metod **12**
3.1 Narrativ litteraturgenomgång 12
3.2 Experimentell design 12
3.2.1 Generering av testdata 13
3.2.2 Utformning av queries 17
3.3 Genomförande av jämförelsestudie 19
4. Resultat **21**
4.1 Enkla uppslag 21
4.2 Filtrering 22
4.3 Intervallsökning 24
4.4 Aggregering 25
4.5 Relationsbaserade operationer 26
4.6 Skrivoperationer 27
4.7 Sammanfattning av resultat från observationsstudie 30
4.7.1 Övergripande prestandamönster 30
4.7.2 Exekveringstid vid ökande datamängder 31
4.7.3 Skillnader mellan querytyper och resurskrävande operationer 31
4.7.4 Variation och stabilitet 31

5. Diskussion **34**
5.1 Resultatdiskussion 34
5.1.1 Prestanda och querytyper 34
5.1.2 Exekveringstid vid ökande datamängder 35
5.1.3 Datamodellens påverkan 36
5.1.4 Transaktionsegenskaper och konsistenskrav 36
5.1.5 Praktisk användbarhet i regional kontext 37
5.2 Metoddiskussion 38
5.2.1 Syntetiskt dataset och AI-genererade data 38
5.2.2 Begränsningar i testmiljö och implementation 39
5.2.3 Queryval och verksamhetskoppling 39
5.2.4 Tillförlitlighet, jämförbarhet och begränsningar 40
6. Slutsatser **41**
6.1 Skillnader i exekveringstid 41
6.2 Resultatens betydelse i en regional ärendehanteringskontext 41
6.3 Vidare forskning 42
Referenser **43**
Appendix A – Slutgiltig prompt för generering av syntetiskt testdataset **47**
Appendix B – Benchmarkqueries och testskript **49**
B.1 Enkla uppslag 49
B.2 Filtrering 50
B.3 Intervallsökning 53
B.4 Aggregering 55
B.5 Relationsbaserade operationer 57
B.6 Skrivoperationer 59

1. Inledning

Offentlig sektor hanterar idag stora mängder digital information och är beroende av IT-system för registrering, lagring och uppföljning av verksamhetsrelaterade data. I regionala ärendehanteringssystem registreras dagligen nya ärenden, handlingar och statusuppdateringar, vilket medför att datamängden kontinuerligt växer över tid. Det finns idag ingen lag eller förordning som reglerar hur länge en region ska lagra data i sina system [1]. Tidigare forskning visar att växande datamängder kan medföra längre exekveringstider och försämrad prestanda [2]. Försämrad prestanda och längre exekveringstider kan leda till stress och frustration hos arkivarien, och försvårar därmed interaktionen med ärendehanteringssystemet [3]. Ärendehanteringssystem i offentlig sektor hanterar bland annat avtal, politiska beslut och felanmälningar [4]. Dessa handlingar är offentliga och väsentliga för att ge invånare insyn i statens verksamhet [5].

Varje dag inkommer flera ärenden till en region [6]. Dessa ärenden kommer bland annat från medborgare, kommun- eller regionanställda samt företag verksamma i regionen och lagras i ett eller flera ärendehanteringssystem [7]. Som i majoriteten av moderna IT-system lagras denna information i databaser. I allmänhet skiljer man ofta mellan två huvudsakliga databastyper: SQL- och NoSQL-databaser [8]. Den huvudsakliga skillnaden mellan dessa är hur datamodellen är uppbyggd. SQL-databaser bygger på tabeller som kopplas samman med länkade kolumner, medan NoSQL-databaser lagrar data i dokument. SQL-databasers största styrkor är höga hämtningshastigheter och tillförlitliga data. NoSQL-databaser erbjuder en flexibel datamodell och möjlighet till billigare uppskalning av systemet än SQL-databaser [9].

Vid utveckling av IT-system behöver både funktionella och icke-funktionella krav beaktas [10]. Funktionella krav beskriver vilka funktioner ett system ska erbjuda och icke-funktionella krav specificerar hur systemet ska fungera i praktiken, exempelvis avseende prestanda, skalbarhet och tillgänglighet. Valet av databashanteringssystem kan därmed få stor påverkan på hur väl ett system uppfyller dessa krav [10]. Om ett system byggs upp i ett olämpligt databassystem kan det leda till bland annat onödiga licenskostnader och dyr uppskalning, vilket kan vara förödande i en dålig ekonomi.

Idag finns det över 380 databashanteringssystem som erbjuder olika sätt att strukturera och hantera data beroende på systemets behov [11]. Faktorer som förväntad datamängd, framtida skalbarhet och hur data kommer att användas, påverkar vilken databastyp som är mest lämplig. I vissa system sker stora mängder sökningar, uppdateringar och transaktioner kontinuerligt, medan andra främst används för långsiktig lagring av information. Valet av databastyp kan därmed påverka både systemets prestanda, skalbarhet och långsiktiga användbarhet. Forskare i tidigare studier är inte eniga kring vilken databas som presterar bäst och därför finns det inget självklart val för alla IT-system.

Mot denna bakgrund undersöker studien hur MySQL och MongoDB skiljer sig åt gällande prestanda, skalbarhet och datamodell i kontexten av ett regionalt

ärendehanteringssystem. Bland tidigare forskning saknas ofta en verksamhets- eller användningsförankring. En jämförelsestudie mellan två av de mest använda databashanterarna, MySQL och MongoDB, i kontexten av ett ärendehanteringssystem kan hjälpa till att fylla en del av den kunskapslucka som identifierats.

1.1 Syfte och frågeställningar

Syftet med arbetet är att skapa en förståelse för vilken databashanteringslösning som bedöms vara mest lämplig, med det testupplägg som användes, för ett regionalt ärendehanteringssystem. Genom en jämförelsestudie av MySQL och MongoDB med ett AI-genererat, verklighetsbaserat testdataset besvaras följande forskningsfrågor:

1) Vilka skillnader i exekveringstid kan identifieras mellan MySQL och MongoDB vid olika databasoperationer utformade för en regional ärendehanteringskontext?
2) Hur kan resultaten förstås utifrån datamodell, querytyp och krav i ett regionalt ärendehanteringssystem?

1.2 Avgränsning

Studien avgränsas till en regional ärendehanteringskontext eftersom regionala organisationer hanterar stora mängder ärenden kopplade till exempelvis politiska beslut, administration, avtal och medborgarkontakt. Den regionala verksamheten kännetecknas av krav på diarieföring, långsiktig lagring, statusuppföljning, sökbarhet och hantering av sekretessmarkerade uppgifter. Dessa krav påverkar vilka databasoperationer som är viktiga, exempelvis uppslag av diarienummer, filtrering efter nämnd eller status, intervallsökningar, historikhantering och aggregering för uppföljning. Regional ärendehantering används därför i studien som ett verksamhetsnära sammanhang för att utforma och tolka jämförelsen mellan MySQL och MongoDB.

Studien avgränsas till en jämförelse mellan MySQL och MongoDB i kontexten av ett regionalt ärendehanteringssystem med hjälp av ett AI-genererat testdataset. I studien har nio user stories utformats för att avgränsa testningen till troliga verkliga scenarier. Studien omfattar inte lagring av tillhörande dokument och filer kopplade till ärenden eller händelser. Fokus har i stället avgränsats till strukturerade metadata och ärenderelaterad information, då inkludering av dokumenthantering hade inneburit ytterligare komplexitet och förändrat studiens fokus. Resultatet speglar i och med det främst databashantering av strukturerade ärendedata snarare än fullständig dokumenthantering i produktionsmiljöer.

Jämförelsestudien berör inte heller testning av andra databashanteringssystem, distribuerade molnmiljöer eller verkliga produktionsdata. Studien fokuserar på prestanda, datamodell, systemegenskaper och användningsområde, medan aspekter som informationssäkerhet och användargränssnitt inte ingår.

2. Bakgrund

Detta kapitel presenterar den teoretiska bakgrund som är relevant för studien. Inledningsvis introduceras centrala databasteoretiska begrepp, med fokus på relationsdatabaser och dokumentdatabaser. Därefter presenteras ärendehanteringssystemets användningsområde, följt av en genomgång av tidigare forskning inom området.

2.1 Databassystem och centrala begrepp

En databas innehåller en mängd data som är sammanhängande och som beskriver ett område, exempelvis en verksamhet [12, s. 8]. När en databas byggs upp, skapas en datamodell. Modellen innehåller information om objekten som förväntas lagras i databasen och innefattar bland annat datatyper, egenskaper och relationer mellan objekten [13]. Det finns flera olika sätt att strukturera data på i databassystem [12, s. 83]. I denna studie var fokus på relationsdatabaser och dokumentdatabaser, då dessa utgör de två huvudsakliga databasmodellerna som jämförs. I följande avsnitt introduceras begreppet queries, och därefter presenteras relationsdatabaser, som länge har utgjort en väletablerad modell inom datalagring.

2.1.1 Queries

För alla typer av databashanteringssystem används frågor (eng. queries) för att hämta, definiera och ändra data i en databas. Relationsdatabaser benämns ibland bara som SQL-databaser för att SQL är det huvudsakliga programmeringsspråket som används för att interagera med relationsdatabaser. SQL är en förkortning för Structured Query Language [12, s. 29]. SQL-frågor innehåller vanligtvis kommandon som SELECT, INSERT, CREATE, UPDATE och DELETE [14]. MongoDB erbjuder flertalet olika programmeringsspråk för användaren att välja mellan. Exempelvis C#, Java och Python för att nämna några [15].

2.1.2 Relationsdatabasen MySQL

I en relationsdatabas lagras data i tabeller. Dessa tabeller består i sin tur av rader och kolumner. I databassammanhang benämns raderna som tuples och kolumnerna som attribut [12, s. 11]. Den data som lagras i tabellernas tuples utgör en mängd, med det menas att datan inte följer någon specifik ordning [12, s. 83]. Det kan inte förekomma dubbletter av tuples, det vill säga där alla attribut i två tuples har samma värden. I tabellerna är vanligtvis minst ett av attributen en primärnyckel, det innebär att varje tuple i det attributet är unik [12, s. 83]. Ett exempel på detta är att alla ärenden som inkommer i ett ärendehanteringssystem får ett unikt diarienummer kopplat till sig. Utöver primärnycklar används även främmande nycklar, även kallat referensattribut, för att skapa kopplingar mellan tabeller. En främmande nyckel är ett attribut i en tabell som refererar till primärnyckeln i en annan tabell, vilket möjliggör relationer mellan olika datamängder [12, s. 122].

I relationsdatabaser lagras information ofta i flera separata tabeller. Återigen ett exempel i ett ärendehanteringssystem, en tabell med ärenden kopplas samman med en tabell över händelser kopplade till dessa ärenden. I tabellen över händelser finns ett fält för diarienummer, det används som främmande nyckel för att kunna koppla händelser till ett ärende. Figur 1 visar ett lagrat ärende och dess lagrade händelser relationsdatabasen MySQL. Notera att figuren består av två skärmavbilder på två tabeller och därför skiljer sig metadata och kolumnrubriker sig åt.

diarienummer	titel	avsandare_mottagare	registrerat_datum	sekretess	arendestatus
GNS/2218/2025	Planeringsunderlag för patientärende avseende väntetider	MediTech Sverige AB	2025-09-08 00:00:00	0	avslutad

diarienummer	handelsenummer	datum_for_handelse	titel_pa_handelse	avsandare_mottagare
GNS/2218/2025	GNS/2218:1/2025	2025-12-17 00:00:00	Samråd genomfört	MediTech Sverige AB

Figur 1. Överst: Tabell över ärende. Underst: Tabell över händelser kopplade till ärendet.

MySQL är ett relationsdatabashanteringssystem som bygger på den relationella databasmodellen och använder SQL som huvudsakligt frågespråk [12, s. 645]. Systemet utvecklades av det svenska företaget MySQL AB och den första versionen lanserades 1995 [16]. MySQL används i praktiken för att lagra, organisera och hantera strukturerad data i tabeller och är ett av de mest använda databashanteringssystemen för webbapplikationer och informationssystem [16].

2.1.3 NoSQL-databasen MongoDB

Begreppet NoSQL uppstod på 1980-talet, när databaser som inte använde SQL programmerades [12, s. 637]. En NoSQL-databas är inte bara en databas som inte använder SQL som språk, utan även lagrar data utan samma relationslogik som SQL-databaser [12, s. 84]. De fyra vanligaste typerna av NoSQL-databaserna är: nyckelvärdesdatabaser, kolumnbaserade databaser, grafdatabaser och dokumentdatabaser [8]. Denna studie är avgränsas till dokumentdatabaser. I dokumentdatabaser lagras data som dokument, där varje dokument består av ett antal fält. Dessa fält kan innehålla olika typer av data, exempelvis textsträngar, datum, listor och talvärden [17]. Till skillnad från relationsdatabaser är dokumentdatabaser inte bundna till ett fast schema där varje post måste innehålla samma uppsättning fält [17]. Två dokument kan innehålla samma data eftersom varje dokument automatiskt tilldelas ett unikt ID när det skapas [18]. Att lagra ärenden i en dokumentdatabas kan upplevas mer komprimerat än en relationsdatabas. Ärenden kan då istället lagras tillsammans med sina händelser i samma dokument, i så kallade inbäddade dokument [19]. I Figur 2 illustreras samma ärende som i Figur 1, där lagras ärendet tillsammans med sina tillhörande händelser i en array inom samma dokument.

Figur 2. Dokument innehållande ärende och dess händelser

MongoDB är ett dokumentorienterat databashanteringssystem och en NoSQL-databas [12, s. 669]. I MongoDB lagras data som dokument, mer specifikt BSON-dokument. Ett BSON-dokument är ett binärt format baserat på JSON, men med stöd för fler datatyper än traditionella JSON-dokument [18]. Dokumenten organiseras i samlingar (eng. collections), som kan liknas vid tabeller i en relationsdatabas [20]. MongoDB kan användas både som fristående databas och i distribuerade miljöer. I distribuerade miljöer kan data kopieras mellan servrar eller delas upp mellan flera maskiner för att öka tillgänglighet och skalbarhet [21].

2.1.4 Transaktioner och ACID

Transaktioner är operationer i databasen som hör ihop, exempelvis att flytta pengar mellan bankkonton, först tas summan bort från ena bankkontot och sedan läggs samma summa till på det andra bankkontot [12, s. 505–506]. Olika databashanterare har olika transaktionsegenskaper och skiljer sig ofta i hur de prioriterar dessa egenskaper [12, s. 505].

Relationsdatabaser förväntas garantera fyra transaktionsegenskaper: atomicitet, konsistensbevarande (eng. consistency preserving), isolering och hållbarhet (eng. durability), förkortade ACID [12, s. 505]. Atomicitet innebär att transaktioner ska ske omedelbart, antingen genomförs hela transaktionen eller inte alls. Om det skulle uppstå ett avbrott i transaktionen på grund av att användaren ångrar sig eller att strömmen går, så ska databasen ta tillbaka alla ändringar som hunnits genomföras [12, s. 506]. Konsistens betyder att det inte finns några inre motsägelser, all data ska följa integritetsvillkoren och får inte bryta mot några relationsregler [22, s. 9]. Konsistensbevarande betyder att konsistensen hos databasen ska bevaras när en transaktion genomförts. Transaktioner ska ske isolerade, med det menas att när flera transaktioner genomförs samtidigt får de inte se ofullständiga transaktioners ändringar [12, s. 506], transaktionerna ska genomföras som om inga andra transaktioner sker samtidigt [22, s. 9]. Hållbarhet implicerar att genomförda och avslutade transaktioner som gjorts i databasen är sparade och kommer inte att försvinna oavsett om strömmen stängs av eller om lagringsdisken går sönder [12, s. 506].

Dokumentdatabasen MongoDB erbjuder också transaktioner med ACID-egenskaper [23], vilket skiljer från andra NoSQL-databaser som ofta prioriterar skalbarhet och tillgänglighet framför strikt ACID-konsistens [24].

2.2 Ärendehanteringssystem och offentlig kontext

Ett ärendehanteringssystem är ett verktyg för att registrera, lagra och strukturera en verksamhets ärenden och förfrågningar. Inkommande ärenden registreras på samma sätt oavsett om de inkommit via e-post, telefon eller webbformulär, och lagras därefter i systemets databas. En av systemets främsta funktioner är möjligheten att kunna följa upp ärendet och för den som skickat in ärendet att kunna se dess aktuella status [25].

