T.C.
İSTANBUL BEYKENT ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

ÖNBELLEK TABANLI, BELGE TABANLI, KOLON TABANLI VERİ TABANLARI VE İLİŞKİSEL VERİ TABANLARININ PERFORMANS KARŞILAŞTIRMASI VE OPTİMİZASYONU

Yüksek Lisans Tezi

Tezi Hazırlayan
Muhammet Erol YİGİN

İstanbul, 2025

T.C.
İSTANBUL BEYKENT ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

ÖNBELLEK TABANLI, BELGE TABANLI, KOLON TABANLI VERİ TABANLARI VE İLİŞKİSEL VERİ TABANLARININ PERFORMANS KARŞILAŞTIRMASI VE OPTİMİZASYONU

Yüksek Lisans Tezi

Tezi Hazırlayan
Muhammet Erol YİGİN

Öğrenci No
2320003045

ORCID ID
0009-0002-3702-2169

Danışman
Dr. Öğr. Üyesi Aykut GÜVEN

İstanbul, 2025

YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum **‘‘Önbellek Tabanlı, Belge Tabanlı, Kolon Tabanlı Veri Tabanları ve İlişkisel Veri Tabanları Performans Karşılaştırması ve Optimizasyonu”** başlıklı bu çalışmanın, bilimsel ahlak ve geleneklere uygun şekilde tarafımdan yazıldığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını, patent ve telif haklarını ihlal edici bir davranışımın olmadığını belirtir ve bunu onurumla doğrularım. 30/06/2025

Muhammet Erol YİGİN

T.C.
İSTANBUL BEYKENT ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ MÜDÜRLÜĞÜ
TEZLİ YÜKSEK LİSANS SINAV TUTANAĞI

16/06/2025

Enstitümüz ***Bilgisayar Mühendisliği*** Anabilim Dalı ***Bilgisayar Mühendisliği*** Programı yüksek lisans öğrencilerinden ***2320003045*** numaralı ***Muhammet Erol YİĞİN***'in *"İstanbul Beykent Üniversitesi Lisansüstü Eğitim – Öğretim Yönetmeliği"*nin ilgili maddesine göre hazırlayarak, Enstitümüze teslim ettiği ***"Önbellek Tabanlı, Belge Tabanlı, Kolon Tabanlı Veri Tabanları Ve İlişkisel Veri Tabanlarının Performans Karşılaştırması Ve Optimizasyonu"*** konulu tezini, Yönetim Kurulumuzun 20/05/2025 tarih ve 2025/22 sayılı toplantısında seçilen ve Taksim yerleşkesinde toplanan biz jüri üyeleri huzurunda, İstanbul Beykent Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 29. maddesinin 3. fıkrası gereğince aday tarafından savunulmuş ve sonuçta adayın tezi hakkında ***"OYBİRLİĞİ"*** ile ***"KABUL"*** kararı verilmiştir.

İşbu tutanak, 2 nüsha olarak hazırlanmış ve Enstitü Müdürlüğü'ne sunulmak üzere tarafımızdan düzenlenmiştir.

DANIŞMAN
Dr. Öğr. Üyesi Ay*** GÜ***
(İstanbul Beykent Üniversitesi)

ÜYE
Dr. Öğr. Üyesi Se*** KI***
(İstanbul Beykent Üniversitesi)

ÜYE
Dr. Öğr. Üyesi Ya*** KA***
(Doğuş Üniversitesi)

Adı ve Soyadı : Muhammet Erol YİGİN

Danışmanı : Dr. Öğr. Üyesi Aykut GÜVEN

Derecesi ve Tarihi : Yüksek Lisans (Tezli), 2025

Alanı : Bilgisayar Mühendisliği

Anahtar Kelimeler : Veri Tabanı, Önbellek Bazlı Veri Tabanı, Sütun Bazlı Veri Tabanı, İlişkisel Veri Tabanı, Belge Bazlı Veri Tabanı, Optimizasyon, Performans

ÖZ

ÖNBELLEK TABANLI, BELGE TABANLI, KOLON TABANLI VERİ TABANLARI VE İLİŞKİSEL VERİ TABANLARI PERFORMANS KARŞILAŞTIRMASI VE OPTİMİZASYONU

Bu çalışmada önbellek tabanlı, belge tabanlı, sütun tabanlı ve ilişkisel veri tabanı mimarileri performans karşılaştırması için seçilmiştir. Gerçek dünya sorgu satırları, donanım ve yazılım konfigürasyonları özdeş sanal makine kümelerinde; küçük, orta, büyük ve çok büyük veri hacimlerinde okuma ve karmaşık sorgu tipleri yürütülmüştür.. Özel otomasyon betikleri, testlerin tutarlılık ve tekrarlanabilirliğini sağlayarak veri toplama, yürütme ve raporlama süreçlerini tamamen otomatikleştirmiştir. Toplanan metrikler yürütme süreleri, gecikme ve kaynak kullanımı tablo ve grafiklerle görselleştirildikten sonra veri hacmi ve sorgu yapısına bağlı performans davranışları analiz edilmiştir. Bulgular, bazı mimarilerin belirli koşullarda üstün performans, bazılarının ise ölçek büyüdükçe darboğaz yaşadığını ortaya koymuştur. Her mimarinin güçlü ve zayıf yönlerine yönelik özgün optimizasyon önerileri sunulmuştur. Bu analiz, veri tabanı seçiminde performans odaklı karar süreçlerine rehberlik sağlamaktadır. Çalışma, farklı uygulama senaryolarındaki performans gereksinimlerine yönelik mimari tercihlerde yol göstericidir.

Name and Surname : Muhammet Erol YİGİN

Supervisor : Asst. Prof. Dr. Aykut GÜVEN

Degree and Date : Master's (Thesis), 2025

Major : Computer Engineering

Keywords : Database, Cache-Based Database, Column-Based Database, Relational Database, Document-Based Database, Optimization, Performance

ABSTRACT

PERFORMANCE COMPARISON AND OPTIMATIZATION OF CACHE BASED, DOCUMENT-BASED, COLUMN BASED AND RELATIONAL DATABASES

In this study, cache-based, document-based, column-based and relational database architectures were selected for performance comparison. Real world query rows, hardware and software configurations were identical in virtiucal machine clusters; small, medium large and very large data volumes were read and complex query types were executed. Custom automation scripts completely automated the data collection, execution and reporting processes, ensuring consistency and repeatability of the tests. The collected metrics, execution times, latency and resource usage were visualized with tables and graphs, and then performance behaviors depending on the data volume and query structure were analyzed. The findings revealed that some architectures provided superior performance under certaion conditions, while others experienced bottlenecks as the sclae grew. Unique optimization suggestions were presented for the strengths and weaknesses of each architecture. This analysis provides guidance for performance-oriented decision-making processes in database selection. The study guides architectural preferences for performance requirements in different application scenarios.