Ärenden och handlingar som inkommer till en myndighet är allmänna handlingar och omfattas av offentlighetsprincipen. Det innebär att vem som helst har rätt att ta del av dem [5]. Vissa handlingar omfattas dock av sekretess, exempelvis när de innehåller uppgifter om enskilda individers ekonomiska situation eller andra känsliga personuppgifter [26]. Det är myndighetens skyldighet att förvara de inkomna handlingarna och det är vanligt att de registreras i ett diarium [6].

2.3 Tidigare studier

Följande avsnitt syftar till att ge en översikt av tidigare forskning kring jämförelser mellan SQL- och NoSQL-databaser, med särskilt fokus på prestanda, skalbarhet, datamodell och metodval. Med utgångspunkt i detta identifieras återkommande jämförelsepunkter samt den kunskapslucka som föreliggande studie ämnar bidra till att fylla.

2.3.1 Prestanda och svarstid

Jämförelser mellan SQL-databaser med NoSQL-databaser är ett väletablerat forskningsområde inom informationsteknologi. Eftersom databastyperna skiljer sig åt i flera avseenden har tidigare studier utgått från olika frågeställningar och jämförelsepunkter. Gemensamt för många av dessa studier är att svarstid vid frågebearbetning utgör en central undersökningspunkt, ofta i kombination med andra prestanda- och systemrelaterade faktorer. Exempelvis undersöker en tidigare studie skillnader i databasernas transaktionsegenskaper, ACID och BASE, samt jämför databaserna utifrån egenskaperna skalbarhet, tillgänglighet, konsistens, hållbarhet och tillförlitlighet [2]. En annan studie fokuserar på effekten av indexering, olika nivåer av konsistens samt dataparallellisering över flera noder [27].

En jämförelsepunkt som förekommer i majoriteten av tidigare forskningsstudier om SQL- och NoSQL-databaser är svarstider. Flera studier undersöker prestandan vid så kallade CRUD-operationer (Create, Read, Update, Delete) [28] ofta vid varierande datamängder. Resultaten visar att NoSQL-databaser ofta uppvisar kortare svarstider vid växande datamängder, särskilt vid läs- och skrivoperationer [2], [27].

Tidigare studier visar motstridiga resultat vid jämförelse av SQL- och NoSQL-databaser med fokus på transaktioner per sekund. Medan en studie visar att NoSQL-databaser uppvisar lägre genomströmning än SQL-databaser vid ökande datamängder [29], visar en annan att NoSQL-databaser hanterar växande datamängder bättre och därmed bibehåller en högre genomströmning [30]. Skillnaderna i resultaten kan bero på experimentmiljö, typ av arbetslast och vald benchmarkmetod. Detta visar att prestandajämförelser mellan databastyper är starkt beroende av kontext och bör tolkas i förhållande till det specifika användningsområdet.

2.3.2 Skalbarhet, konsistens och transaktioner

En av NoSQL-databasers främsta fördelar är möjligheten att kunna skala upp ett system på ett mer effektivt sätt än traditionella SQL-databaser [2]. Tidigare studier påpekar att en central fördel med att skala upp en NoSQL-databas är att det oftast är billigare än att skala upp en SQL-databas. NoSQL-databaser skalas horisontellt vilket innebär att antalet servrar eller noder adderas till systemet. SQL-databaser skalas istället vertikalt, vilket innebär att hårdvaran uppdateras exempelvis med starkare processorer eller större RAM-minne [2], [31]. Horisontell skalning är därför ofta mer kostnadseffektiv, särskilt vid hantering av stora datamängder och distribuerade system [27]. Samtidigt kan vertikal skalning vara ett lämpligt alternativ i sammanhang där datamängden är hanterbar och där prestandabehoven kan mötas genom uppgradering av befintlig hårdvara [2]. Även om detta kan innebära en högre initial kostnad, kan förbättrad processorkraft, minneskapacitet och lagringsprestanda göra SQL-databaser kostnadseffektiva i vissa typer av system. Detta visar att skalbarhet bör förstås i förhållande till systemets datamängd, tillväxttakt och systemets tekniska struktur.

En viktig transaktionsegenskap att överväga vid valet av databastyp är konsistens. Tidigare studier visar att NoSQL-databaser i högre grad prioriterar tillgänglighet till databasen framför att data omedelbart är konsistent [2]. Detta kan innebära att användare inte ser samma version av datasetet [27] vilket kan skapa konsekvenser i system som kräver att samtliga användare har åtkomst till den senast uppdaterade informationen. I NoSQL-databaser blir data eventuellt konsistent [27], vilket möjligen kan vara acceptabelt under en begränsad period, men denna typ av databas lämpar sig främst för system där tillfälliga inkonsekvenser i data är mer tolererbara.

Som tidigare nämnt prioriterar NoSQL-databaser tillgänglighet framför omedelbar konsistens i data. Denna prioritering ställer lägre krav på transaktionshanteringen, vilket i sin tur kan förbättra både prestanda och lagringskapacitet [2].

Tidigare studier lyfter fram skalbarhet som en central faktor vid val av databassystem, särskilt i miljöer där datamängden förväntas öka över tid och där kostnadseffektiv expansion är viktig [2], [27], [31]. Detta innebär att både långsiktig tillväxt och kostnadseffektiv skalning framstår som viktiga organisatoriska faktorer vid val av databastyp.

2.3.3 Datamodell och systemkontext

Tidigare studier lyfter fram att relationsdatabaser lämpar sig väl för strukturerad data [30]. Detta talar för att en SQL-baserad relationsdatabas kan vara mer lämplig än en dokumentbaserad NoSQL-databas i sammanhang där datas struktur är tydligt definierad och följer ett konsekvent schema.

2.3.4 Dataset och experimentella upplägg

I tidigare genomförda studier förekommer olika typer av dataset. En studie är gjord med ett testdataset från samarbetspartnern, en är gjord med verkliga data från en industri, en annan på verkliga anonymiserade medicinska data [2], [27], [31]. I flertalet av studierna diskuteras användning av verkliga data, och forskarna menar att det är viktigt att data som används inte får riskera att skada vare sig samarbetspartnerns eller individernas integritet [29], [30]. Att använda syntetiska data hävdas även vara problematisk för att det kan göra det svårare att hävda ett verklighetstroget resultat [29]. När testdataset finns tillgängliga att tillgå från samarbetspartner eller uppdragsgivare är det rimligt att använda dessa för att kunna dra slutsatser som beslut senare kan grundas på. I en uppsats om testdata och GDPR skriver författaren att testning är avgörande för att skapa en bra och fungerande IT-lösning [32]. Efter antagandet av GDPR har testning av IT-system förändrats, främst för system som behandlar känsliga data som personnummer och kontaktuppgifter.

Detta visar att valet mellan syntetiska och verkliga data innebär en avvägning mellan kontroll och realism. Syntetiska dataset tillåter hög kontroll över struktur, datamängd och testscenarier, medan verkliga dataset i större utsträckning speglar faktiska användningsmönster och därmed ökar studiens externa validitet. Valet av dataset påverkar därmed inte enbart experimentets genomförande utan även hur resultaten kan tolkas. Studier baserade på verkliga dataset kan ge mer realistiska resultat medan syntetiska dataset riskerar att förenkla datas struktur och därmed inte fullt ut återge den komplexitet som återfinns i praktiska informationssystem.

Tidigare studier visar på en stor variation i datasetstorlek, där valen i hög grad styrs av studiens syfte och den aktuella verksamheten som undersöks. I den studie där datasetet tillhandahölls av samarbetspartnern uppgick den ursprungliga datamängden till 6 000 objekt, men denna kopierades och utökades för att skapa fler undersökningspunkter [2]. Den största datamängden i studien var 500 000 objekt, vilket motiverades med att den storleken krävdes för att NoSQL-databaser ska ge ett resultat att analysera. För att ge underlag för ytterligare jämförelser inkluderades även en mellanliggande storlek om 100 000 objekt [2]. I studien som använde anonymiserade sjukhusdata testades betydligt större dataset om 500 000, 5 000 000 och 50 000 000 objekt, vilket motiverades av den kraftiga ökningen av sjukhusdata under det senaste decenniet [31]. I studien med ett industriellt dataset låg fokus i stället på skalbarhet, där datasetets storlek angavs i gigabyte, och begränsades av testmaskinens tillgängliga RAM-minne [27].

Detta tyder på att datasetstorleken inte enbart är en teknisk parameter, utan även en central del av metoden som påverkar vilka slutsatser som kan dras om prestanda och skalbarhet. Mindre dataset används för grundläggande jämförelser av funktionalitet och svarstid, medan större dataset är nödvändiga för att synliggöra skillnader i skalbarhet och genomströmning mellan SQL- och NoSQL-databaser. Valet av datasetstorlek behöver därför relateras till det praktiska användningsområdet för att resultaten ska bli relevanta och generaliserbara.

I tidigare studier förekommer både standardiserade benchmarkverktyg, såsom Yahoo Cloud Serving Benchmark (YCSB) [27], [29], och experimentella upplägg framtagna inom ramen för studien för att mäta databasers prestanda [2], [30], [31]. En utmaning vid jämförelser mellan SQL- och NoSQL-databaser är att de bygger på olika arkitekturer, vilket medför att olika benchmarkverktyg ofta används [29]. För SQL-databaser förekommer exempelvis verktyg från Transaction Processing Performance Council (TPC), medan YCSB har utvecklats specifikt för att jämföra moln- och NoSQL-databaser [29]. Ett återkommande experimentellt upplägg i tidigare studier är att forskarna utvecklar en egen applikation tillsammans med scripts för att simulera användningen av en webbapplikation, och därigenom mäta databasernas prestanda [26]. Andra experimentmetoder som identifierats är klientapplikationer kopplade till REST API:er samt mätscript avsedda att testa läs-, skriv- och uppdateringsoperationer [29], [31]. Skillnader i benchmarkupplägg kan sannolikt förklara delar av de motstridiga resultaten i tidigare studier, då vissa använder standardiserade benchmarkverktyg som YCSB medan andra använder egenutvecklade testmiljöer, REST API:er eller webbapplikationer.

2.3.5 Identifierad kunskapslucka

Sammantaget visar tidigare forskning att jämförelser mellan SQL- och NoSQL-databaser huvudsakligen har genomförts i generella eller industriella miljöer. Denna kunskapslucka är särskilt relevant eftersom offentlig sektor kännetecknas av höga krav på datakonsistens, långsiktig informationslagring och integritetshantering, vilket kan påverka hur SQL- respektive NoSQL-databaser presterar i praktiken. Det saknas däremot studier som undersöker databastypernas prestanda i svenska regionala ärendehanteringssystem. Vidare har ingen av de identifierade studierna använt AI-genererade syntetiska dataset, framtagna genom prompt engineering, för att efterlikna verkliga ärendedata. De identifierade studierna har inte heller använt user stories som grund för att utforma verksamhetsnära databasfrågor. Detta tyder på en tydlig vetenskaplig kunskapslucka som föreliggande studie avser att adressera.

3. Metod

Studien har genomförts som en jämförande experimentell studie där MySQL och MongoDB jämförs med samma syntetiska testdataset och nio verksamhetsnära queries. Studien har kompletterats med en narrativ litteraturgenomgång, vars syfte var att skapa en teoretisk förståelse för tidigare forskning om SQL- och NoSQL-databaser samt att identifiera relevanta jämförelsepunkter för den empiriska undersökningen. I detta kapitel redogörs för studiens metoder, praktiska genomförande och de metodmässiga begränsningar som studien innebär.

3.1 Narrativ litteraturgenomgång

För att skapa en teoretisk förståelse för studiens ämnesområde och forskningsfrågor genomfördes en narrativ litteraturgenomgång [33, s. 130]. I denna studie användes litteraturgenomgången för att belysa centrala begrepp, identifiera relevanta jämförelsepunkter mellan SQL- och NoSQL-databaser samt synliggöra den kunskapslucka som studien avser att undersöka [33, s. 131].

Litteraturgenomgången genomfördes inte enbart som en beskrivande sammanställning av tidigare studier, utan även med ett kritiskt förhållningssätt. Det innebar att studiernas syften, utgångspunkter, antaganden, styrkor och svagheter beaktades, liksom hur de relaterade till varandra och till denna studies forskningsfrågor [33, s. 131,133].

Urvalet av litteratur styrdes av studiens forskningsfrågor och omfattade forskning om SQL- och NoSQL-databaser, med särskilt fokus på MySQL, MongoDB, prestanda, skalbarhet och datamodellering. Sökningar genomfördes främst i DiVA samt via Uppsala universitetsbiblioteks söktjänster. Exempel på använda sökord och sökfraser var ”databas jämförelsestudie”, ”jämförelsestudie MySQL MongoDB”, ”jämförelsestudie MySQL” och ”jämförelsestudie MongoDB”. De studier som bedömdes som mest relevanta för arbetets syfte valdes ut och sammanställdes tematiskt. Tematiseringen låg sedan till grund för strukturen i uppsatsens bakgrundskapitel, där tidigare forskning organiserades baserat på återkommande teman inom området [33, s. 136,139]

3.2 Experimentell design

I den experimentella jämförelsestudien användes MySQL och MongoDB som databashanteringssystem. Valet att använda MySQL motiverades av att systemet är en av de mest etablerade relationsdatabaserna [11], samt att den tillhandahölls av lärosätet, vilket möjliggjorde god tillgänglighet och en praktisk genomförbar implementation inom ramen för studien. MongoDB valdes eftersom det är en av de mest använda dokumentdatabaserna [11] och därmed utgör ett relevant jämförelseobjekt i förhållande till MySQL. Valet av MongoDB motiverades även av att teknisk kompetens fanns tillgänglig hos det samarbetande företaget, vilket skapade goda förutsättningar för en

tillförlitlig implementation. Tabell 1 redovisar de databashanteringssystem, klientverktyg och den testmiljö som användes i studien.

Tabell 1. Testmiljö och verktyg som användes i studien

Kategori	Specifikation
Databashanteringssystem	MySQL Server 9.6.0; MongoDB Server 7.0.30
Klientverktyg	MySQL Workbench 8.0.46; MongoDB Compass 1.49.6
Testdator	MacBook Pro, 16 tum, 2019
Processor	2,6 GHz 6-Core Intel Core i7
Minne	16 GB 2667 MHz DDR4 RAM
Operativsystem	macOS Tahoe 26.4.1

Samtliga tester genomfördes lokalt på samma dator och under samma nätverksförhållanden. Detta gjordes för att säkerställa jämförbara testförutsättningar mellan databashanteringssystemen och minska risken för att skillnader i testmiljön påverkade resultatet.

3.2.1 Generering av testdata

Att skapa ett användbart, realistiskt och anonymiserat testdataset var en central del av studiens experimentella upplägg, eftersom testdatans struktur i hög grad påverkar jämförelsens relevans och tillförlitlighet mellan databashanteringssystemen [34, s. 5]. För att kunna skapa ett så verklighetsnära testdataset som möjligt användes metadata från ett webbdiarium [35] som utgångspunkt. I denna studie avsåg metadata information om vilka typer av fält och attribut som förekommer i ett regionalt ärendehanteringssystem. I webbdiariet förekom både ärenden och händelser. Alla ärenden hade ett diarienummer, titel, användare/mottagare, registreringsdatum, sekretessmarkering och ärendestatus och alla händelser hade ett händelsenummer, handling, inkommande/utgående, titel, avsändare/mottagare och patientuppgiftslagen [35]. Tabell 2 redogör för den metadata som identifierades i webbdiariet, de olika datatyperna samt exempel på data.

Tabell 2. Metadata från webbdiarium

Metadata	Datatyp	Exempel
Diarienummer	Textsträng	GNS/2218/2025
Titel	Textsträng	Planeringsunderlag för patientärende avseende väntetider
Avsändare/mottagare	Textsträng	MediTech Sverige AB
Registreringsdatum	Datum	2025-09-08
Sekretessmarkering	Boolean	Sant eller falskt
Ärendestatus	Textsträng	Under beredning eller avslutat
Händelsenummer	Textsträng	GNS/2218:1/2025
Handling	PDF	Mötesanteckningar
Inkommande/utgående	Datum	2025-12-17
Titel	Textsträng	Samråd genomfört
Avsändare/mottagare	Textsträng	MediTech Sverige AB
Patientuppgiftslagen	Boolean	Sant eller falskt

För att göra det möjligt att genomföra testning på flera olika datatyper inkluderades majoriteten av de fält som identifierades i webbdiariet, med undantag för handling och patientuppgiftslagen. Fältet handling exkluderades eftersom studien avgränsades till strukturerade metadata och inte omfattade lagring eller hantering av dokumentfiler. Fältet patientuppgiftslagen exkluderades eftersom fältet bedömdes ha en begränsad relevans i förhållande till studiens syfte och den nuvarande regleringen genom GDPR [36]. Eftersom sekretessmarkeringen redan utgjorde ett booleskt fält bedömdes fältet inte heller tillföra någon ytterligare datatyp till testdatasetet.

Valet att utgå från metadata från ett offentligt arkiv motiverades dels av praktiska skäl, dels av metodiska. Jämfört med exempelvis intervjuer med en arkivarie eller systemförvaltare möjliggjorde denna ansats en mer tidseffektiv datainsamling [33, s. 393]. Eftersom det insamlade materialet var offentligt minskade även risken för reaktivitet och bias som annars kan uppstå vid intervjubaserad datainsamling [33, s. 393]. Samtidigt kunde studien genomföras utan att använda produktionsdata, verkliga personuppgifter eller sekretesskänsligt innehåll. I stället användes metadata från ett offentligt webbdiarium som underlag för att generera ett syntetiskt dataset med hjälp av AI. Detta minskade risken för att individer eller organisationer skulle kunna identifieras, samtidigt som datasetet kunde utformas för att efterlikna en realistisk ärendehanteringsstruktur. En begränsning är dock att syntetiska data inte fullt ut kan återspegla komplexiteten i verkliga ärenden. Att utgå från ett verklighetsnära metadataunderlag underlättade därför både genereringen av datasetet och bedömningen av när det syntetiska materialet var tillräckligt realistiskt för studiens syfte.

För att generera det syntetiska testdataset användes den generativa AI-tjänsten ChatGPT, version ChatGPT-5.4 Thinking. Användningen av stora språkmodeller, som ChatGPT, för att skapa syntetisk testdata har beskrivits som ett tids- och resurseffektivt sätt att ta fram testdata [34, s. 5]. I denna studie användes modellen för att generera strukturerade data som efterliknar ett regionalt ärendehanteringssystem utifrån fördefinierade regler och instruktioner.

Genereringen av testdata genomfördes iterativt, vilket innebar att prompten successivt omformulerades och förbättrades tills det genererade datasetet i så hög grad som möjligt efterliknade det verkliga datasetets struktur. Fokus låg främst på diarienumrens format, förekomsten av nämndprefix, datumformat, statusvärden, sekretessmarkeringar samt relationen mellan ärenden och tillhörande händelser. Processen upprepades till datasetet bedömdes vara tillräckligt strukturellt likt det verkliga underlaget för att kunna användas i jämförelsestudien.

Inledningsvis användes promptar som genererade mindre dataset om 100 rader. Syftet med dessa mindre dataset var att fungera som provdata för att utvärdera promptarnas tydlighet och bedöma om instruktionerna tolkades på avsett sätt. Resultaten analyserades därefter genom att det genererade materialet jämfördes med strukturen i det verkliga webbdiarium som låg till grund för studien [35]. När avvikelser eller brister i upptäcktes reviderades promptarna, varefter en ny version av datasetet genererades. De mindre dataseten användes enbart i syfte att utveckla och utvärdera promptarna. Eftersom användning av stora språkmodeller kan ge resultat som inte alltid är korrekta eller konsekventa, granskades varje genererad version kritiskt innan den användes vidare i studien [34, s. 10,13,110]. Dataseten genererades i CSV-format, vilket är ett vanligt textbaserat filformat för tabulär data [37].

Den iterativa utvecklingen av promptarna ledde till flertalet justeringar. Exempelvis infördes regler för sekventiella diarienummer per enhet och år, en större andel företag och kommunala organ som avsändare eller mottagare, samt en andel skyddade poster

som bättre motsvarade webbdiariets struktur. När det genererade datasetet bedömdes överensstämma med strukturen i det verkliga diariet skapades ytterligare ett dataset innehållande händelser kopplade till ärendena i det första datasetet. Datasetet över händelser utformades med främmande nyckel och unikt händelse-ID för att möjliggöra relationer mellan ärenden och tillhörande händelser. Dessa förändringar genomfördes för att successivt öka datasetets realism och förbättra dess användbarhet i jämförelsestudien.

Efter den första pilotgenereringen reviderades prompten iterativt utifrån en jämförelse mellan det genererade resultatet och strukturen i det verkliga webbdiarium som låg till grund för studien [35]. Syftet med dessa ändringar var att successivt öka datasetets realism och säkerställa att den syntetiska data i så hög utsträckning som möjligt efterliknades det ursprungliga datasetets uppbyggnad. Tabell 3 visar de ändringar som gjordes för de olika versionerna av datasetet.

Tabell 3. Utveckling av prompt

Version av dataset	Observation från föregående version	Ändring i denna version
V2	Diarienummer saknade sekventiell logik per enhet och år	Lade till regel om löpnummer som börjar på 001 per enhet och år
V2	För hög andel personnamn som avsändare/mottagare	Justerade prompten för större andel företag och kommunala organ
V2	För låg andel skyddade poster jämfört med diariet	Specificerade att cirka 20% ska ha avsändare/mottagare “[skyddat]”
V3	Mer realistiskt	Lade till händelsetabell, med egna specificerade instruktioner
V4	Saknade relation mellan tabeller	Lade till diarienummer som främmande nyckel i händelsetabellen samt ett unikt händelse-ID som primärnyckel

För att skapa transparens och göra det möjligt att reproducera studien återges den slutgiltiga prompten i Appendix A.

När promptarna bedömdes ge ett tillräckligt tydligt och realistiskt resultat skapades ett initialt ett dataset om 790 000 ärenden, vilket motsvarade den största observerade

datamängden i studiens empiriska underlag [38]. Utöver den storleken observerades även storlekarna 65 000 och 21 000 [35], [39]. För att undvika alltför stora intervall mellan datasetstorlekarna beräknades tre storlekar mellan 65 000 och 790 000. Dessa var jämnt fördelade på intervallet och underlättade för att kunna identifiera prestandatrender och eventuella brytpunkter där systemens beteende förändras. Tabell 4 visar en sammanställning av storlekarna på de dataset som användes i studien.

Tabell 4. Översikt av dataseten

Region	Antal ärenden[1]
Kronoberg	21 000
Jämtland Härjedalen	65 000
Testdataset 1	246 250
Testdataset 2	427 500
Testdataset 3	608 750
Stockholm	790 000

3.2.2 Utformning av queries

För att säkerställa att de valda testfrågorna speglar realistiska arbetsmoment i ett regionalt ärendehanteringssystem utformades queries med utgångspunkt i återkommande verksamhetsprocesser. Syftet var att skapa ett testupplägg som efterliknar faktiska operationer som kan förekomma i offentlig förvaltning, exempelvis registrering, uppföljning och sökning av ärenden.

Som metodiskt stöd användes användarberättelse (eng. user stories) för att formulera verksamhetsnära queries [40, s. 170]. User stories är vanligen utformade enligt följande mall:

Som X vill jag utföra Y för att uppnå Z

Där X representerar användarrollen, Y den uppgift som ska utföras och Z det önskade resultatet [40, s. 170]. Syftet med att använda user stories var att motivera och stärka valet av testqueries i studien. De valda user stories utformades med utgångspunkt i de behov som identifierats som viktiga i regionala ärendehanteringssammanhang, så som sökbarhet, statusuppföljning, historikhantering och aggregering.

De centrala funktionerna i ett ärendehanteringssystem har tidigare beskrivits i *Avsnitt 2.2 Ärendehantering och offentlig kontext*. Med utgångspunkt i dessa funktioner formulerades nio user stories, vilka därefter översattes till funktionellt likvärdiga queries i MySQL och MongoDB. De utformade queries omfattade registrering av nya ärenden,

[1] Explicit för de verkliga dataseten är att antalet ärenden är avrundat till närmsta tusental

exakt sökning, intervallsökning, uppdatering, borttagning, aggregering samt hämtning av ärendehistorik. De user stories som användes i studien och deras tillhörande querytyp presenteras i Tabell 5.

Tabell 5. Sammanställning av user stories och tillhörande querytyper

ID	User story	Querytyp
US1	Som medarbetare vill jag kunna se status för ett specifikt diarienummer för att snabbt kunna följa ärendets nuvarande handläggningsläge.	Enkla uppslag
US2	Som handläggare vill jag kunna visa alla ärenden kopplade till en specifik nämnd för att kunna få en överblick över inkomna och pågående ärenden inom ansvarsområdet.	Filter
US3	Som behörig handläggare vill jag kunna filtrera fram pågående sekretessklassade ärenden för att hantera känsliga handlingar korrekt.	Filter
US4	Som chef vill jag kunna se alla ärenden registrerade under ett specifikt datumintervall för att följa arbetsbelastningen över tid.	Intervall-sökning
US5	Som verksamhetschef vill jag kunna se antal ärenden per nämnd för att följa resursfördelning och ärendemängd.	Aggregering
US6	Som handläggare vill jag kunna se alla händelser kopplade till ett specifikt ärende för att följa dess historik.	Relations-baserad sökning
US7	Som handläggare vill jag kunna uppdatera status på ett ärende för att spegla det aktuella handläggningsläget.	Uppdatering
US8	Som registrator vill jag kunna lägga till ett nytt ärende i systemet för att säkerställa att inkommande handlingar registreras och kan följas upp.	Insättning
US9	Som administratör vill jag kunna ta bort eller arkivera felregistrerade ärenden för att upprätthålla datakvalitet.	Borttagning

3.3 Genomförande av jämförelsestudie

Den jämförande studien genomfördes genom att samma syntetiska dataset importerades i MySQL och MongoDB. Därefter utformades likvärdiga queries i respektive system baserade på centrala funktioner i ett regionalt ärendehanteringssystem. För att möjliggöra jämförelser mellan databaserna testades samma queries på samtliga datasetstorlekar. För queries som utgick från ett specifikt diarienummer, exempelvis enkla uppslag och uppdateringar, valdes diarienumret som förekom i respektive datasetstorlek. Inom varje datasetstorlek användes samma diarienummer i både MySQL och MongoDB, för att säkerställa att databashanteringssystemen testades mot motsvarande ärende. Exempelvis användes diarienummer GNU/547/2006 vid testning av US1 vid 790 000 ärenden i båda databaserna.

Eftersom datasetet är syntetiskt och AI-genererat är de enskilda diarienumren inte viktiga för reproducerbarheten i strikt mening, eftersom en ny generering av datasetet inte nödvändigtvis skulle skapa exakt samma poster. Däremot stärker denna urvalsprincip jämförbarheten mellan MySQL och MongoDB i det genomförda experimentet. För intervallsökningar användes däremot samma datumintervall i samtliga dataset och i båda databashanteringssystemen. De fullständiga queryscripten återfinns i Appendix B.

För att minska påverkan av tillfälliga variationer upprepades varje query 10 000 gånger med hjälp av for-loopar. För queries med mycket långa exekveringstider reducerades antalet iterationer per körning till 1 000 för att undvika timeout-begränsningar. Dessa körningar upprepades därefter tio gånger, vilket innebar att även dessa queries totalt exekverades 10 000 gånger. Antalet iterationer motiverades av att standardfelet för ett medelvärde minskar i relation till kvadratroten av antalet observationer. En förenklad approximation kan uttryckas som ekvation (1). Vid 10 000 iterationer motsvarar faktorn i ekvation (1) värdet 0,01, vilket motsvarar cirka 1%. Fler upprepningar ger därför ett mer stabilt medelvärde, även om den osäkerheten fortfarande beror på variationen i mätvärdena. Approximationen används här för att motivera valet av antal iterationer, snarare än som ett exakt mått på mätosäkerheten.

$$e \approx \frac{1}{\sqrt{n}} \tag{1}$$

För att beräkna exekveringstiden skapades en timer som noterade tiden före och efter att queryn exekverats, varefter differensen användes för att beräkna exekveringstiden. Resultatet presenterades på en logaritmisk skala för att underlätta avläsning och förändring över datamängderna. Ytterligare beräknades exponenten utifrån antagandet att exekveringstiden kan approximeras enligt ekvation (2) där n motsvarar datasetstorleken.

$$tid(n) = n^p \quad (2)$$

För varje query och databashanteringssystem beräknades logaritmen av både datasetstorleken och exekveringstiderna. Exponenten p beräknades därefter som lutningen mellan log(datasetstorlek) och log(exekveringstid). Beräkningarna genomfördes i Microsoft Excel med funktionen LUTNING. Ett högre p-värde indikerar att exekveringstiden ökar snabbare vid större datamängder.

Resultaten analyserades därefter med fokus på skillnader i prestanda, skalbarhet och datamodellens lämplighet. För att möjliggöra en rättvis jämförelse användes samma dataset, samma övergripande frågetyper och så likvärdiga testförutsättningar som möjligt i båda databashanteringssystemen.

MySQL och MongoDB kan optimeras genom att använda index anpassade för de queries som exekverades i databaserna. Indexering används för att effektivisera sökningar genom att reducera behovet av fullständig genomsökning av datamängden, vilket kan minska exekveringstiden [41], [42]. I studien skapades därför flera index baserat på de fält som användes testoperationerna. Indexens användning kontrollerades med hjälp av databashanteringssystemens verktyg för exekveringsanalys: EXPLAIN i MySQL och explain(”executionStats”) i MongoDB. De index som nyttjades vid exekveringen av testoperationerna behölls och användes i den slutliga testningen. Dessa index redovisas i Tabell 6.

Tabell 6. Sammanställning av index kopplade till studiens user stories

Indexerat fält	Kopplad user story	Användes för
diarienummer	US1, US6, US7	Exakta uppslag, sökning efter händelser och uppdatering av ärendestatus
diarienummer + arendestatus	US2	Filtrering av ärenden utifrån nämndprefix och status
registrerat_datum	US4	Intervallsökning utifrån registreringsdatum
arendestatus + sekretess	US3	Filtrering av pågående sekretessmarkerade ärenden

Indexeringen genomfördes på motsvarande fält i båda databashanteringssystemen i den mån det var möjligt. Indexen fungerar dock inte identiskt i MySQL och MongoDB, eftersom systemen har olika datamodeller och interna optimeringsmekanismer. Däremot stärker det jämförbarheten mellan systemen genom att båda databaserna gavs möjlighet att använda index vid de operationer där detta var relevant.

4. Resultat

I detta kapitel presenteras resultaten från jämförelsestudien med fokus på svarstid, skalbarhet och skillnader i datamodellering mellan MySQL och MongoDB. Resultaten har delats upp i sex avsnitt baserat på de kategorier som studiens user stories, och tillhörande queries, kan delas in i.

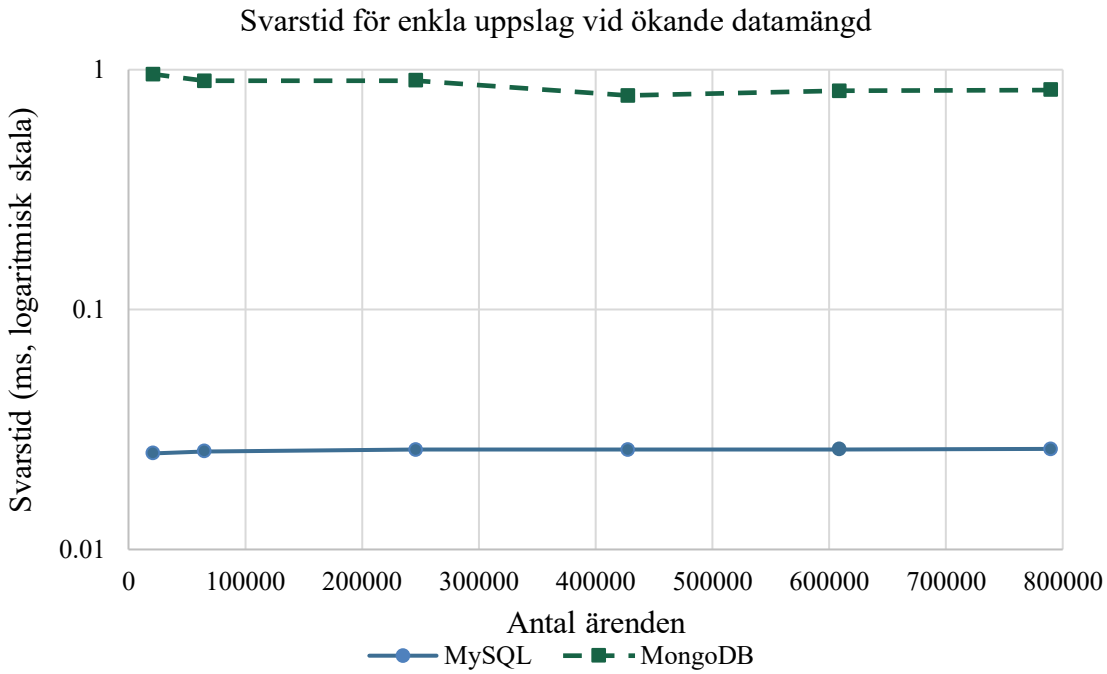
Eftersom exekveringstiderna varierade kraftigt mellan querytyperna redovisas diagrammen med logaritmisk skala. Detta gör det lättare att jämföra förändringsmönster mellan databaserna, men innebär samtidigt att visuella avstånd i figurerna inte ska tolkas som linjära skillnader.