İÇİNDEKİLER

TABLOLAR LİSTESİ

Sayfa No.

ŞEKİLLER LİSTESİ

Sayfa No.

KISALTMALAR

ACID : Atomicity (Bütünlük), Consistency (Tutarlılık), Isolation (Yalıtım), Durability (Dayanıklılık)

AWS : Amazon Web Services (Amazon Web Servisleri)

BSON : Binary JSON (İkili Değerli JSON Belgesi)

CPU : Central Processing Unit (Merkezi İşlem Birimi)

DBMS : Database Management System (Veri Tabanı Yönetim Sistemi)

EC2 : Elastic Compute Cloud (Esnek Hesaplama Bulutu)

EBS : Elastic Block Store (Esnek Blok Depolama)

ETL : Extact, Transform, Load (Çıkar, Dönüştür, Yükle)

JSON : Javascript Object Notation (Javascript Nesne Gösterimi)

MSSQL : Microsoft SQL Server

NOSQL : Not Only Structured Query Language (İlişkisel Olmayan Veri Tabanı)

OLAP : Online Analytical Processing (Çevrimiçi Analitik İşleme)

RAM : Random Access Memory (Rastgele Erişimli Bellek)

RDBMS : Relational Database Management System (İlişkisel Veri Tabanı Yönetim Sistemi)

REDIS : Remote Dictionary Server (Uzak Sözlük Sunucusu)

SQL : Structured Query Language (Yapılandırılmış Sorgu Dili)

XML : Extensible Markup Language (Genişletilebilir Biçimlendirme Dili)

YCSB : Yahoo Cloud Serving Benchmark (Yahoo Bulut Hizmet Kıyaslaması)

vd. : ve diğerleri

vb. : ve benzeri

SÖZLÜK

ACID: Bir veri tabanı işleminin bütünlük (Atomicity), tutarlılık (Consistency), izolasyon (Isolation) ve dayanıklılık (Durability) özelliklerini garanti eden standarttır.

AWS EC2: Amazon Web Services ekosisteminde ölçeklenebilir sanal sunucular sağlayan bulut hizmeti; bu tezde test ortamı olarak kullanılmıştır.

BASE: NoSQL sistemlerinin "Basically Available, Soft state, Eventually consistent" yaklaşımını ifade eder; yüksek yatay ölçeklenebilirliğe vurgu yapar.

Belge Tabanlı Veri Tabanı: JSON / BSON benzeri belgeleri şemasız biçimde saklayan, esnek yapılı NoSQL veri tabanı türü (örn. MongoDB).

Dikey Ölçeklenebilirlik: Tek sunucunun CPU, RAM veya disk kaynaklarını artırarak performans büyütme yöntemidir; ilişkisel veri tabanlarında yaygındır.

İlişkisel Veri Tabanı: Verileri satır-sütun tablolarda saklayıp ACID kurallarını izleyen geleneksel model.

İndeks: Sorgu hızını artımrak için tablodaki sütunlara ait ilave veri yapısı; ClickHouse'ta Skip Index veya Order By yapılarıyla optimize edilir.

Kaggle: NYC Yellow Taxi Data veri seti gibi açık veri kümelerinin paylaşıldığı platform; deneye esas ham veriler buradan türetilmiştir.

MergeTree: ClickHouse'un parçalı saklama, sıkıştırma ve sıralama özelliklerini yöneten tablo motoru ailesi.

OLAP: "Online Analytical Processing"; sütun-tabanlı mimarilerde çok boyutlu analitik sorguları milisaniyelerle yanıtlamaya odaklı yaklaşım.

Önbellek Tabanlı Veri Tabanı: Veriyi RAM'de tutarak milisaniyelik erişim sağlayan anahtar-değer sistemleri (örn. Redis).

Replication (Replikasyon): Verinin birden fazla düğümde eş zamanlı kopyalanması; arıza anında kesintisiz devamlılık sağlar.

Sharding (Yatayda Parçalama): Büyük veri kümelerini anahtar aralıklarına göre sunucular arasında bölerek yatay ölçeklenebilirliği artırma tekniği.

Sütun Tabanlı Veri Tabanı: Veriyi sütun odaklı saklayıp sadece istenen sütunları okuyarak analitik sorgularda yüksek hız sunar (örn. ClickHouse).

Yatay Ölçeklenebilirlik: Sisteme yeni düğümler ekleyerek kapasiteyi artırma yaklaşımı; NoSQL sistemlerinin temel avantajıdır.

GİRİŞ

Günümüzde veri, belirlenen hedeflere ulaşmak için kullanılan bir varlık olarak görülmektedir. Verinin boyutu arttıkça erişim süresi de artmaktadır. Verinin erişilebilirliği için veri yönetimi gibi süreçler sağlanmakta olup, bu süreçler günümüz bilgi teknolojileri alanında oldukça önemlidir. Veri yönetimini kolaylaştırmak için veri tabanları oluşturulmuştur. Veri tabanı sistemleri, bilgisayar ortamında büyük, organize veri kümelerini yönetmek için yaygın olarak kullanılan bir araçtır. Veri tabanı genellikle bir veri tabanı yönetim sistemi (DBMS) tarafından kontrol edilir (Yarımağan, 2016, s. 44).

Günümüzde operasyonda kullanılan en yaygın veri tabanı türlerindeki veriler genellikle veri işleme ve sorgulamayı verimli hale getirmek için bir dizi tabloda satırlar ve sütunlar veya koleksiyonlar halinde tutulmaktadır. Bu yönetim sistemlerinin özellikleri de veri tabanı türlerine bağlı olarak değişmektedir. Veri tabanları dört farklı mimariye ayrılmaktadır. Bu mimariler önbellek tabanlı veri tabanları, belge tabanlı veri tabanları, sütun tabanlı veri tabanları ve ilişkisel veri tabanları olarak karşımıza çıkmaktadır.

Çalışmanın amacı, bu dört veri tabanını performans açısından karşılaştırmak ve veri tabanlarında uzmanlaşmak isteyen şirketlere, yazılım geliştiricilerine, öğrencilere ve bireylere performans optimizasyonu konusunda rehberlik etmektir. Aynı zamanda büyük ölçekli sistem uygulamalarında veri tabanı seçimi konusunda bilgi sunmaktır.