4.1 Enkla uppslag

User story 1 tolkades till en query för att mäta svarstiden vid exakt uppslagning av ett diarienummer för hämtning av ärendets status.

Tabell 7. Exekveringstid US1

Antal rader	MySQL (ms)	MongoDB (ms)
21 000	0,0251	0,9579
65 000	0,0256	0,8966
246 250	0,0260	0,9016
427 500	0,0261	0,7808
608 750	0,0262	0,8156
790 000	0,0263	0,8246



Figur 3. Svarstid för US1 vid ökande datamängder på logaritmisk skala.

Resultatet i Tabell 7 visar att MySQL uppvisade en lägre svarstid än MongoDB för samtliga datamängder. Figur 3 illustrerar samtidigt att exekveringstiden för båda databaserna förblev relativt stabil vid ökande datamängder. MongoDB uppvisade genomgående högre svarstider än MySQL, men resultaten visar inte någon tydlig ökning i takt med datamängdens storlek. I stället varierade exekveringstiderna mellan datamängderna, där vissa större datamängder uppvisade kortare exekveringstid än mindre datamängder.

4.2 Filtrering

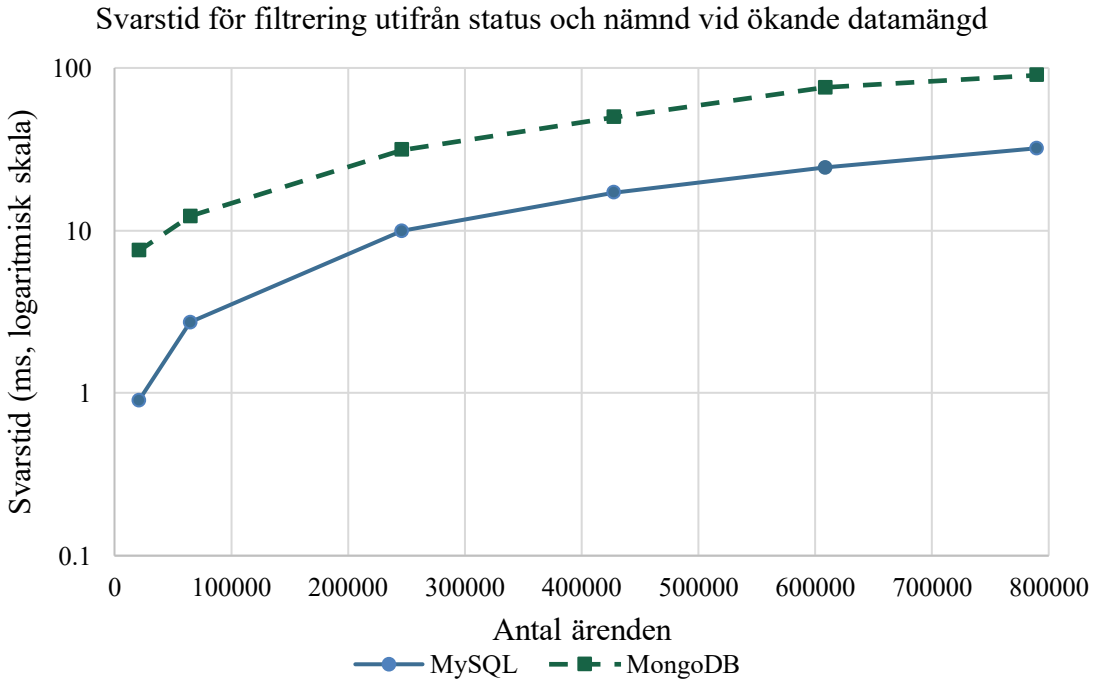
User story 2 och user story 3 tolkades till queries med filtrering. Den första queryn användes för att mäta exekveringstiden vid filtrering av pågående ärenden för en specifik nämnd. Den andra queryn användes för att mäta exekveringstiden vid filtrering av pågående sekretessbelagda ärenden.

Tabell 8. Exekveringstid US2

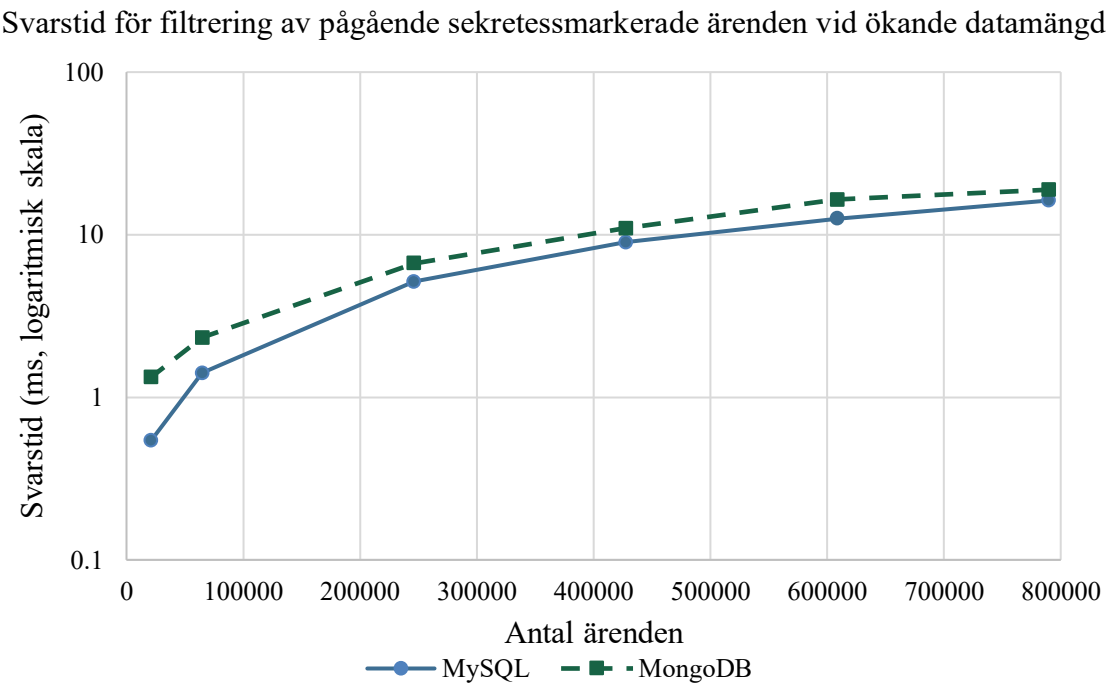
Antal rader	MySQL (ms)	MongoDB (ms)
21 000	0,9040	7,5091
65 000	2,7255	12,2283
246 250	9,9467	31,2644
427 500	17,1416	49,7477
608 750	24,4237	75,8468
790 000	32,0712	90,1568

Tabell 9. Exekveringstid US3

Antal rader	MySQL (ms)	MongoDB (ms)
21 000	0,5412	1,3278
65 000	1,4106	2,3240
246 250	5,1489	6,6696
427 500	8,9704	10,9415
608 750	12,5708	16,4393
790 000	16,2362	18,9151



Figur 4. Svarstid för US2 vid ökande datamängder på logaritmisk skala



Figur 5. Svarstid för US3 vid ökande datamängder på logaritmisk skala

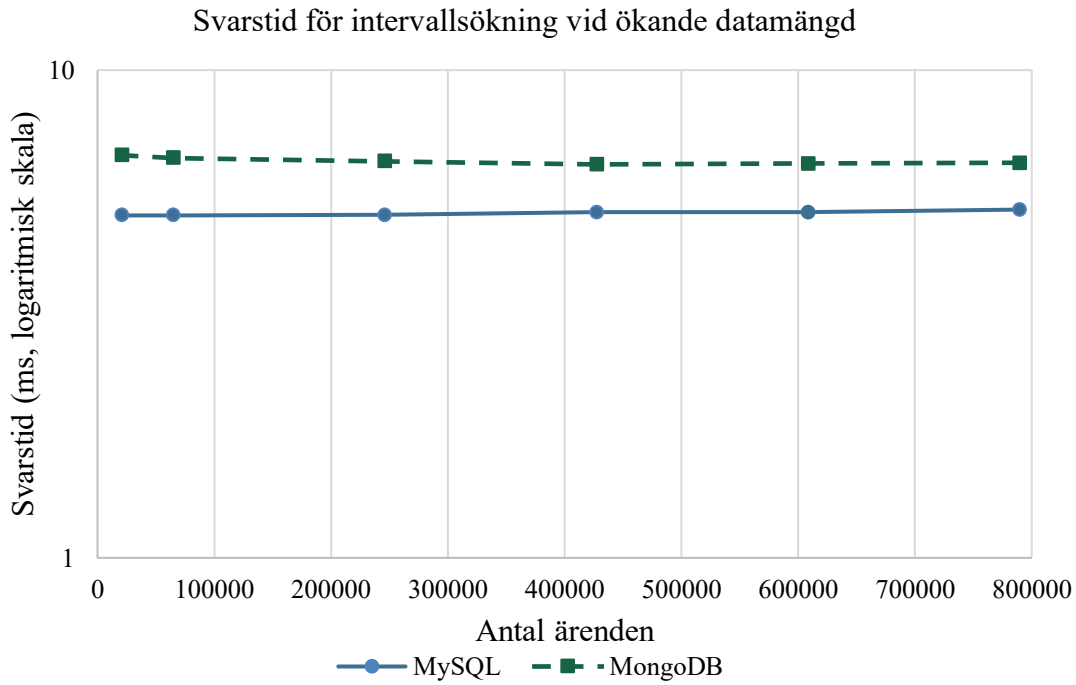
Resultatet i Tabell 8 och Tabell 9 visar att MySQL uppvisade lägre svarstider än MongoDB för både US2 och US3 vid samtliga datamängder. Figur 4 och Figur 5 visar samtidigt att exekveringstiden ökade successivt i takt med att datamängderna ökade för båda databaserna. Skillnaden mellan databaserna var särskilt tydlig i Figur 4, där MongoDB uppvisade högre svarstider vid större datamängder.

4.3 Intervallsökning

User story 4 tolkades till en query som användes för att beräkna exekveringstiden vid hämtning av ärenden som inkommit under ett givet datumintervall.

Tabell 10. Exekveringstid US4

Antal rader	MySQL (ms)	MongoDB (ms)
21 000	5,0361	6,6903
65 000	5,0410	6,6073
246 250	5,0452	6,5011
427 500	5,1080	6,4070
608 750	5,1132	6,4347
790 000	5,1769	6,4547



Figur 6. Svarstid för US4 vid ökande datamängder på logaritmisk skala.

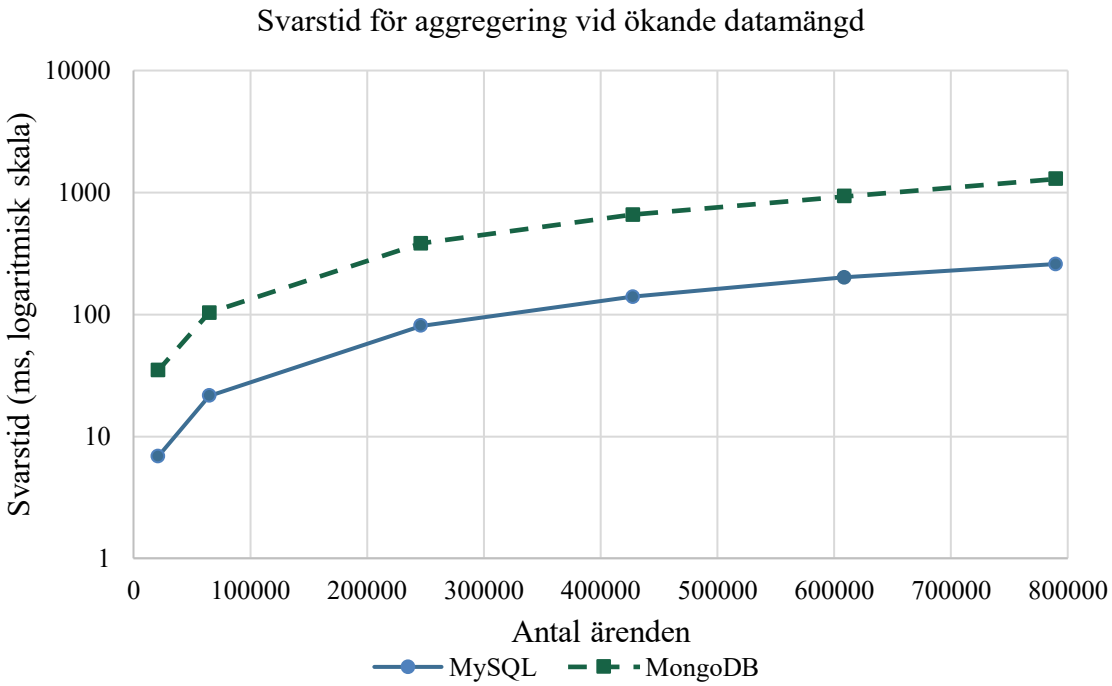
Resultatet i Tabell 10 visar att MySQL uppvisade lägre exekveringstider än MongoDB för US4 vid samtliga datamängder. Figur 6 illustrerar att exekveringstiden förblev stabil över ökande datamängder för båda databaserna. I Tabell 10 visar MySQL en låg och relativt stabil exekveringstid över samtliga datamängder, MongoDB visar ingen tydlig ökning i takt med datamängdens storlek. Precis som för US1, som avsåg enkla uppslag, visar resultatet att denna typ av query inte påverkades i någon större utsträckning av ökande datamängd i den aktuella testmiljön.

4.4 Aggregering

User story 5 tolkades till en query som användes för att mäta exekveringstiden vid aggregering av ärenden baserat på prefixet i diarienumret för att beräkna antal ärenden per nämnd.

Tabell 11. Exekveringstid US5

Antal rader	MySQL (ms)	MongoDB (ms)
21 000	6,8228	34,6041
65 000	21,5439	103,0138
246 250	80,6545	381,0375
427 500	139,4916	655,9922
608 750	201,0347	926,8450
790 000	258,9652	1288,7259



Figur 7. Svarstid för US5 vid ökande datamängder på logaritmisk skala

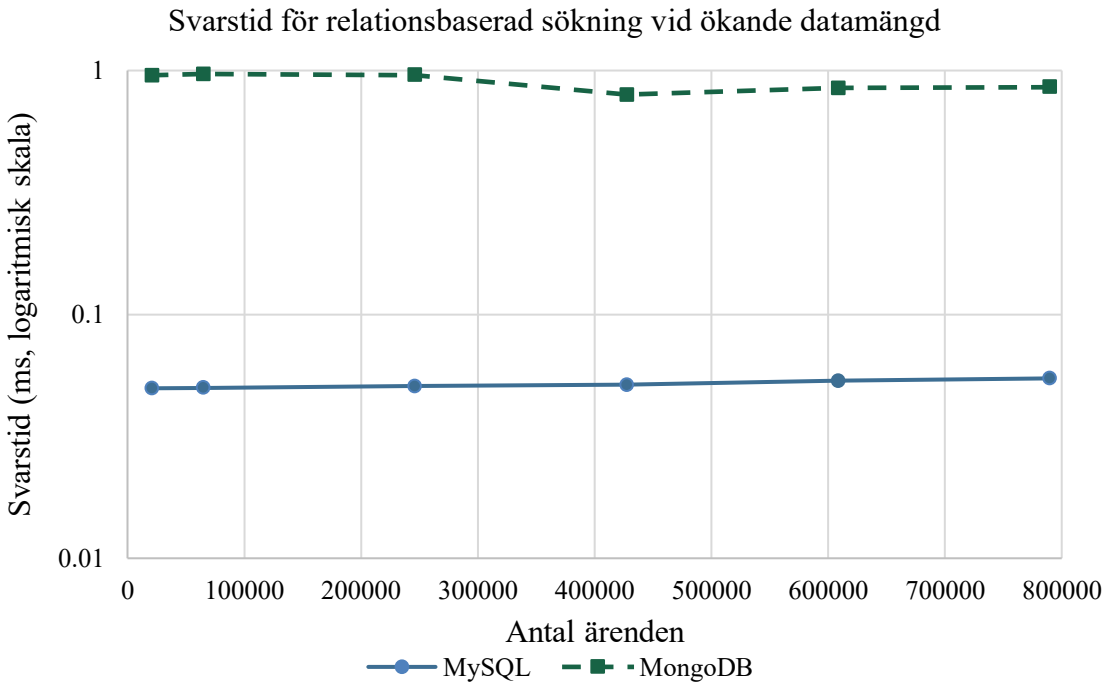
Resultatet i Tabell 11 visar att MySQL uppvisade lägre exekveringstider än MongoDB vid samtliga datamängder. Figur 7 visar samtidigt att exekveringstiden ökade för båda databaserna i takt med större datamängder.

4.5 Relationsbaserade operationer

User story 6 tolkades till en query som användes för att mäta exekveringstiden vid sökning efter relaterade händelser kopplade till ett specifikt ärende.

Tabell 12. Exekveringstid US6

Antal rader	MySQL (ms)	MongoDB (ms)
21 000	0,0498	0,9559
65 000	0,0500	0,9680
246 250	0,0507	0,9589
427 500	0,0514	0,7969
608 750	0,0535	0,8482
790 000	0,0547	0,8562



Figur 8. Svarstid för US6 vid ökande datamängder på logaritmisk skala.

Resultatet i Tabell 12 visar att MySQL uppvisar lägre exekveringstider än MongoDB för samtliga datamängder. Figur 8 visar att exekveringstiden för MySQL förblev relativt stabil, med en svag ökning vid större datamängder. MongoDB uppvisade däremot mindre variationer utan någon tydlig ökning i takt med datamängdens storlek. I vissa fall hade större datamängder kortare exekveringstid än mindre datamängder, vilket visar att MongoDB inte följde ett entydigt mönster kopplat till datamängdens storlek.

4.6 Skrivoperationer

User story 7, user story 8 och user story 9 tolkades till queries som utför skrivoperationer. User story 7 användes för att mäta exekveringstiden vid uppdatering

av ett ärendes status. User story 8 användes för att mäta exekveringstiden vid skapandet av ett nytt ärende, medan user story 9 användes vid borttagning av ett ärende.

Tabell 13. Exekveringstid US7

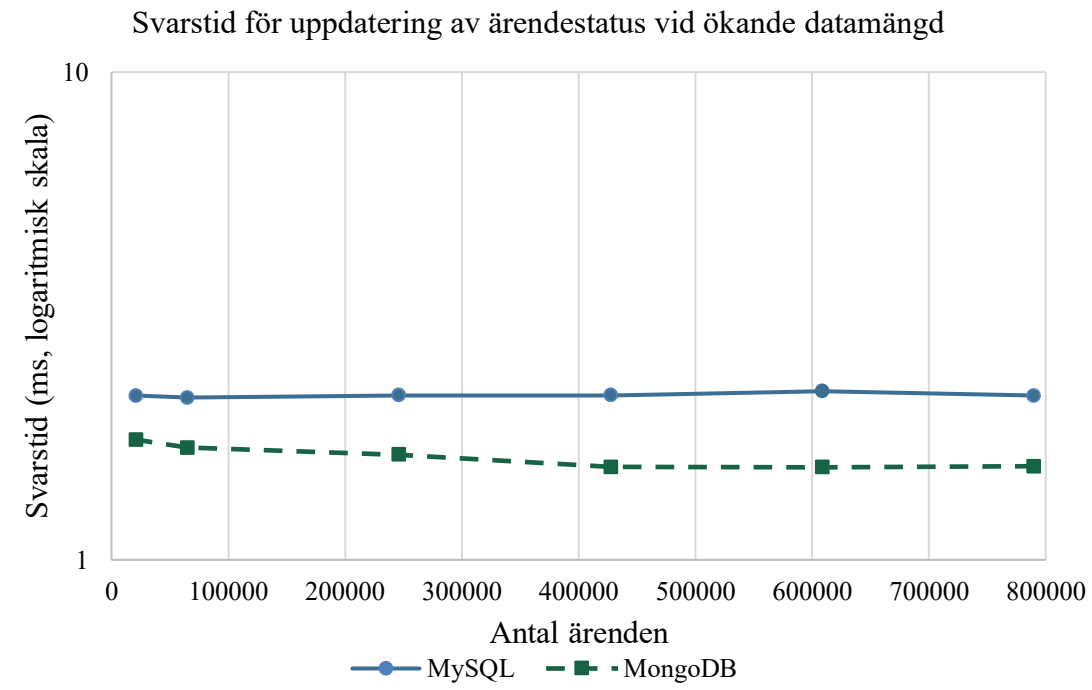
Antal rader	MySQL (ms)	MongoDB (ms)
21000	2,1695	1,7638
65000	2,1516	1,6976
246250	2,1742	1,6428
427500	2,1755	1,5492
608750	2,2168	1,5478
790000	2,1697	1,5554

Tabell 14. Exekveringstid US8

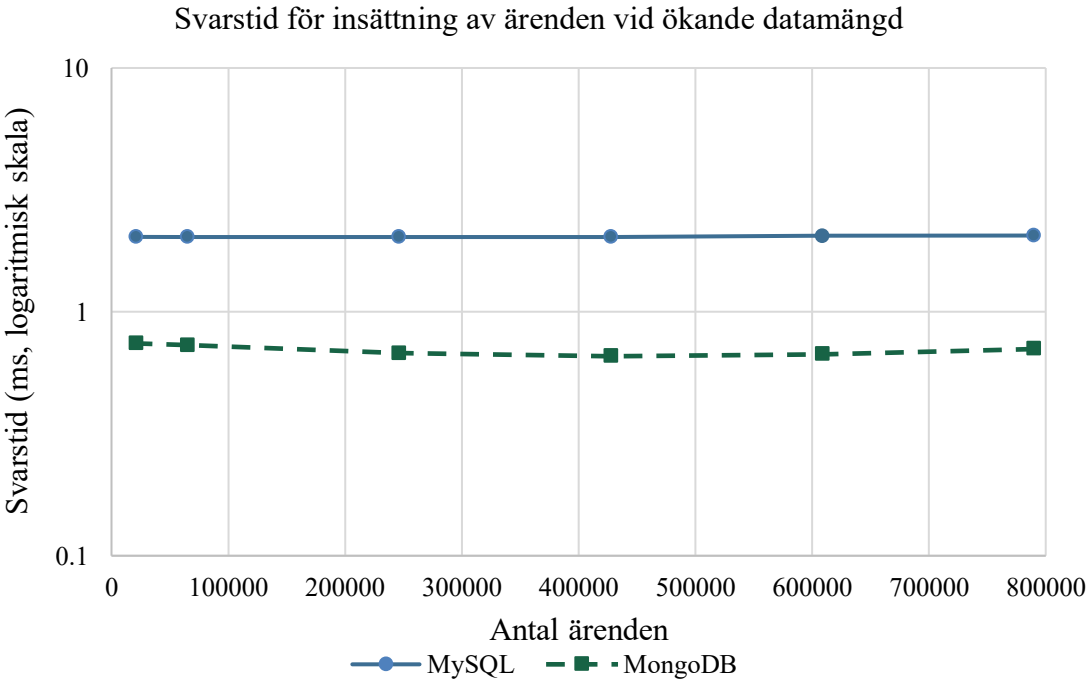
Antal rader	MySQL (ms)	MongoDB (ms)
21 000	2,0293	0,7421
65 000	2,0309	0,7291
246 250	2,0309	0,6762
427 500	2,0323	0,6575
608 750	2,0494	0,6702
790 000	2,0554	0,7053

Tabell 15. Exekveringstid US9

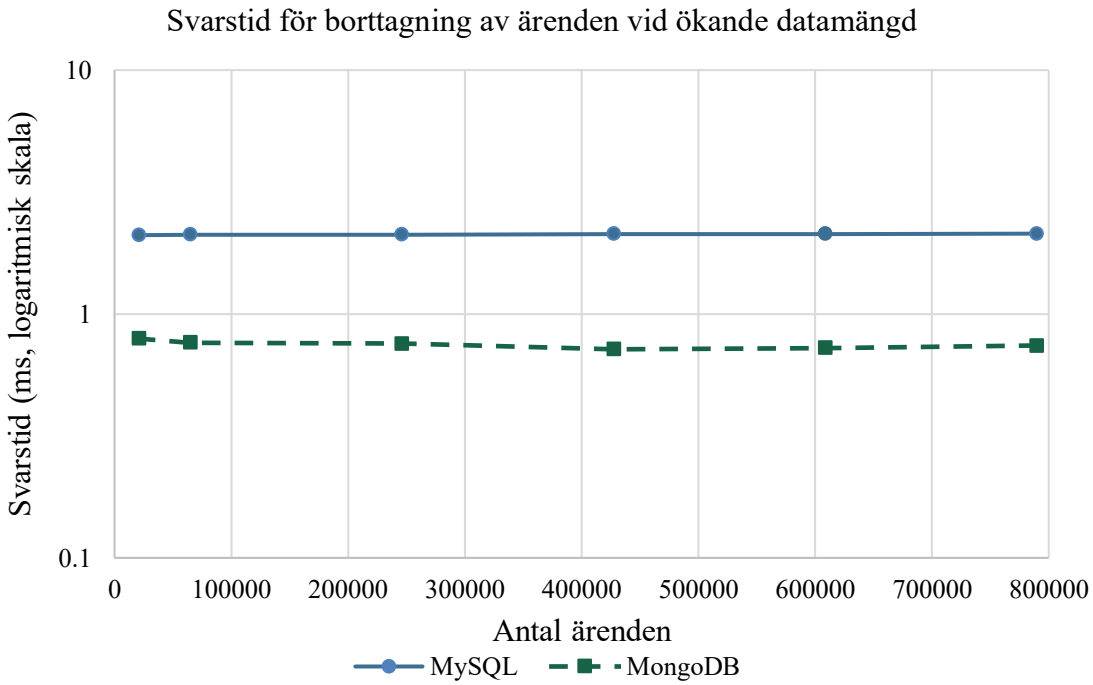
Antal rader	MySQL (ms)	MongoDB (ms)
21 000	2,1044	0,7921
65 000	2,1173	0,7618
246 250	2,1162	0,7551
427 500	2,1273	0,7166
608 750	2,1310	0,7261
790 000	2,1354	0,7411



Figur 9. Svarstid för US7 vid ökande datamängder på logaritmisk skala.



Figur 10. Svarstid för US8 vid ökande datamängder på logaritmisk skala.



Figur 11. Svarstid för US9 vid ökande datamängder på logaritmisk skala.

Resultatet i Tabell 13–15 visar att MongoDB uppvisade lägre exekveringstider än MySQL för samtliga datamängder vid skrivoperationer. Figur 9–11 visar samtidigt att exekveringstiderna för båda databaserna förblev relativt stabila vid ökande datamängder. För MongoDB framgår inget tydligt samband mellan datamängdens storlek och exekveringstid, eftersom tiderna endast varierade i mindre utsträckning mellan de olika datamängderna.

4.7 Sammanfattning av resultat från observationsstudie

Detta avsnitt sammanfattar studiens övergripande resultat med fokus på prestandamönster, skalbarhet, variation mellan querytyper och skillnader mellan databashanteringssystemen.

4.7.1 Övergripande prestandamönster

Sammantaget visar resultatet att MySQL uppvisade lägre exekveringstider än MongoDB för 6 av 9 user stories. Resultaten varierade dock beroende på querytyp. MySQL uppvisade generellt lägre svarstider vid uppslag, filtrering, aggregering och relationsbaserade operationer. Skillnaden mellan databaserna var proportionellt störst vid enkla uppslag och relationsbaserade operationer, där MongoDB uppvisade betydligt högre svarstider. MongoDB uppvisade samtidigt lägre svarstider vid skrivoperationer såsom insättning, uppdatering och borttagning av ärenden. Resultatet visar därmed att databaserna presterade olika beroende på vilken typ av operation som utfördes.

4.7.2 Exekveringstid vid ökande datamängder

Resultatet visar att exekveringstiden för majoriteten av querytyperna endast ökade marginellt när datamängden växte. Tydligast ökning observerades vid filtreringsqueries och aggregeringsqueryn vilket illustreras i Figurer 4, 5 och 7. Dessa operationer påverkades därmed mer av ökande datamängder än exempelvis enkla uppslag, intervallsökningar och skrivoperationer.

Aggregeringsqueryn uppvisade den största ökningen i exekveringstid för båda databashanteringssystemen. Filtreringsquerierna visade också en tydlig ökning, särskilt vid större dataset. Resultatet indikerar därför att operationer som kräver bearbetning av större delar av datamängden var mer känsliga för ökande datamängder än operationer som utgick från specifika uppslag eller enskilda poster.

4.7.3 Skillnader mellan querytyper och resurskrävande operationer

Skillnaden mellan querytyperna visar att vissa operationer var mer resurskrävande än andra. Aggregeringsqueryn uppvisade de högsta exekveringstiderna för båda databaserna och var därmed den mest resurskrävande operationen i studien. Även filtreringsqueries påverkades tydligt av ökande datamängder, särskilt vid större dataset.

Exakta uppslagningar och relationsbaserade operationer uppvisade däremot genomgående låga svarstider och relativt små förändringar mellan datasetstorlekarna. Skrivoperationer var också stabila över datamängderna, men skilde sig från övriga querytyper genom att MongoDB genomgående uppvisade lägre exekveringstid än MySQL. Sammantaget visar resultatet att aggregering och filtrering var mest känsliga för ökande datamängder, medan uppslag, relationsbaserade sökningar och skrivoperationer påverkades i mindre utsträckning.

4.7.4 Variation och stabilitet

Resultatet visar att majoriteten av querytyperna uppvisade relativt stabila exekveringstider mellan datamängderna. De största variationerna observerades för filtrerings- och aggregeringsqueries, där exekveringstiden ökade tydligt vid större datamängder. Skrivoperationer och enkla uppslag uppvisade däremot mindre variationer och förblev stabila även vid ökande datamängder.

Ett återkommande mönster var att MongoDB inte uppvisade något tydligt samband mellan datamängdens storlek och exekveringstid för flera av studiens user stories. Exekveringstiderna för enkla uppslag, intervallsökning, relationsbaserad sökning och skrivoperationer var relativt stabila men varierade mellan datamängderna utan någon konsekvent ökning när datamängden växte. För MySQL var sambandet mellan datamängd och exekveringstid generellt tydligare, även om ökningen i vissa fall var begränsad.

4.7.5 Skalningsmönster utifrån exponenten p

För att ytterligare sammanfatta hur exekveringstiden förändrades vid ökande exekveringstid beräknades exponenten p för respektive query och databashanteringssystem. Resultatet av beräkningarna redovisas i Tabell 16. Exponenten används som ett kompletterande mått på hur starkt exekveringstiden påverkades av datasetstorleken.

Tabell 16. Beräknad exponent p för respektive query

User story	Querytyp	MySQL p	MongoDB p
US1	Enkla uppslag	0,01123991	-0,0466089
US2	Filtrering	0,9808962	0,70016188
US3	Filtrering	0,94540688	0,75708906
US4	Intervallsökning	0,00635134	-0,0114366
US5	Aggregering	1,00120923	0,98791608
US6	Relationsbaserad sökning	0,0225007	-0,0427218
US7	Uppdatering	0,00354066	-0,0378796
US8	Insättning	0,00283909	-0,0267533
US9	Borttagning	0,00356418	-0,0224136

De högsta p-värdena återfanns för filtrerings- och aggregeringsqueries. Aggregeringsqueryn hade ett p-värde nära 1 för både MySQL och MongoDB, vilket tyder på att exekveringstiden ökade ungefär i takt med datamängden. Filtreringsqueries hade också relativt höga p-värden, framför allt för MySQL.