Önbellek tabanlı, belge tabanlı, sütun tabanlı ve ilişkisel veri tabanları, belirli veri işleme gereksinimlerini karşılamak ve verileri depolamak için tasarlanmış çeşitli veri tabanı türleri arasındadır. Bu mimariler için tasarlanan veri tabanları performans açısından önemli iyileştirmelerle geliştirilmiştir.

İlişkisel veri tabanları onlarca yıldır modern bilgi işlem uygulamalarının kalesi olmuştur. ACID özellikleri, ilişkisel veri tabanlarını neredeyse tüm veri yönetim sistemleri için bir çözüm haline getirmiştir. Ancak, özellikle büyük veri ve web ölçekli sistemlerde veri işleme ihtiyacı birçok NoSQL veri tabanının oluşturulmasına yol açmıştır (Lourenço vd., 2015, s. 1).

Önbellek tabanlı veri tabanları verileri doğrudan bellekte depolamakta ve hızlı erişim gerektiren sistemlerde yaygın olarak kullanılmaktadır. Bu veri tabanlarına anahtar-değer sistem kümeleri de denilmektedir. Her bir işleme düğümü, anahtar-değer sistemlerini karma (hashing) yöntemiyle neredeyse eşit şekilde dağıtabilmektedir. Farklı sunucular, istemci isteklerini aynı anda minimum performans kesintisiyle karşılamaktadır. Disk için optimize edilmiş anahtar-değer sistemleri, tüm uygulama verilerini yönetmek için dahili bellek veri yapılarını kullanmakta ve tahsis edilen fiziksel belleğin boyutunu aşan büyük veri kümelerini işlemek için akıllıca sanal bellekten yararlanmaktadır. Redis, Memcached, Apache Ignite, Amazon ElastiCache, Hazelcast gibi veri tabanları önbellek mimarisini kullanmaktadır (Cao vd., 2016, s. 1).

Belge tabanlı veri tabanları, yapısal esnekliğe sahip verileri depolamak için önemli hale gelmiştir. Belge tabanlı veri tabanları, yapılandırılmamış veya yarı yapılandırılmış verilerle karakterize edilen birçok uygulama alanında kullanılmaktadır. JSON veya BSON gibi belge biçimlerini destekleyen veri tabanları, veri depolamada esneklik sağlar ve hızlı geliştirme süreçleri için uygun bir çerçeve sunar. Sabit şema zorunluluğu olmayan belge tabanlı veri tabanları, yapılandırılmamış veri depolama gereksinimlerinin büyüyen sorununa bir cevap olarak kullanılabilmektedir. Ancak, bu veri tabanlarında depolanan belgelerden bilgi almak için yapılandırılmamış metinsel içeriği hızlı ve kolay bir şekilde işleme teknikleri eksiktir. MongoDB, CouchDB, Firebase Firestore, Amazon DocumentDB, ArangoDB, MarkLogic gibi veri tabanları belge tabanlı mimariyi kullanmaktadır (Jose ve Abraham, 2023, s. 1777).

Sütun tabanlı veri tabanları, veri işleme açısından önemli avantajlar sunmaktadır. Verileri sütun düzeyinde yapılandırarak depolama ve erişime olanak tanıyan bu veri tabanları, büyük veri analitiği, veri madenciliği ve raporlama gibi yüksek hacimli veri işleme faaliyetleri, verileri diskteki sütunlarda ayrı ayrı depolayan bir tür veri tabanı yönetim sistemidir. ClickHouse, Apache Hbase, Apache Cassandra, Google BigTable, Amazon RedShift gibi veri tabanları sütun tabanlı mimariyi kullanmaktadır (Dwivedi vd., 2012, s. 31).

İlişkisel veri tabanları, verileri tablolarda düzenleyerek veri bütünlüğünü sağlamak ve yönetmek için ilişkisel modeller kullanan geleneksel bir veri tabanı türüdür. Veri tabanı modeli, ilişkisel veri tabanı olarak bilinen ilişkisel model yaklaşımına dayanmaktadır. Veri tabanı modeli yapılandırılmış veri tablolar veya birleşimler arasındaki ilişkiler şeklinde depolar. Veriler önceden tanımlanmış tablolara veya yapılara uymalıdır. Dikey ölçeklenebilirlik, CPU, RAM ve sabit disk gibi donanımların yeteneklerini artırmaktadır (Faraj vd., 2014, s. 11).

Bu çalışma, yukarıda belirtilen dört veri tabanı türünün performansının kapsamlı bir analizini yapmaktadır. Araştırma, farklı veri türleri ve uygulamalar genelinde veri tabanlarının etkinliğini değerlendirmektedir. Performans karşılaştırması, veri erişim hızlarını karşılaştırmaktadır. Ek olarak, her veri tabanı türüne özgü performans iyileştirme çözümleri incelenmekte ve veri tabanı bakımında verimliliği artırmak için öneriler sunmaktadır.

Bu çalışmanın bulguları, çeşitli veri tabanı türlerinin güçlü ve zayıf yönlerini ortaya çıkarmakta ve hangisinin belirli iş ortamlarına en uygun olduğunu göstermektedir. Bu araştırma, veri tabanı yönetiminde en iyi teknikleri belirlemek ve büyük veri analitiği, gerçek zamanlı uygulamalar ve dağıtılmış sistemler gibi yüksek performans gerektiren alanlar için etkili çözümler sağlamak amacıyla yürütülmektedir.

Bu çalışma, işletmelerin verileri stratejik olarak kullanmasını sağlamak için çağdaş veri tabanı teknolojisinin etkinliğini analiz etmeyi amaçlamaktadır. Günümüzün teknolojik ortamında, veri tabanı yönetiminde yetkin olmak, kuruluşların rekabet gücünü artırmada rehber olmayı amaçlamaktadır. Veri yönetiminde en iyi uygulamaları belirlemek ve daha etkili veri tabanı çözümleri formüle etmek için önemli bir referans olmayı hedeflemektedir.

1. MİMARİ ve TUTARLILIK ÜZERİNE YAPILAN LİTERATÜR TARAMASI

Veri tabanı dünyasında tutarlılık ve mimari tercihler arasındaki dengeyi anlamak için temel modeller ve optimizasyon yaklaşımları ele alınmıştır. ACID ve BASE prensipleri ışığında, ilişkisel sistemlerin sabit şema ve dikey ölçeklenebilirlik kısıtlarına karşın sağladığı güçlü tutarlılık garantileri ile NoSQL çözümlerinin yatay ölçeklenebilirlik ve esneklik avantajları karşılaştırılmıştır. Ayrıca, MSSQL Server veri tabanı üzerinde indeks tasarımı ve Dinamik Yönetim Görünümleri aracılığıyla yürütülen optimizasyon çalışmaları, gerçek iş yükleri altında veri tabanı kararlılığının ve performansının nasıl sürdürebilir kılınabileceğini göstermektedir.