För enkla uppslag, intervallsökning, relationsbaserad sökning och skrivoperationer låg p-värdena däremot nära 0. Det innebär att exekveringstiden förändrades mycket lite när datamängden ökade. Några queries i MongoDB fick svagt negativa p-värden. Detta hänger samman med att MongoDB för flera querytyper inte uppvisade ett tydligt ökande mönster när datamängden växte. De negativa p-värdena bör därför inte tolkas som att större datamängder i sig leder till kortare exekveringstid, utan snarare som ett tecken på att exekveringstiderna var relativt stabila och varierade mellan mätningarna.

Sammantaget visar p-värdena att aggregering och filtrering var de operationer där exekveringstiden ökade tydligast med datamängden, medan övriga querytyper

uppvisade mer stabila exekveringstider. Detta stödjer den övergripande resultatbilden att skalbarheten främst påverkades av operationer som bearbetade större delar av datamängden.

5. Diskussion

I detta kapitel diskuteras studiens resultat i relation till tidigare forskning och forskningsfrågor. Kapitlet behandlar även metodmässiga begränsningar och deras betydelse för resultatens tillförlitlighet.

5.1 Resultatdiskussion

Denna del av diskussionen fokuserar på studiens resultat kopplat till forskningsfrågorna och tidigare forskning. Framför allt behandlas hur MySQL och MongoDB presterade vid olika typer av databasoperationer och vid växande datamängder. Diskussionen undersöker även hur skillnader i datamodell, indexering och databassystemens underliggande egenskaper kan ha påverkat resultaten.

5.1.1 Prestanda och querytyper

Resultatet från experimentet visar att prestandaskillnaderna mellan MySQL och MongoDB varierade beroende på vilken typ av query som exekverades. Detta tyder på att databassystemens styrkor och svagheter inte kan bedömas enbart baserat på ett sammanlagt resultat, utan behöver förstås i förhållande till respektive typ av databasoperation.

MySQL uppvisade särskilt god prestanda vid enkla uppslag och relationsbaserade operationer. Databasen presterade även bättre än MongoDB vid filtrerings-, intervall- och aggregeringssökningar. Detta kan delvis bero på att dessa operationer var väl anpassade till relationsmodellens tabellstruktur och de index som skapades för motsvarande kolumner. Eftersom flera av de operationer där MySQL presterade bättre än MongoDB utgick från indexerade kolumner, kan indexeringen ha bidragit till att begränsa mängden data som behövde genomsökas. Index fungerar som en genväg till relevanta rader i tabellen, och gör det möjligt för databasen att snabbare lokalisera de poster som matchar villkoren i queryn. I stället för att behöva läsa igenom hela tabellen kan databasen använda indexet för att begränsa sökningen. Detta är särskilt fördelaktigt vid operationer som bygger på filtrering utifrån specifika kolumnvärden, vilket kan förklara varför MySQL presterade väl i dessa delar av experimentet.

MongoDB presterade däremot bättre vid samtliga skrivoperationer. En möjlig förklaring är att MongoDB:s dokumentorienterade lagringsmodell kan ha varit väl anpassad för de skrivrelaterade operationer som testades. Samtidigt bör resultatet inte förklaras med att MySQL behövde hantera samtidiga skrivningar i flera relaterade tabeller, eftersom operationerna inte omfattade exempelvis insättning av både ärenden och tillhörande händelser. Om skrivoperationerna även hade omfattat tillhörande händelser hade skillnaden i prestanda kunnat diskuteras tydligare baserat på datamodellernas olika struktur. Även om MongoDB presterade bättre vid skrivoperationerna visar studien inte exakt vilka interna mekanismer i databasen som orsakade skillnaden. Resultatet visar

däremot att MongoDB, under experimentets förutsättningar, hanterade skrivoperationer mer effektivt än MySQL.

Att MongoDB presterade sämre vid filtrerings-, intervall- och aggregeringssökningar kan delvis förstås utifrån hur data strukturerades i dokumentmodellen. I MongoDB är händelserna inbäddade i ärendedokumenten något som kan vara fördelaktigt då ett helt ärende, med tillhörande information ska hämtas. Däremot kan operationer som kräver filtrering, summering eller bearbetning över många dokument bli mer resurskrävande, särskilt om flera dokument eller inbäddade fält behöver analyseras.

5.1.2 Exekveringstid vid ökande datamängder

Resultaten visar att datasetens storlek påverkade exekveringstiden i olika grad beroende på vilken query som exekverades. Majoriteten av experimentets operationer uppvisade relativt stabila exekveringstider, medan tre av operationerna påverkades tydligare av växande datamängder. Detta framgår i Figur 4, 5 och 7, där exekveringstiden ökade i takt med att datamängden växte. Dessa skillnader i resultatet visar att skalbarheten inte enbart beror på vilket databashanteringssystem som används, utan även vilken query som exekveras och hur den är utformad. Det beror också på om databasen kunde använda index vid sökningen och hur många rader eller dokument som behövde genomsökas för att besvara queryn.

De logaritmiska diagrammen används för att tydliggöra skillnader i hur exekveringstiderna förändras när datamängden ökar. På så sätt blir det lättare att se om ökningen följer ett relativt stabilt mönster eller om vissa operationer får en kraftigare ökning när datamängden ökar.

Filtrerings- och aggregeringsoperationerna påverkades mer av ökande datamängder och detta kan förklaras av att databasen i dessa fall behöver bearbeta fler poster för att identifiera, gruppera eller beräkna relevanta resultat. Om en query inte helt kan isoleras med index kan en större del av tabellen eller samlingen behöva genomsökas, något som leder till att exekveringstiderna ökar när datamängden växer.

Resterande operationer, så som enkla uppslag, intervallsökning, relationsbaserade operationer, och skrivoperationerna, visade alla relativt stabila exekveringstider. Enkla uppslag och relationsbaserade operationer kan ha dragit nytta av indexering av diarienummer, och de är sannolikt därför dessa operationer inte påverkas märkvärt av att datamängden ökar. Intervallsökningen utgick i sin tur från registrerat datum, där indexering kan ha bidragit till lägre exekveringstider.

En viktig metodisk aspekt är samma datumintervall användes för samtliga datamängder. Eftersom datamängden ökade medan datumintervallet hölls konstant, omfattade intervallsökningen en mindre procentuell andel av datasetet vid växande datasetstorlek. Detta kan ha bidragit till att exekveringstiderna framstod som mer stabila vid ökande

datamängder än vad de hade gjort om intervallet hade skalats proportionellt med datamängden.

I resultatavsnittet noterades avsaknaden av mönster för flertalet av studiens user stories för MongoDB. Det innebär att resultaten inte enbart kan förstås utifrån datamängdens storlek. En möjlig förklaring och felkälla kan vara att mätningarna påverkades av cachelagring, tillfällig systembelastning, indexering eller MongoDB:s frågeoptimering. Det kan även bero på att vissa operationer var så korta att de var mer känsliga för mindre förändringar i testmiljön och därför fick en större påverkan på resultatet. Resultaten bör därför tolkas med viss försiktighet vid jämförelser mellan enskilda datamängder, medan de fortfarande kan användas för att identifiera övergripande skillnader mellan databashanterarna och querytyper.

5.1.3 Datamodellens påverkan

Datamodellen hade betydelse för hur resultaten kan tolkas. MySQL strukturerade ärenden och händelser i separata tabeller, vilket passar en relationsmodell där samband mellan datatyper hanteras med primära och främmande nycklar. Fördelen med detta är att data hålls konsekvent, strukturerad och sökbar, något som är viktigt i ett ärendehanteringssystem. Överlag presterade MySQL bättre än MongoDB på alla queries förutom skrivoperationer, men den största skillnaden påvisades för relationsbaserade operationer och enkla uppslag. Detta kan indikera att relationsmodellen var väl anpassad för dessa typer av operationer.

Dokumentmodellen i MongoDB möjliggjorde att ärenden kunde lagras tillsammans med tillhörande händelser, vilket kan underlätta vid hämtning av all data som är kopplat till ett specifikt ärende. Resultatet för enkla uppslag visar däremot att det tog längre tid för MongoDB än MySQL att hämta ett specifikt ärendes aktuella status. En möjlig tolkning är att dokumentmodellen inte automatiskt förbättrar prestandan vid alla typer av ärendeuppslag. Strukturen framstod även som mindre fördelaktig vid aggregering och filtrering över många dokument.

Datamodellen påverkar därför hur resultaten kan förstås, särskilt eftersom olika operationstyper gynnas av olika sätt att strukturera data. Detta blir relevant för den fortsatta diskussionen om databasernas praktiska lämplighet i kontexten av ett regionalt ärendehanteringssystem.

5.1.4 Transaktionsegenskaper och konsistenskrav

Även om transaktionsegenskaper är en vanligt förekommande jämförelsepunkt i tidigare jämförelsestudier av SQL- och NoSQL-databaser, var det inte relevant för denna studie. MySQL och MongoDB stödjer båda ACID-egenskaper vid transaktioner och framstod inte därför som en huvudsaklig förklaring till de prestandaskillnader som observerades i studien. Eftersom studien inte testade samtidiga transaktioner, återställning efter fel eller olika konsistensnivåer bör slutsatserna om transaktionshantering dras med försiktighet. I

stället bör resultaten främst förstås baserat på querytyp, indexering, datamodell och studiens specifika testmiljö.

I ett ärendehanteringssystem behöver flera uppdateringar ofta ske samtidigt. Ett exempel är att om ett ärende flyttas mellan olika ansvariga enheter. Det kan innebära ändring av ansvarig enhet, tillägg av ny händelse och uppdatering av ärendets status. Dessa uppdateringar är beroende av varandra och bör därför genomföras inom samma transaktion för att säkerställa att systemet förblir konsistent. Om en del av uppdateringen misslyckas, exempelvis att status ändras men händelseposten inte sparas, riskerar systemet att hamna i ett inkonsistent tillstånd, vilket talar för vikten av ACID-baserad transaktionshantering i denna typ av system. I ett ärendehanteringssystem kan inkonsistent information innebära att olika handläggare ser olika status för samma ärende, vilket riskerar att påverka handläggning, uppföljning och beslutsfattande. Detta innebär att ACID-egenskaper, särskilt atomicitet och konsistens, kan vara viktiga vid bedömningen av databassystemens praktiska lämplighet.

Studien har inte empiriskt testat samtidiga transaktioner, rollback eller isoleringsnivåer. Därför kan inga direkta slutsatser dras om hur MySQL och MongoDB hanterar dessa situationer i praktiken. Däremot visar studien att transaktionshantering är en viktig aspekt vid val av databashanteringssystem för ärendehantering, särskilt eftersom data ofta behöver vara korrekt, spårbar och konsekvent över tid.

5.1.5 Praktisk användbarhet i regional kontext

Studien är baserad på verksamhetsbaserade user stories för ett regionalt ärendehanteringssystem. Resultatet visar att inget av databashanteringssystemen presterade bäst vid samtliga user stories. Det innebär att databasernas praktiska lämplighet beror på vilka operationer som är mest centrala i den aktuella verksamheten.

I ett regionalt ärendehanteringssystem är det sannolikt att uppslag, filtrering, historikhantering och uppföljning förekommer i större utsträckning än borttagning av ärenden. Eftersom MySQL uppvisade lägre exekveringstider i flera av dessa operationer framstår MySQL som särskilt relevant i ärendehanteringssystem där struktur, sökbarhet och konsistens är viktiga krav. Samtidigt visar resultatet att MongoDB uppvisade lägre exekveringstider vid skrivoperationer, vilket kan vara relevant i system där registrering, uppdatering och borttagning sker i hög omfattning.

I ett diarium lagras allmänna handlingar under längre tid innan de överförs till arkiv, vilket innebär att datamängden successivt ökar. I en regional organisation kan dessutom flera samtidiga användare, exempelvis handläggare och administrativ personal, behöva åtkomst till systemet samtidigt. Detta ställer krav på att databasen kan hantera både växande datamängder och återkommande sökningar på ett tillförlitligt sätt.

Datamodellen är också viktig för att förstå resultatet. Ärendehanteringsdata bedöms vara huvudsakligen strukturerad, eftersom information som diarienummer,

registreringsdatum, status, avsändare eller mottagare samt sekretessnivå följer en tydlig och fördefinierad struktur. Det finns även tydliga relationer mellan olika objekt i systemet. Ett ärende kan exempelvis vara kopplat till flera händelser, till exempel statusuppdateringar, kommentarer eller beslut, vilket skapar en en-till-många-relation mellan ärenden och deras historik. Detta medför ett behov av att kunna sammanställa information från flera relaterade tabeller, exempelvis genom joins, vilket talar för att en relationsdatabas kan vara väl lämpad för denna typ av system.

5.2 Metoddiskussion

I följande avsnitt diskuteras studiens metodval, styrkor och begränsningar. Med fokus på dataset, testmiljö, queryval samt studiens tillförlitlighet och jämförbarhet.

5.2.1 Syntetiskt dataset och AI-genererade data

All användning av AI bör göras med viss försiktighet och det AI producerar bör granskas kritiskt, eftersom genererade data kan innehålla felaktigheter eller avvikelser från verkliga förhållanden. Beslutet att generera datasetet med hjälp av AI fattades grundat i att det inte gick att exportera de befintliga dataset som fanns i de webbdiarium som fanns att tillgå. Genom att använda de befintliga datasetens metadata kunde en prompt formuleras för att skapa ett syntetiskt dataset som efterliknade det verkliga, med fokus på format på diarienummer och datatyper.

Det dataset som genererades till studien bestod av 790 000 rader. Att exportera ett redan befintligt dataset hade troligtvis varit mer tidseffektivt, men då det inte var möjligt återstod två alternativ, AI-generera ett dataset eller en manuell återskapning. En manuell återskapning av motsvarande datamängd hade samtidigt varit alltför tidskrävande. Generativ AI framstod därför som ett praktiskt genomförbart alternativ för att skapa ett stort och strukturerat testdataset.

En central fördel med att använda syntetiska data är att studien kunde genomföras utan att använda verkliga personuppgifter, sekretesskänsligt innehåll eller produktionsdata. För denna studie blir detta särskilt relevant då datasetet, i det ärendehanteringssystem som använts i studien, inte kan exporteras och dessutom i viss mån kan anses innehålla känslig information. Detta motiverar användning av generativ AI som möjliggör ett syntetiskt men realistiskt dataset som efterliknar verkliga ärenden, utan att riskera individers eller organisationers integritet.

Den främsta skillnaden mellan det syntetiska datasetet och det verkliga som försökte efterliknas, är att i det verkliga förekommer PDF-filer. Studien avgränsades till strukturerade ärendedata och inkluderade därför inte PDF-dokument. Att inkludera dessa dokument hade inneburit ytterligare komplexitet och skiftat studiens fokus från databashantering av strukturerade data till hantering av filer och dokumentlagring.

5.2.2 Begränsningar i testmiljö och implementation

Experimentet genomfördes på samma hårdvara i en lokal miljö. Detta skapade en kontrollerad miljö där externa faktorer, exempelvis skillnader i hårdvara eller nätverksförhållanden, kunde begränsas. Samtidigt innebär det en begränsning, eftersom databaserna inte testades i en distribuerad eller molnbaserad miljö. Eftersom MongoDB ofta används i distribuerade miljöer kan den lokala testmiljön ha begränsat möjligheten att synliggöra systemets styrkor kopplade till horisontell skalning. Resultatet bör därför inte tolkas som ett generellt omdöme om MongoDB:s skalbarhet, utan som ett resultat av den specifika implementation och testmiljö som användes i studien.

En ytterligare begränsning rör implementation av datamodellerna. I MySQL användes två tabeller, en för ärenden och en för händelser. Tabellerna relaterades genom diarienummer, där diarienumret i ärendetabellen användes som främmande nyckel i händelsetabellen. I MongoDB lagrades händelser i respektive ärendedokument, något som går i linje med MongoDB:s dokumentation och rekommendationer för dokumentorienterad modellering.

Databaserna modellerades därför inte på ett identiskt sätt, utan snarare baserat på respektive databassystems rekommenderade användningssätt. Syftet med detta var att undvika att någon av databaserna fick sämre förutsättningar genom att tvingas följa en datamodell som inte är anpassad efter dess arkitektur. Samtidigt innebär detta en metodologisk avvägning, eftersom skillnader i resultat inte enbart kan härledas till databasmotorn, utan även hur data strukturerades i respektive system. Denna avvägning bedömes dock motiverad, eftersom studien jämför databaserna baserat på hur de sannolikt skulle användas i praktiken.

5.2.3 Queryval och verksamhetskoppling

De user stories som formulerades för teststudien var baserade på de centrala funktioner som ett ärendehanteringssystem har, samt även hur en regions webbdiarium var utformat. De formulerades även efter de olika querytyper som finns för att täcka flera huvudsakliga funktioner och operationstyper i databashanteringssystemen. Det kan därför vara troligt att de user stories som formulerats inte är helt verklighetstrogna, och därför inte fullt ut speglar verkliga användningsmönster.

I början av projektet kontaktades möjliga intervjupersoner med värdefulla insikter i verksamheten nära ett regionalt ärendehanteringssystem. Intervjuerna förväntades ge information om hur ett ärendehanteringssystem används, vilken typ av data som lagras samt storlek på datasetet i det berörda ärendehanteringssystemet. User stories baserade på hur systemet faktiskt används hade gett ett än mer verklighetstroget testscenario. De tilltänkta intervjupersonerna avböjde till intervjuer och hänvisade till att den typen av information hade riskerat att avslöja sekretessbelagd information.

I efterhand visade det sig att den enda information som gick förlorad från uteblivna intervjuer var uppgifter om användningsbeteende. Att i stället formulera fiktiva user stories baserade på troliga användningsmönster tillsammans med querytyper, kan tänkas ha gett ett bredare resultat i prestandatestning och skalbarhet.

5.2.4 Tillförlitlighet, jämförbarhet och begränsningar

För att eftersträva en rättvis jämförelsestudie, i den mån det varit möjligt, har följande principer tillämpats. Inledningsvis användes identiska dataset i båda databaserna för att säkerställa att motsvarande databasoperationer kunde utföras under likvärdiga förutsättningar. Vidare skapades index för motsvarande kolumner i respektive databas. För att minimera ytterligare felkällor genomfördes samtliga tester på samma hårdvara och under likvärdiga nätverksförutsättningar.

För att beräkna den genomsnittliga exekveringstiden för respektive query upprepades varje operation 10 000 gånger. Detta reducerade risken för att mätfel och enskilda avvikande mätvärden skulle påverka resultaten. Som nämndes i *Avsnitt 3.3 Genomförande av jämförelsestudien* minskar felmarginalen proportionellt med antalet iterationer, enligt ekvation (1). För att minimera felmarginalen ytterligare hade ett högre antal iterationer varit nödvändigt, men detta hade samtidigt ökat experimentets totala tidsåtgång. En felmarginal på 1% bedömdes vara tillräcklig i relation till de observerade skillnaderna mellan databaserna.

Trots studiens begränsningar bedöms resultaten ha en relativt hög tillförlitlighet, då samma metodik, testmiljö och dataset konsekvent användes genom hela experimentet. Dessa metodmässiga avvägningar bidrar till att resultaten kan jämföras på ett mer rättvist och enhetligt sätt.

6. Slutsatser

Detta avslutande kapitel sammanfattar studiens huvudsakliga slutsatser och presenterar förslag till vidare forskning. Studien syftade till att undersöka hur MySQL och MongoDB skiljer sig åt som databashanteringslösningar i kontexten av ett regionalt ärendehanteringssystem, med fokus på exekveringstid, datamodell och transaktionsegenskaper. Följande forskningsfrågor har legat till grund för studien:

1) Vilka skillnader i exekveringstid kan identifieras mellan MySQL och MongoDB vid olika databasoperationer utformade för en regional ärendehanteringskontext?
2) Hur kan resultaten förstås utifrån datamodell, transaktionsegenskaper och krav i ett regionalt ärendehanteringssystem?

6.1 Skillnader i exekveringstid

Sammantaget visar studiens resultat att MySQL uppvisade lägre exekveringstider för sex av nio user stories. MySQL presterade väl på enkla uppslag, filtrering, intervallsökning, aggregering och relationsbaserade operationer. Den största skillnaden mellan MySQL och MongoDB observerades vid enkla uppslag och relationsbaserade operationer, där MySQL presterade bäst. MongoDB presterade bättre än MySQL på samtliga skrivoperationer, dvs. insättning, uppdatering och borttagning av ärenden.

Skillnaden mellan databashanteringssystemen var därmed beroende av vilken typ av databasoperation som utfördes. Studien visar därmed inte att ett av systemen generellt presterar bäst i alla situationer, utan att MySQL och MongoDB har olika styrkor beroende på operationstyp.

6.2 Resultatens betydelse i en regional ärendehanteringskontext

Resultaten kan förstås med utgångspunkt i datamodell, transaktionsegenskaper och krav i regionalt ärendehanteringssammanhang. Ärendehanteringsdata är i hög grad strukturerad och består av återkommande fält, exempelvis diarienummer, registreringsdatum, status och avsändare eller mottagare. Det finns även tydliga relationer mellan ärenden och tillhörande händelser, vilket gör relationsmodellen relevant för denna typ av system.

Eftersom MySQL uppvisade lägre exekveringstider vid flera operationer kopplade till sökning, filtrering, historik och uppföljning framstår MySQL som särskilt relevant i system där struktur, sökbarhet och konsistens är viktiga krav. MongoDB:s styrkor framträdde främst i skrivoperationer, vilket kan vara relevant i system där registrering och uppdatering sker i hög omfattning.

Studien har inte empiriskt testat transaktionshantering, samtidiga användare eller rollback. Däremot visar den valda kontexten att transaktionsegenskaper är viktiga att ta i

beaktning, eftersom ärendehanteringssystemen ofta kräver korrekt, spårbar och konsekvent information över tid.

6.3 Vidare forskning

För vidare forskning inom ämnet föreslås att undersöka MySQL och MongoDB i en mer produktionsnära miljö, exempelvis med verkliga eller anonymiserade data. Det vore även relevant att genomföra tester i distribuerade eller molnbaserade miljöer, särskilt eftersom MongoDB ofta används i sådana sammanhang.

Framtida studier bör även inkludera samtidiga användare, transaktionstester, rollback, isoleringsnivåer och mer komplexa arbetsflöden där flera relaterade poster uppdateras samtidigt. Det skulle ge en tydligare bild av hur databashanteringssystemen fungerar i praktisk ärendehantering.

Ytterligare forskning kan också inkludera dokumentfiler, exempelvis PDF-handlingar, eftersom denna studie avgränsade strukturerade metadata. På så sätt skulle framtida studier kunna undersöka databashantering i ett mer komplett ärendehanteringssystem.

Referenser

[1] *Arkivlag*. Åtkomstdatum: 20 april 2026. [Online]. Tillgänglig vid: https://www.riksdagen.se/sv/dokument-och-lagar/dokument/svensk-forfattningssamling/arkivlag-1990782_sfs-1990-782/#overgang

[2] J. Bygdemark och A. Arvidsson, ”Jämförelse mellan Oracle RDBMS, Oracle NoSQL och MongoDB”, Kandidatuppsats, Institutionen för datavetenskap, Umeå universitet, Umeå, Sverige, 2019. [Online]. Tillgänglig vid: https://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-163179

[3] C. Wannegård, ”Faktorer som har samband med stress vid IT-användning”, Kandidatuppsats, Psykologiska instutionen, Stockholms universitet, Stockholm, Sverige, 2006. Åtkomstdatum: 18 maj 2026. [Online]. Tillgänglig vid: https://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-1039

[4] Sokigo, ”Konsten att skapa ordning i en komplex värld”. Åtkomstdatum: 18 maj 2026. [Online]. Tillgänglig vid: https://sokigo.com/dokument-och-arendehantering/om/#arenden-offentlig-forvaltning

[5] *Offentlighetsprincipen*. Åtkomstdatum: 18 maj 2026. [Online]. Tillgänglig vid: https://www.regeringen.se/sa-styrs-sverige/grundlagar-och-demokratiskt-deltagande/offentlighetsprincipen/

[6] Region Jämtland Härjedalen, ”Diarium”. Åtkomstdatum: 19 mars 2026. [Online]. Tillgänglig vid: https://www.regionjh.se/politik/diarium

[7] Härjedalens kommun, ”Riktlinjer för ärendehantering”. 17 maj 2023. Åtkomstdatum: 11 mars 2026. [Online]. Tillgänglig vid: https://www.herjedalen.se/kommun-och-politik/styrdokument-och-regler/interna-styrdokument/riktlinjer-for-arendehantering.html

[8] Microsoft Azure, ”Vad är databaser?” Åtkomstdatum: 13 maj 2026. [Online]. Tillgänglig vid: https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-are-databases

[9] Microsoft Azure, ”Vad är SQL-databas?” Åtkomstdatum: 19 maj 2026. [Online]. Tillgänglig vid: https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-is-sql-database

[10] A. Olausson M., ”Vad är icke-funktionella krav?”, i *Kvalitetstårtan: en handbok för hållbara system*, Älvsjö: Amocon, 2026. Åtkomstdatum: 19 maj 2026. [Online]. Tillgänglig vid: https://andersmolausson.se/vad-ar-icke-funktionella-krav/

[11] DB-Engines, ”DB-Engines Ranking”. Åtkomstdatum: 26 mars 2026. [Online]. Tillgänglig vid: https://db-engines.com/en/ranking

[12] T. Padron-McCarthy och T. Risch, *Databasteknik*, 2:a uppl. Lund, Sverige: Studentlitteratur, 2018.

[13] K. Nilsson och C. Lindholm, ”databas”, *Nationalencyklopedin*. Åtkomstdatum: 07 april 2026. [Online]. Tillgänglig vid: https://www-ne-se.ezproxy.its.uu.se/uppslagsverk/encyklopedi/lång/databas

[14] T. Padron-McCarthy och T. Risch, ”Sammanfattning av SQL-kommandon”, i *Databasteknik*, Andra upplagan., Lund: Studentlitteratur, 2018, s. 195–211.

[15] MongoDB, ”MongoDB Supported Languages”. Åtkomstdatum: 26 maj 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/resources/languages

[16] G. Nicholas, ”MySQL”. Åtkomstdatum: 07 april 2026. [Online]. Tillgänglig vid: https://www.britannica.com/topic/MySQL

[17] MongoDB, ”What is a Document Database?” Åtkomstdatum: 11 mars 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/resources/basics/databases/document-databases

[18] MongoDB, ”Documents”. Åtkomstdatum: 07 april 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/docs/manual/core/document/

[19] MongoDB, ”Embedding MongoDB”. Åtkomstdatum: 18 maj 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/resources/products/fundamentals/embedded-mongodb

[20] MongoDB, ”What is MongoDB?” Åtkomstdatum: 07 april 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/docs/manual/

[21] MongoDB, ”Sharding”. Åtkomstdatum: 18 maj 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/docs/manual/sharding/

[22] H. Garcia-Molina, J. D. Ullman, och J. Widom, *Database systems: the complete book*, 2:a uppl. i Always learning. Harlow: Pearson, 2014.

[23] MongoDB, ”Transactions”. Åtkomstdatum: 27 maj 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/docs/manual/core/transactions/

[24] D. Ganesh Chandra, ”BASE analysis of NoSQL database”, *Future Gener. Comput. Syst.*, vol. 52, s. 13–21, nov. 2015, doi: 10.1016/j.future.2015.05.003.

[25] Nilex, ”Vad är modern ärendehantering 2026 - en komplett guide”. Åtkomstdatum: 11 mars 2026. [Online]. Tillgänglig vid: https://nilex.se/blogg/vad-ar-arendehanteringssystem-2026-en-komplett-guide/#elementor-toc__heading-anchor-1

[26] Sveriges riksdag, ”Allmänna handlingar”. Åtkomstdatum: 19 mars 2026. [Online]. Tillgänglig vid: https://www.riksdagen.se/sv/dokument-och-lagar/allmanna-handlingar/

[27] A. H. Al Hinai, ”A Performance Comparison of SQL and NoSQL Databases for Large Scale Analysis of Persistent Logs”, Masterexamen, Institutionen för informationsteknologi, Uppsala universitet, Uppsala, Sverige, 2016.

Åtkomstdatum: 26 mars 2026. [Online]. Tillgänglig vid: https://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-302304

[28] GeeksForGeeks, ”CRUD Full Form”. Åtkomstdatum: 30 mars 2026. [Online]. Tillgänglig vid: https://www.geeksforgeeks.org/websites-apps/crud-full-form/

[29] J. Hedman och M. Holmberg, ”Jämförelse av NoSQL-databas och SQL-baserad relationsdatabas: En förklarande studie för när NoSQL kan vara att föredra framför en relationsdatabas”, Kandidatuppsats, Institutionen för information och teknik, Högskolan Dalarna, 2018. Åtkomstdatum: 26 mars 2026. [Online]. Tillgänglig vid: https://urn.kb.se/resolve?urn=urn:nbn:se:du-29817

[30] J. Gustavsson, ”Jämförelse av relationsdatabaser och NoSQL-databaser: När kommunikation ska ske med en webbapplikation i ett odistribuerat system”, Masterexamen, Institutionen för informationsteknologi, Högskolan i Skövde, Skövde, Sverige, 2014. Åtkomstdatum: 26 mars 2026. [Online]. Tillgänglig vid: https://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-9524

[31] C. Elmhäll, ”Prestandajämförelse mellan NoSQL databaser för sjukhusdata: En jämförelse mellan MongoDB och Couchbase”, Kandidatuppsats, Institutionen för informationsteknologi, Högskolan i Skövde, 2022. Åtkomstdatum: 26 mars 2026. [Online]. Tillgänglig vid: https://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-21541

[32] M. Arsala, ”Processdokument om testdatahantering som ökar efterlevnaden av GDPR: En kvalitativ studie från testarnas perspektiv”, Examensarbete, Högskolan i Skövde, Skövde, Sverige, 2023. Åtkomstdatum: 31 mars 2026. [Online]. Tillgänglig vid: https://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-22904

[33] A. Bryman, *Samhällsvetenskapliga metoder*, 3:e uppl. Stockholm, Sverige: Liber, 2018.

[34] M. Winteringham, *Software testing with generative AI*. Shelter Island, NY, USA: Manning Publications Co, 2025.

[35] Region Jämtland Härjedalen, ”Region Jämtland Härjedalens webbdiarium”. Åtkomstdatum: 23 mars 2026. [Online]. Tillgänglig vid: https://diariet.regionjh.se/diariet/

[36] Internetmuseum, ”PUL är här - nu kan du inte skriva vad du vill om andra”, Den personliga integriteten i en digital värld. [Online]. Tillgänglig vid: https://internetmuseum.se/utstallningar/personlig-integritet/pul/

[37] GeeksForGeeks, ”CSV File Format | .csv Extension”. Åtkomstdatum: 19 mars 2026. [Online]. Tillgänglig vid: https://www.geeksforgeeks.org/data-analysis/csv-file-format/

[38] Region Stockholm, ”Diarium”. Åtkomstdatum: 16 april 2026. [Online]. Tillgänglig vid: https://www.diarium.regionstockholm.se/

[39] Region Kronoberg, ”Webbdiarium”. Åtkomstdatum: 16 april 2026. [Online]. Tillgänglig vid: https://webbdiarium.regionkronoberg.se/Home/DoSearchCases

[40] L. Wiktorin, *Utveckling av IT-system: krav, metod och arkitektur*, Första upplagan. Lund: Studentlitteratur, 2018.