1.1. ACID ve BASE Modelleri

İlişkisel veri tabanları ACID prensipleri aracılığıyla veri tutarlılığını ve bütünlüğünü garanti etmektedir. Bu yapı karmaşık sorguların doğru sonuçlar üretmesini ve işlemler arası kesişmelerin önlenmesini sağlamaktadır. Ancak sabit şema gereksinimi ve dikey ölçeklenebilirlik sınırları, büyük veri ortamlarında performans darboğazlarına yol açabilmektedir (Gökşen ve Aşan, 2015, s. 125).

NoSQL veri tabanları BASE prensipleri çerçevesinde yatay ölçeklenebilirlik ve esneklik sunmaktadır. Şemasız veri modelleri ile otomatik parçalama (sharding) ve çoğaltma (replication) mekanizmaları yüksek okuma yazma hızları sağlamaktadır. Bu sayede düşük gecikme süreleri ve yüksek erişilebilirlik elde edilmektedir. Hibrit RDBMS – NoSQL mimarilerinin tutarlılık ve performans dengesini optimize ettiği vurgulanmıştır (Gökşen ve Aşan, 2015, s. 126).

1.2. Veri Tabanı Optimizasyonu

MSSQL Server'da sorgu performansını iyileştirmek için doğru indeks stratejileri kritik öneme sahiptir. Clustured ve non-clustured indekslerin B-Tree yapıları üzerinden yürütme planlarına etkileri incelenmiş; yanlış veya eksik indekslerin sorgu sürelerini anlamlı ölçüde uzattığı gösterilmiştir (Arıcı, 2017, s. 4).

Dinamik Yönetim Görünümleri ve Fonksiyonları (DMV(DMF) kullanımı, eksik, kullanılmayan ve parçalanmış indekslerin tespit edilmesini sağlamaktadır.

Özellikle sys.dm-db_missing_index_details, sys.dm_db_index_usage_stats ve sys.dm_db_index_physical_stats gibi görünümler; otomatik metriklerle bakım ve yeniden düzenleme stratejilerinin belirlenmesine olanak tanıyarak uzun vadede veri tabanı kararlılığını ve verimliliğini artırmaktadır (Arıcı, 2017, s. 47).

2. PERFORMANS ve ÖLÇEKLENEBİLİRLİK ÜZERİNE YAPILAN LİTERATÜR TARAMASI

Büyük veri analitik iş yüklerinde öne çıkan veri tabanı mimarilerinin performans ve ölçeklenebilirlik özellikleri incelenmektedir. Öncelikle sütunlu depolama ve denormalize modellerin ETL süreçleri ve karmaşık sorgular üzerinde hız avantajları tartışılmış; ardından yatay ölçeklenebilirlik, otomatik parçalama ve replikasyon mekanizmalarının NoSQL sistemlerine sağladığı düşük gecikme ve yüksek erişilebilirlik yetenekleri ele alınmıştır. Benchmark yaklaşımları ile gerçek dünya iş yüklerine göre farklı çözümlerin nasıl karşılaştırıldığı ve donanım yazılım konfigürasyonlarının performans sonuçlarının nasıl şekillendirileceği özetlenmiştir.

2.1. Analitik Veri Tabanları

Analitik veri tabanlarının performansını artırmak amacıyla, sütunlu veri tabanı teknolojisini denormalize edilmiş veri modelleriyle birleştiren yenilikçi bir yaklaşım önerilmiştir. Bu yöntem, özellikle büyük veri analitiği ve gerçek zamanlı karar destek sistemlerinde kritik rol oynayan ETL süreçlerinin ve analitik sorguların verimliliğini yükseltmeyi hedeflemektedir (Jukic vd., 2017, s. 61). Sütunlu depolama yapısı satır tabanlı modellere kıyasla yüksek veri sıkıştırma oranları sunmakta ve bellek içi işlem performansını önemli ölçüde iyileştirmektedir.

Denormalizasyonun veri modelleme karmaşıklığını azalttığını ve geniş kapsamlı sorgularda hız kazandırdığı belirtilmiştir (Jukic vd., 2017, s. 61). Bu sayede, birleşik sütunlu denormalize mimarinin veri filtreleme, toplama ve birleştirme işlemlerinde hem disk I/O maliyetlerini düşürdüğünü hem de sorgu yürütme sürelerini kısalttığını göstermiştir. Sonuç olarak, önerilen yaklaşım karar destek süreçlerinde anlık içgörü elde etmeyi kolaylaştırarak sistem ölçeklenebilirliğini ve tepki sürelerini optimize etmektedir.

2.2. NoSQL ve Dağıtık Sistemler

Literatürde yapılan karşılaştırmalar, NoSQL çözümlerinin esnek şemasız yapıları sayesinde gerçek zamanlı iş yüklerinde ilişkisel modellere kıyasla düşük gecikme ve yüksek işlem hacmi sunduğunu, özellikle MongoDB ve Cassandra gibi

sistemlerin, büyük küme konfigürasyonlarında milyonlarca işlemi eş zamanlı yöneterek performans üstünlüğü sağladığını göstermektedir.

2.2.1. Performans Karşılaştırmaları

İlişkisel ve ilişkisel olmayan (NoSQL) veri tabanı yönetim sistemlerinin performansını analiz edilmiştir. Çalışma, büyük verileri ve karmaşık sorguları etkili bir şekilde yönetmek için bir veri tabanının seçiminin kritik olduğunu vurgulamaktadır. Literatür, NoSQL veri tabanlarının esnekliğini, yatay ölçeklenebilirliğini ve performans avantajlarını not ederken, ilişkisel veri tabanları veri bütünlüğünü ve tutarlılığını sağlamak için değerlidir (Öztürk ve Atmaca, 2017, s. 199).

Tablo tabanlı yapıları ve dikey ölçeklenebilirlikleri ile ilişkisel veri tabanlarının (SQL) veri bütünlüğünü ve tutarlılığını güçlü bir şekilde koruduğu; ancak hiyerarşik ve yapılandırılmamış veri modelleri için kısıtlı kaldığı vurgulanmıştır (Ali, 2018, s. 5). Öte yandan NoSQL veri tabanları, yatay ölçeklenebilirlik sayesinde kümeleme (sharding) ve çoğaltma (replication) mekanizmalarıyla veri işlem hacmini kolayca artırabilmekte; şemasız yapılarıyla esnek veri modellerine olanak tanıyarak büyük veri ve gerçek zamanlı uygulamalarda düşük gecikme süresi ve yüksek işlem hızı sunmaktadır.

MongoDB ve Cassandra'ya odaklanarak NoSQL veri tabanlarının performans ve güvenlik özelliklerini karşılaştırılmıştır. Çalışma, bu iki sistemin tek düğümlü ve çok düğümlü yapılandırmalardaki performansını ölçmek için YCSB aracını kullanılmıştır. Sonuçlar, MongoDB'nin tek düğümlü ortamlarda daha iyi performans gösterdiğini, Cassandra'nın ise büyük veri kümeleriyle çok düğümlü testlerde üstün olduğunu göstermektedir (Shwaysh, 2018, s. 6).

2.2.2. Ölçeklenebilirlik ve Dağıtık Teknikler

Yükse performans için öncelikle performans metriklerinin toplanması ve eşik değerlerinin (threshold) belirlenmesi gerekliliğini vurgulamaktadır. Bu amaçla Oracle'ın Automatic Database Diagnostic Monitor ile toplanan istatistikler yorumlanarak sistem yapılandırılmalarında donanım ve bellek ayarlarının optimize edilmesi önerilmektedir (Yorulmaz, 2018, s. 22). Yüksek erişilebilirlik tarafında ise aktif-aktif ve aktif-pasif mimariler; farklı lokasyonlarda yedeklenmiş, yıllık "switch-

over” testleriyle doğrulanmış altyapıların kritik uygulamalar için tercih edilmesi gerektiği ortaya koyulmuştur (Yorulmaz, 2018, s. 34).

İlişkisel ve NoSQL veri tabanlarının toplu veri işlemlerindeki performansını karşılaştırmıştır. MySQL, MongoDB ve Cassandra ile yapılan testler, MongoDB'nin toplu veri işlemlerinde daha hızlı sonuçlar verdiğini, MySQL'in ise gruplama ve silme işlemlerinde üstün olduğunu ortaya koymuştur (Seçgin, 2018, s. 48).

SQL Server 2017'de sorgu performansı iyileştirme tekniklerini ayrıntılı olarak açıklamıştır. Çalışma, sorgu yürütme planlarını analiz etmeye, dizin yapılarını ve sorgu tasarımını optimize etmeye odaklanmıştır. Literatür, veri tabanı performansını olumlu yönde etkilemek için doğru dizinleri seçmenin ve sorgu planlarını optimize etmenin önemini vurgulamaktadır (Fritchey, 2018, s. 4).

Veri tabanı mimarileri ve yönetim sistemleri kavramlarını tanımlayarak büyük veri ve NoSQL paradigmalarını ele almaktadır. İlişkisel veri tabanı mimarisi başta olmak üzere düz dosya, hiyerarşik ve ağ modelleri incelendikten sonra Veri Tabanı Yönetim Sistemleri tanıtılmış, ardından büyük veri kavramının üç temel bileşeni olan hacim, çeşitlilik ve hız sunulmuştur. PostgreSQL’in mimarisi ve özellikleri ayrıntılı biçimlerde açıklanmıştır (Menteşe, 2019, s. 21). Ayrıca, Hadoop ve PostgreSQL’in yapısal karşılaştırması veri tabanı modeli, lisans, geliştirici, uygulama dili, sunucu işletim sistemi, SQL dili, veri şeması, erişim yöntemleri, desteklenen programlama dilleri ve gerçek zamanlı analiz gibi metrikler üzerinden yapılmıştır (Menteşe, 2019, s. 38).

Sütun bazlı veri tabanlarının analitik sorgularda sunduğu performans avantajları incelenmiştir. Farklı büyüklüklerde (1 milyon, 10 milyon 25 milyon ve 50 milyon) veri setleri üzerinde gerçekleştirilen karşılaştırmalı ölçümlerde, sorgu türlerine göre tepki sürelerini raporlanmıştır. MySQL ve PostgreSQL gibi ilişkisellerin yanı sıra Cassandra, MonetDB ve MongoDB gibi sütun bazlı belgeleme tabanlı NoSQL sistemleri ele alınmıştır. Yöntem olarak, her bir senaryoda ortalama yanıt süreleri alınmış ve bellek-içi teknolojinin etkisi vurgulanmıştır (Duran-Cazar vd., 2019, s. 2).

Dağıtık veri tabanlarında ver parçalama (fragmentation) yöntemleri, performans ve iletişim maliyetini düşürmek amacıyla klasik bölme stratejilerinin

ötesine geçmiştir. Hiyerarşik aglomeratif kümeleme ve k-means gibi kümeleme tabanlı yaklaşımların yanı sıra değiştirilmiş CRUD matrisine dayalı yöntemler incelenmiştir. Ayrıca DYFRAM modeliyle dinamik e isteğe bağlı fragment yönetiminin iletişim maliyetlerini minimize ettiğini gösterilmiştir (Alluhaibi, 2019, s. 11). Veri tahsisi tarafında ise tavuk sürüsü optimizasyonu (CSO), karınca kolonisi optimizasyonu ve genetik algoritma temelli evrimsel yaklaşımlar, fragmentlerin en uygun sunuculara yerleştirilmesini sağlayarak sorgu gecikmelerini azalttığı belirtilmiştir (Alluhaibi, 2019, s. 15).

İlişkisel veri tabanı ve NoSQL veri tabanı yönetim sistemlerini performans açısından karşılaştırmıştır. MongoDB ve MySQL kullanılarak yapılan ölçümlerde MongoDB'nin büyük veri kümelerinde veri okuma sürelerinde MySQL'e göre büyük bir avantaj sağladığı gözlemlenmiştir. MongoDB, veri ekleme ve okuma işlemlerinde %50 daha yüksek performans sağlarken, SELECT sorgularında daha düşük gecikme süresi sunmuştur. Sonuç olarak, İlişkisel veri tabanları veri bütünlüğü ve karmaşık sorgular açısından avantajlı olsa da MongoDB'nin hız ve ölçeklenebilirlik açısından üstün olduğu bulunmuştur (Jose ve Abraham, 2020, s. 2036).

Veri saklama yöntemleri ve uygulamaları detaylı bir şekilde incelenmiştir. Veri kaydetme, işleme ve yönetim süreçlerinin önemi, farklı veri türleri ve saklama çözümleri bağlamında ele alınmıştır. İlişkisel veri tabanları, yapılandırılmış verilerin saklanmasında yaygın olarak kullanılan yöntemlerden biri olup, veri erişimini ve organizasyonunu kolaylaştıran bir yapıya sahiptir. İlişkisel veri tabanlarının temel avantajları arasında veri bütünlüğü, tutarlılık, güvenlik ve ölçeklenebilirlik bulunmaktadır. NoSQL veri tabanları ise özellikle büyük ölçekli ve düzensiz veri yapıları için çözümler sunmaktadır (Akadal, 2021, s. 1).