[41] MySQL, ”10.3.1 How MySQL Uses Indexes”, Documentation. Åtkomstdatum: 08 maj 2026. [Online]. Tillgänglig vid: https://dev.mysql.com/doc/refman/8.4/en/mysql-indexes.html

[42] MongoDB, ”Indexes”, Development. Åtkomstdatum: 08 maj 2026. [Online]. Tillgänglig vid: https://www.mongodb.com/docs/manual/indexes/

Appendix A – Slutgiltig prompt för generering av syntetiskt testdataset

Generera ett testdataset som efterliknar ett ärendehanteringssystem för en svensk region. Datasetet ska bestå av två tabeller, vara nedladdningsbar fil i CSV-format och genereras utifrån reglerna nedan och instruktionerna under:

- # betecknar tabellens namn
- % betecknar kolumnernas namn
- Kolumnerna beskrivs med namn, datatyp och dataalternativ. Dessa separeras med ett lodstreck |.
- I fall där dataalternativet är slumpat, slumpa data baserat på kolumnnamnet och datatypen
- Enheter för akronym:
    - Landstingsstyrelsen
    - Revisorer
    - Regionala utvecklingsnämnden
    - Vårdvalsnämnden
    - Regionstyrelsen
    - Gemensamma nämnden för upphandling
    - Regionförbundet
    - Personalrekryteringsärenden
    - Gemensam nämnd för samverkan inom drift och service, utveckling samt specialistfunktioner
    - Hälso- och sjukvårdsnämnden
    - Patientnämnden
    - Kollektivtrafiknämnden
- Första ärendet en enhet får in under ett år har det nummer 001, ex. HSN/001/2025. om det inkommer ett till ärende till samma enhet får det då diarienumret HSN/002/2025

- mottagare/avsändare är i högsta grad företag eller kommunala organ som socialtjänst, ytterst få privatpersoner
- ca 20% av ärenden har mottagare/avsändare [skyddat]
- Varje ärende har mellan 0 och 6 händelser
- ärendetabellen ska ha 790 000 rader
- händelserna har diarienumret från ärendet som en främmande nyckel

Instruktioner:

Ärenden

% Diarienummer | textsträng | (akronym av relevant enhet)/XXX/ÅÅÅÅ

% Titel | textsträng | slumpad

% Avsändare/mottagare | textsträng | slumpad person eller företag

% Registrerat datum | datum | slumpad fr.o.m. 2006-01-01 till 2026-03-23

% Sekretess | boolean | true or false

% Status | textsträng | under beredning or avslutad

Händelser

% Diarienummer | textsträng | (akronym av relevant enhet)/XXX/ÅÅÅÅ

% Händelsenummer | textsträng | (akronym av relevant enhet)/XXX:X/ÅÅÅÅ, baserad på ärendets diarienummer

% Datum för händelse | datum | slumpad fr.o.m. inkommet datum t.o.m. 2026-03-23

% Titel på händelse | textsträng | slumpad

% Avsändare/mottagare | textsträng | slumpad, samma som för ärendet eller [skyddat]

Appendix B – Benchmarkqueries och testskript

B.1 Enkla uppslag

User story 1

Som medarbetare vill jag kunna se status för ett specifikt diarienummer för att snabbt kunna följa ärendets nuvarande handläggningsläge.

MySQL

```
DROP PROCEDURE IF EXISTS testa_lookup;

DELIMITER $$

CREATE PROCEDURE testa_lookup()

BEGIN

   DECLARE i INT DEFAULT 0;

   DECLARE status TEXT;

   DECLARE start_time DATETIME(6);

   DECLARE end_time DATETIME(6);

   SET start_time = NOW(6);

   WHILE i < 10000 DO

      SELECT arendestatus

      INTO status

      FROM ärenden

      WHERE diarienummer = 'GNU/547/2006';

      SET i = i + 1;

   END WHILE;

   SET end_time = NOW(6);

   SELECT

      TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
total_sekunder,

      TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
genomsnitt_ms;

END$$
```

```
DELIMITER ;
```

MongoDB

```
let iterations = 1000;
let status = null;
let start = new Date();
for (let i = 0; i < iterations; i++) {
  let result = db.arenden.findOne(
    { diarienummer: "GNU/547/2006" },
    { arendestatus: 1, _id: 0 }
  );
  if (result) {
    status = result.arendestatus;
  }
}
let end = new Date();
let totalMs = end - start;
printjson({
  status: status,
  total_sekunder: totalMs / 1000,
  genomsnitt_ms: totalMs / iterations
});
```

B.2 Filtrering

User story 2

Som handläggare vill jag kunna visa alla ärenden kopplade till en specifik nämnd för att kunna få en överblick över inkomna och pågående ärenden inom ansvarsområdet.

MySQL

```
DROP PROCEDURE IF EXISTS testa_ktn_10000;
DELIMITER $$
CREATE PROCEDURE testa_ktn_10000()
```

```
BEGIN
DECLARE i INT DEFAULT 0;
DECLARE antal INT DEFAULT 0;
DECLARE start_time DATETIME(6);
DECLARE end_time DATETIME(6);
SET start_time = NOW(6);
WHILE i < 10000 DO
SELECT COUNT(*)
INTO antal
FROM Arendehantering.ärenden USE INDEX (idx_arenden_diarienummer_status)
WHERE diarienummer LIKE 'KTN/%'
AND arendestatus = 'under beredning';
SET i = i + 1;
END WHILE;
SET end_time = NOW(6);
SELECT
antal,
TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
total_sekunder,
TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
genomsnitt_ms;
END$$
DELIMITER ;
```

MongoDB

```
let iterations = 1000;
let antal = 0;
let start = new Date();
for(let i = 0; i<iterations; i++){
  antal = db.arenden.countDocuments({
    diarienummer: /^KTN\//,
```

```
    arendestatus: "under beredning"
  });
}
let end = new Date();
let totalMs = end - start;
printjson({
  antal: antal,
  total_sekunder: totalMs/1000,
  genomsnitt_ms: totalMs/iterations
})
```

User story 3

Som behörig handläggare vill jag kunna filtrera fram pågående sekretessklassade ärenden för att hantera känsliga handlingar korrekt.

MySQL

```
DROP PROCEDURE IF EXISTS testa_status_sekretess;
DELIMITER $$
CREATE PROCEDURE testa_status_sekretess()
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE antal INT DEFAULT 0;
    DECLARE start_time DATETIME(6);
    DECLARE end_time DATETIME(6);
    SET start_time = NOW(6);
    WHILE i < 10000 DO
        SELECT COUNT(*)
        INTO antal
        FROM ärenden
        WHERE arendestatus = 'under beredning'
        AND sekretess = 1;
```

```
    SET i = i + 1;
  END WHILE;
  SET end_time = NOW(6);
  SELECT
    antal AS antal_matchande_rader,
    TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
total_sekunder,
    TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
genomsnitt_ms;
END$$
DELIMITER ;
```

MongoDB

```
let iterations = 1000;
let antal = 0;
let start = new Date();
for (let i = 0; i < iterations; i++) {
  antal = db.arenden.countDocuments({
    arendestatus: "under beredning",
    sekretess: true
  });
}
let end = new Date();
let totalMs = end - start;
printjson({
  antal_matchande_dokument: antal,
  total_sekunder: totalMs / 1000,
  genomsnitt_ms: totalMs / iterations
});
```

B.3 Intervallsökning

User story 4

Som chef vill jag kunna se alla ärenden registrerade under ett specifikt datumintervall för att följa arbetsbelastningen över tid.

MySQL

```
DROP PROCEDURE IF EXISTS testa_datum_range;

DELIMITER $$

CREATE PROCEDURE testa_datum_range()

BEGIN

   DECLARE i INT DEFAULT 0;

   DECLARE antal INT DEFAULT 0;

   DECLARE start_time DATETIME(6);

   DECLARE end_time DATETIME(6);

   SET start_time = NOW(6);

   WHILE i < 10000 DO

      SELECT COUNT(*)

      INTO antal

      FROM Arendehantering.ärenden

      WHERE registrerat_datum BETWEEN '2025-10-01' AND '2026-02-15';

      SET i = i + 1;

   END WHILE;

   SET end_time = NOW(6);

   SELECT

      antal AS antal_matchande_rader,

      TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
total_sekunder,

      TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
genomsnitt_ms;

END$$

DELIMITER ;
```

MongoDB

```
let iterations = 1000;
```

```
let antal = 0;
let start = new Date();
for (let i = 0; i < iterations; i++) {
  antal = db.arenden.countDocuments({
    registrerat_datum: {
      $gte: new Date("2025-10-01"),
      $lte: new Date("2026-02-15")
    }
  });
}
let end = new Date();
let totalMs = end - start;
printjson({
  antal_matchande_dokument: antal,
  total_sekunder: totalMs / 1000,
  genomsnitt_ms: totalMs / iterations
});
```

B.4 Aggregering

User story 5

Som verksamhetschef vill jag kunna se antal ärenden per nämnd för att följa resursfördelning och ärendemängd.

MySQL

```
DROP PROCEDURE IF EXISTS testa_groupby;
DELIMITER $$
CREATE PROCEDURE testa_groupby()
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE antal INT DEFAULT 0;
    DECLARE start_time DATETIME(6);
```

```
  DECLARE end_time DATETIME(6);
  SET start_time = NOW(6);
  WHILE i < 1000 DO
    SELECT COUNT(*) INTO antal
    FROM (
      SELECT
        SUBSTRING_INDEX(diarienummer,'/',1) AS namnd_prefix
      FROM ärenden
      GROUP BY namnd_prefix
    ) AS sub;
    SET i = i + 1;
  END WHILE;
  SET end_time = NOW(6);
  SELECt
    antal AS antal_grupper,
    TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
total_sekunder,
    TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 1000 AS
genomsnitt_ms;
END$$
DELIMITER ;
```

MongoDB

```
let iterations = 1000;
let antal = 0;
let start = new Date();
for (let i = 0; i < iterations; i++) {
 let result = db.arenden.aggregate([
  {
   $project: {
    namnd_prefix: {
```

```
          $arrayElemAt: [
            { $split: ["$diarienummer", "/"] },
            0
          ]
        }
      }
    },
    {
      $group: {
        _id: "$namnd_prefix"
      }
    },
    {
      $count: "antal_grupper"
    }
  ]).toArray();
  antal = result.length > 0 ? result[0].antal_grupper : 0;
}

let end = new Date();
let totalMs =end - start;
printjson({
  antal_grupper: antal,
  total_sekunder: totalMs / 1000,
  genomsnitt_ms: totalMs / iterations
});
```

B.5 Relationsbaserade operationer

User story 6

Som handläggare vill jag kunna se alla händelser kopplade till ett specifikt ärende för att följa dess historik.

MySQL

```
DROP PROCEDURE IF EXISTS testa_handelser;

DELIMITER $$

CREATE PROCEDURE testa_handelser()

BEGIN

	DECLARE i INT DEFAULT 0;

	DECLARE antal INT DEFAULT 0;

	DECLARE start_time DATETIME(6);

	DECLARE end_time DATETIME(6);

	SET start_time = NOW(6);

	WHILE i < 10000 DO

		SELECT COUNT(*)

		INTO antal

		FROM händelser

		WHERE diarienummer = 'PRA/1588/2024';

		SET i = i + 1;

		END WHILE;

	SET end_time = NOW(6);

	SELECT antal AS antal_rader,

	TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
	total_sekunder,

	TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
	genomsnitt_ms;

END$$

DELIMITER ;
```

MongoDB

```
let iterations = 1000;

let antal = 0;
```

```
let start = new Date();
for (let i = 0; i < iterations; i++) {
  let doc = db.arenden.findOne(
    { diarienummer: "PRA/1588/2024" },
    { handelser: 1, _id: 0 }
  );
  antal = doc.handelser.length;
}
let end = new Date();
let totalMs = end - start;
printjson({
  antal_handelser: antal,
  total_sekunder: totalMs / 1000,
  genomsnitt_ms: totalMs / iterations
});
```

B.6 Skrivoperationer

User story 7

Som handläggare vill jag kunna uppdatera status på ett ärende för att spegla det aktuella handläggningsläget.

MySQL

```
DROP PROCEDURE IF EXISTS testa_update;
DELIMITER $$
CREATE PROCEDURE testa_update()
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE status VARCHAR(50);
    DECLARE start_time DATETIME(6);
    DECLARE end_time DATETIME(6);
    SET start_time = NOW(6);
```

```
    WHILE i < 10000 DO
        SELECT arendestatus INTO status
        FROM ärenden
        WHERE diarienummer = 'RF/699/2022';
        IF status = 'avslutat' THEN
            UPDATE ärenden
            SET arendestatus = 'under beredning'
            WHERE diarienummer = 'RF/699/2022';
        ELSE
            UPDATE ärenden
            SET arendestatus = 'avslutat'
            WHERE diarienummer = 'RF/699/2022';
        END IF;
        SET i = i + 1;
    END WHILE;
    SET end_time = NOW(6);
    SELECT
        TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
total_sekunder,
        TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
genomsnitt_ms;
END$$
DELIMITER ;
```

MongoDB

```
let iterations = 1000;
let diarienummer = "RF/699/2022";
let start = new Date();
for (let i = 0; i < iterations; i++) {
  let result = db.arenden.findOne(
    { diarienummer: diarienummer },
```

```
    { arendestatus: 1, _id: 0 }
  );
  if (result.arendestatus === "avslutat") {
    db.arenden.updateOne(
      { diarienummer: diarienummer },
      { $set: { arendestatus: "under beredning" } }
    );
  } else {
    db.arenden.updateOne(
      { diarienummer: diarienummer },
      { $set: { arendestatus: "avslutat" } }
    );
  }
}
let end = new Date();
let totalMs = end - start;
printjson({
  total_sekunder: totalMs / 1000,
  genomsnitt_ms: totalMs / iterations
});
```

User story 8

Som registrator vill jag kunna lägga till ett nytt ärende i systemet för att säkerställa att inkommande handlingar registreras och kan följas upp.

MySQL

```
DROP PROCEDURE IF EXISTS testa_insert;
DELIMITER $$

CREATE PROCEDURE testa_insert()
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE nytt_diarienummer VARCHAR(50);
```

```
DECLARE start_time DATETIME(6);
DECLARE end_time DATETIME(6);

SET start_time = NOW(6);

WHILE i <= 10000 DO
SET nytt_diarienummer = CONCAT('TEST/', LPAD(i, 6, '0'), '/2026');

INSERT INTO ärenden (
diarienummer,
titel,
avsandare_mottagare,
registrerat_datum,
sekretess,
arendestatus
)
VALUES (
nytt_diarienummer,
'Testärende för insertmätning',
'Testmyndigheten',
'2026-03-23',
false,
'under beredning'
);

SET i = i + 1;
END WHILE;

SET end_time = NOW(6);

SELECT
10000 AS antal_insert_operationer,
TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000000 AS
total_sekunder,
TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
genomsnitt_ms;
END$$

DELIMITER ;
```

MongoDB

```
let iterations = 10000;

let start = new Date();
```

```
for (let i = 1; i <= iterations; i++) {
  db.arenden.insertOne({
    diarienummer: "TEST/" + String(i).padStart(6, "0") + "/2026",
    titel: "Testärende för insertmätning",
    avsandare_mottagare: "Testmyndigheten",
    registrerat_datum: new Date("2026-03-23"),
    sekretess: false,
    arendestatus: "under beredning"
  });
}
let end = new Date();
let totalMs = end - start;

printjson({
  tid_ms_genomsnitt: totalMs / iterations,
  tid_sekunder: totalMs / 1000
});
```

User story 9

Som administratör vill jag kunna ta bort eller arkivera felregistrerade ärenden för att upprätthålla datakvalitet.

MySQL

```
DROP PROCEDURE IF EXISTS testa_delete;
DELIMITER $$

CREATE PROCEDURE testa_delete()
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE test_diarienummer VARCHAR(50);
DECLARE start_time DATETIME(6);
DECLARE end_time DATETIME(6);

SET start_time = NOW(6);

WHILE i <= 10000 DO
```

```
SET test_diarienummer = CONCAT('TEST/', LPAD(i, 6, '0'), '/2026');

DELETE FROM ärenden
WHERE diarienummer = test_diarienummer;

SET i = i + 1;
END WHILE;

SET end_time = NOW(6);

SELECT
10000 AS antal_delete_operationer,
TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 AS total_tid_ms,
TIMESTAMPDIFF(MICROSECOND, start_time, end_time) / 1000 / 10000 AS
genomsnitt_ms;
END$$

DELIMITER ;
```

MongoDB

```
let iterations = 10000;

let start = new Date();

for (let i = 1; i <= iterations; i++) {
  let testDiarienummer = "TEST/" + String(i).padStart(6, "0") + "/2026";
  db.arenden.deleteOne({
    diarienummer: testDiarienummer
  });
}

let end = new Date();

let totalMs = end - start;

printjson({
  antal_delete_operationer: iterations,
  total_tid_ms: totalMs,
  medeltid_per_delete_ms: totalMs / iterations
});
```