MongoDB gibi ilişkisel olmayan veri tabanlarının performansını incelemiştir. Çalışma, büyük veri uygulamaları için esneklik ve performans avantajları sağlayan MongoDB'nin belge tabanlı yapısını vurgulamıştır. Ayrıca, MSSQL ile karşılaştırmalarda gösterildiği gibi, NoSQL veri tabanlarının esnek veri yönetimindeki avantajlarına da dikkat çekmiştir (Yıldız vd., 2021, s. 152).

ClickHouse'un büyük veri analizinde üstün performans sağlayan sütunlu bir OLAP veri tabanı olduğunu belirtmiştir. MergeTree motoru ölçeklenebilirlik ve dayanıklılık sağlamaktadır. SIMD paralelleştirme teknikleriyle işlemci verimliliğini artırmaktadır. MySQL, Kafka, PostgreSQL gibi sistemlerle entegrasyon desteği

sağlayarak esnek kullanım sağlamaktadır. ClickHouse'un diğer veri tabanlarına kıyasla tıklama akışı ve trafik analizinde yüksek performans gösterdiği belirtilmiştir (Schulze vd., 2024, s. 3731–3732).

3. TEST ORTAMI ve SANAL SUNUCUNUN BELİRLENMESİ

Veri tabanı mimarilerinin genel performans değerlendirmesi için sistematik bir test metodolojisi hazırlanmıştır. Bu değerlendirme için önbellek tabanlı veri tabanı, belge tabanlı veri tabanı, sütun tabanlı veri tabanı ve ilişkisel veri tabanı mimarileri değerlendirilmiştir. Önbellek tabanlı veri tabanı için Redis, belge tabanlı veri tabanı için MongoDB, sütun tabanlı veri tabanı için ClickHouse, ve ilişkisel veri tabanı için PostgreSQL veri tabanı kullanılmıştır.

Performansa dayalı karşılaştırma için, Amazon'un geliştirmiş olduğu AWS' de dağıtılan Linux tabanlı EC2 sanal makineleri test ortamı olarak kullanılmıştır. AWS' nin Paris bölgesindeki her veri tabanı için bir t3.2xlarge sanal makinesi tahsis edilmiştir. Kaynakların oluşturulmasında tutarlılığı sağlamak için standart donanım yapılandırmaları uygulanmıştır. Tüm testler bu koşullara göre gerçekleştirilmiştir. Farklı veri hacimlerindeki performansı incelemek için 1 milyon, 10 milyon 50 milyon, 100 milyon içeren veri kümeleri kullanılmıştır (Amazon Web Services, Inc., 2024).

Veri kümeleri oluşturulurken, örnek olarak Kaggle'daki NYC Yellow Taxi Trip Data veri kümesi alınmıştır. Örnek veri kümesi, Python dilinde geliştirilen betikle sıfırdan oluşturulmuştur. Her veri tabanı için sırasıyla 1 milyon, 10 milyon, 50 milyon ve 100 milyon kayıttan oluşan tablolar oluşturulmuştur. Veri oluşturma sürecinde, her sütunun verileri rastgele oluşturulmuştur. Oluşturulan tablolaların sütun adları travel_id, passenger_count, trip_distance, total_amount ve pickup_datetime olarak belirlenmiştir. Travel_id kolonu için integer veri tipi seçilmiştir. Passenger_count kolonu için integer veri tipi seçilmiştir. Trip_distance kolonu için decimal veri tipi seçilmiştir. Total_amount kolonu için decimal veri tipi seçilmiştir. Pickup_datetime kolonu için datetime veri tipi seçilmiştir. Bu kolonlar için rastgele sayılar, her bir veri kümesi için oluşturulmuştur (Elemento, 2023).

Her veri tabanı ve veri kümesi için ALL, COUNT, SUM, MAX, MIN, ROUND, GROUP BY, DISTINCT, HAVING AVG, TOMONTH, CASE, BETWEEN, UNIQEXACT ve IN sorguları Python dilinde geliştirilen dinamik bir betik aracılığıyla çalıştırılmıştır. Her sorgu üç kez çalıştırılmış ve ortalama yürütme süreleri aynı betik tarafından ölçülmüştür. Elde edilen ölçüm sonuçları bir metin

dosyasına kaydedilmiş ve veriler tablolar ve grafiklerle karşılaştırmalı olarak analiz edilmiştir.

Sorguları izlemek için oluşturulan dinamik betik, her veri tabanı için ayrı ayrı sorgular oluşturmak üzere tasarlanmıştır. Sorgularda bulunan {table} ifadesi ile dize işleme teknikleri kullanılarak her veri kümesi için dinamik olarak yürütülmek üzere geliştirilmiştir. Tüm sorgular, her veri tabanı için tablolarda listelenmiştir.

3.1. AWS

Amazon Web Service (AWS), Amazon tarafından sunulan, altyapı hizmetleri (IaaS), platform hizmetleri (PaaS) ve paketlenmiş yazılım çözümleri (SaaS) gibi çeşitli bileşenleri kapsayan kapsamlı ve sürekli gelişen bir bulut bilişim platformudur. AWS, Amazon.com'un çevrimiçi perakende operasyonlarını desteklemek amacıyla geliştirilen dahili altyapı temel alınarak 2006 yılında hizmete sunulmuştur. AWS, kullanıcıların ihtiyaçlarına göre ölçeklenebilen ve "kullandıkça öde" modeliyle hizmet sunan bulut bilişim yaklaşımlarını tanıtan öncü platformlardan biri olarak ön plana çıkmaktadır. AWS, dünya genelinde 190'dan fazla ülkede faaliyet gösteren birçok işletmeye yüksek güvenilirlik, ölçeklenebilirlik, ve düşük maliyet avantajları sunan bir altyapı platformu sağlamaktadır (Li vd., 2010, s. 1).

Milyonlarca müşteri en hızlı büyüyen girişimlerden, büyük ölçekli kurumsal yapılara ve önde gelen kamu kurumlarına kadar operasyonel maliyetleri düşürmek, çeviklik kazanmak ve inovasyon süreçlerini hızlandırmak amacıyla AWS'yi tercih etmektedir. Özellikle kurumsal uygulamalarda ve veri işleme sistemlerinde projelerin sağladığı esneklik nedeniyle AWS, CPU, ağ depolama ve işletim sistemi bileşenlerini içeren bilgi işlem ortamları sunarak yoğun biçimde kullanılmaktadır (Jackson vd., 2010, s. 159).

AWS'nin sağladığı hizmetlerden bazıları; Elastik Hesaplama Bulutu (EC2), Elastik Blok Depolama (EBS) ve Basit Depolama Hizmeti (S3) şeklindedir. EC2 kullanıcıya esnek bilgi işlem gücü sağlamakta, EBS, çalışan sanal sunuculara bağlanabilen kalıcı blok depolama alanı sunmaktadır. S3 internet üzerinden yüksek erişilebilirlik ve güvenilirlik sunan bir nesne depolama hizmeti olarak öne çıkmaktadır (Schad vd., 2010, s. 460–471).

Veri seti üzerinde yapılan performans çalışması için AWS'nin sunmuş olduğu EC2 hizmeti üzerinde gerçekleştirilmiştir. EC2 hizmeti üzerinden sanal makine oluşturulmuş olup çalışma bu makine üzerinden gerçekleştirilmiştir.

Amazon Elastic Compute Cloud (EC2), AWS bünyesinde, ölçeklenebilir bilgi işlem kapasitesi sunan bir web hizmetidir. Kullanıcıların sanal sunucular oluşturmasına ve yönetmesine olanak tanıyan EC2, özellikle yüksek performanslı bilimsel hesaplamalar için esnek ve maliyet etkin çözümler sağmaktadır.

EC2 üzerinde t3.2xlarge makinesi seçilmiş ve tüm veri işlemleri bu makine üzerinden gerçekleştirilmiştir. T3.2xlarge makinesi 8 çekirdekli, 32 gigabit (GİB) özelliklerini barındırmaktadır. Makineye aynı zamanda 300 GB SSD harici bellek tahsis edilmiştir. Veri boyutu sebebiyle bu makine tercih edilmiştir. Makine üzerinde işletim sistemi olarak da Linux tabanlı işletim sistemi seçilmiştir.

3.2. Linux

Linux, 1991 yılında Helsinki Üniversitesi'nde öğrenim görmekte olan Linus Torvalds tarafından geliştirilmeye başlanan, Unix benzeri mimariye sahip, açık kaynak kodlu ve özgür bir işletim sistemi çekirdeğidir. İlk olarak Torvalds'ın kendi lisans koşulları altında dağıtılan Linux, kısa bir süre içerisinde GNU Genel Kamu Lisansı (GPL) ile lisanslanarak özgür yazılım topluluğunun katkılarına açık hale getirilmiş ve böylece dünya genelindeki geliştiriciler tarafından ortaklaşa geliştirilen önemli bir yazılım projesi haline gelmiştir (Torvalds, 1999, s. 38).

Linux çekirdeği, monolitik ve modüler mimari yaklaşımlarının bir bileşimi olarak tasarlanmıştır. Monolitik mimari, çekirdeğin tüm temel işlevlerinin aynı adres alanı içerisinde çalışmasına olanak tanırken; modüler yapı, çekirdeğin belirli bileşenlerinin gereksinim doğrultusunda dinamik olarak yüklenip kaldırılabilmesini mümkün kılmaktadır. Bu mimari esneklik, kullanıcıların sistem yapılandırmalarını kendi ihtiyaçlarına göre özelleştirebilmelerine imkân sağlamaktadır. Linux işletim sisteminin evrimi süresince, çekirdek mimarisi ve modülerlik üzerine gerçekleştirilen çeşitli çalışmalar, sistemin farklı donanım mimarileriyle uyumlu çalışabilmesini kolaylaştırmış ve Linux'un çok çeşitli platformlarda yaygın olarak benimsenmesine katkı sunmuştur (Chen vd., 2021, s. 277–287).

Günümüzde Linux, masaüstü bilgisayarlardan sunuculara, süper bilgisayarlardan gömülü sistemlere kadar geniş bir yelpazede kullanılmaktadır. Özellikle süper bilgisayarlar alanında, Linux tabanlı sistemler, yüksek performanslı hesaplamalar için tercih edilen platformlar haline gelmiştir. Ayrıca mobil cihazlar ve gömülü sistemlerde de yaygın olarak kullanılmakta, esnekliği ve güvenilirliği sayesinde farklı endüstrilerde önemli bir rol oynamaktadır (Yan vd., 2022, s. 250).

Linux işletim sistemi üzerinde komut sistemi kullanılmaktadır. Dosyalama yöntemleri, ağ yapılandırması gibi çeşitli görevlerin etkin bir şekilde gerçekleştirilmesine olanak tanımaktadır. Veri tabanı kurulumları esnasında bu komutlar ile işlemler sağlanmıştır. ClickHouse, Redis, MongoDb ve PostgreSQL veri tabanları için özel Linux komutları kullanılmıştır.

4. SÜTUN TABANLI ve ÖNBELLEK TABANLI VERİ TABANLARI

Sütun tabanlı OLAP veri tabanları ile önbellek tabanlı anahtar-değer çözümlerinin mimari ve performans özellikler ele alınmıştır. İlk olarak, veri sütunlar halinde depolandığı ve analitik sorguların büyük hacimli veri kümelerinde en yüksek verimlilikle işlenmesini sağlayan ClickHouse'un temel tasarım şablonları, veri türü desteği ve ölçeklenebilirlik avantajları incelenmiştir. Bellek içi çalışma modeliyle düşük gecikmeli I/O işlemleri için tercih edilen Redis'in esnek veri yapıları, kalıcılık seçenekleri ve gerçek zamanlı uygulamalardaki kullanımı değerlendirilmiştir.

4.1. Clickhouse

ClickHouse veri tabanı, sütun tabanlı Online Analytical Processing (OLAP) veri tabanı yönetim sistemi olarak, özellikle yüksek hacimli analitik sorguların verimli ve hızlı bir şekilde işlenmesi amacıyla tasarlanmıştır. İlişkisel veri tabanlarının aksine, sütun tabanlı mimarisi sayesinde sadece ihtiyaç duyulan sütunlara erişim sağlanarak işlem süresi önemli ölçüde azaltılmakta ve bellek kullanımı optimize edilmektedir. Bu özellik, veri analizinde yüksek performans gerektiren uygulamalar için ClickHouse veri tabanını uygun bir tercih haline getirmektedir (ClickHouse, 2024).

ClickHouse, tam sayı (integer), kayan noktalı sayı (floating-point), metin (string), tarih ve dizi (array) gibi farklı veri türlerini desteklemektedir. Bu çeşitlilik, veri modelleme süreçlerinde esneklik sağlamaktadır. Sistemin yüksek veri yazma hızı, büyük veri kümeleriyle çalışan uygulamalar için zaman kazanımı sağlayan önemli bir avantaj haline gelmiştir (ClickHouse, 2025).

ClickHouse'un diğer özelliği ise kullanıcıya veri işaretleme ve kelime boyutu üzerinde kontrol imkanı sunmasıdır. Bu sayede, veri depolama yapısı kullanıcı ihtiyaçlarına göre özelleştirilebilmektedir. Bu özellik sayesinde performansın daha artırılmasına olanak tanınmaktadır. Saniyeler içerisinde petabaytlarca veriyi okuma kapasitesine sahip olması, ClickHouse' un büyük ölçekli veri analizlerinde ne denli güçlü bir çözüm olduğunu ortaya koymaktadır (Vasile vd., 2020, Makale 1).

AWS sisteminde EC2 makinesi üzerinde kurulan ClickHouse veri tabanı üzerinde 1 milyon, 10 milyon, 50 milyon ve 100 milyon kayıtlı veri setleri

oluşturulmuştur. Kaggle üzerinde bulunan NYC Yellow Taxi Trip Data örneklenerek veri setleri, Python dilinde geliştirilen otomatik betikler sayesinde oluşturulmuştur.

Clickhouse için kurulan tüm tablolarda son 10 kayıt için listeleme sağlanıp ekran görüntüleri ile kayıtlar gösterilmiştir. Son 10 kayıt gösterilebilmesi için sorgular ORDER BY DESC komutu ve LIMIT 10 komutu ile son 10 kayıt gösterilmiştir.

```
SELECT *
FROM nyc_taxi_data_1m
ORDER BY travel_id DESC
LIMIT 10

Query id: f21c1b81-f688-489d-b50c-ccfff5177828

    ┌─travel_id─┬─passenger_count─┬─trip_distance─┬─total_amount─┬─────pickup_datetime─┐
 1. │   1000000 │               3 │         16.67 │        69.75 │ 2025-06-09 20:50:02 │
 2. │    999999 │               5 │          3.68 │        79.82 │ 2025-06-09 20:51:02 │
 3. │    999998 │               6 │         10.34 │        72.25 │ 2025-06-09 20:52:02 │
 4. │    999997 │               2 │          2.42 │        30.41 │ 2025-06-09 20:53:02 │
 5. │    999996 │               3 │           4.9 │        58.46 │ 2025-06-09 20:54:02 │
 6. │    999995 │               4 │          6.34 │        34.71 │ 2025-06-09 20:55:02 │
 7. │    999994 │               3 │          7.31 │         5.28 │ 2025-06-09 20:56:02 │
 8. │    999993 │               1 │          4.49 │        48.12 │ 2025-06-09 20:57:02 │
 9. │    999992 │               2 │         11.31 │         64.4 │ 2025-06-09 20:58:02 │
10. │    999991 │               1 │         10.02 │        54.23 │ 2025-06-09 20:59:02 │
    └───────────┴─────────────────┴───────────────┴──────────────┴─────────────────────┘
```

Şekil 1. ClickHouse 1 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu sorgu kapsamında, nyc_taxi_data_1m veri seti üzerinde gerçekleştirilen sorgu ile en güncel 10 taksi yolculuğuna ait veriler incelenmiştir. Sorgu, yolculuk kimliği (travel_id) alanına göre azalan sıralama yapılarak sınırlı sayıda kaydın seçilmesini sağlamaktadır. Elde edilen sonuçlar; yolcu sayısı, yolcu mesafesi, toplam ücret ve yolculuk başlangıç zamanı gibi temel değişkenleri içermektedir.

```
SELECT *
FROM nyc_taxi_data_10m
ORDER BY travel_id DESC
LIMIT 10

Query id: 71ddf921-6e5c-4f47-8589-3529be3bf3f6

    ┌─travel_id─┬─passenger_count─┬─trip_distance─┬─total_amount─┬─────pickup_datetime─┐
 1. │  10000000 │               2 │          1.29 │        51.48 │ 2025-06-09 20:52:46 │
 2. │   9999999 │               3 │         14.94 │         16.7 │ 2025-06-09 20:53:46 │
 3. │   9999998 │               4 │          7.46 │        69.68 │ 2025-06-09 20:54:46 │
 4. │   9999997 │               2 │          1.52 │        40.59 │ 2025-06-09 20:55:46 │
 5. │   9999996 │               4 │          7.86 │        14.71 │ 2025-06-09 20:56:46 │
 6. │   9999995 │               5 │         22.54 │        75.38 │ 2025-06-09 20:57:46 │
 7. │   9999994 │               4 │         20.76 │        62.56 │ 2025-06-09 20:58:46 │
 8. │   9999993 │               1 │          0.66 │         71.4 │ 2025-06-09 20:59:46 │
 9. │   9999992 │               4 │          2.73 │        77.97 │ 2025-06-09 21:00:46 │
10. │   9999991 │               3 │          6.12 │         22.5 │ 2025-06-09 21:01:46 │
    └───────────┴─────────────────┴───────────────┴──────────────┴─────────────────────┘
```

Şekil 2. ClickHouse 10 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu sorgu kapsamında, nyc_taxi_data_10m veri seti üzerinde travel_id değerine göre azalan sıralama ile en güncel 10 yolculuk verisi sorgulanmıştır.

```
SELECT *
FROM nyc_taxi_data_50m
ORDER BY travel_id DESC
LIMIT 10

Query id: 7c6b72c5-4df0-4920-bc60-b7c3bbb8bf9c
```

	travel_id	passenger_count	trip_distance	total_amount	pickup_datetime
1.	50000000	4	10.36	51.97	2025-06-09 21:01:00
2.	49999999	6	8.17	57.78	2025-06-09 21:02:00
3.	49999998	2	7.21	66.54	2025-06-09 21:03:00
4.	49999997	4	8.18	57.18	2025-06-09 21:04:00
5.	49999996	4	21.88	15.19	2025-06-09 21:05:00
6.	49999995	3	17.87	71.12	2025-06-09 21:06:00
7.	49999994	4	1.88	14.89	2025-06-09 21:07:00
8.	49999993	1	17.17	45.21	2025-06-09 21:08:00
9.	49999992	3	21.7	22.77	2025-06-09 21:09:00
10.	49999991	2	8.01	41.24	2025-06-09 21:10:00

Şekil 3. ClickHouse 50 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu sorgu kapsamında, nyc_taxi_data_50m veri seti üzerinde travel_id değerine göre azalan sıralama ile en güncel 10 yolculuk verisi sorgulanmıştır.

```
SELECT *
FROM nyc_taxi_data_100m
ORDER BY travel_id DESC
LIMIT 10

Query id: 26455a5c-12e5-46ea-8531-919cc429a131
```

	travel_id	passenger_count	trip_distance	total_amount	pickup_datetime
1.	100000000	1	8.25	43.66	2025-06-09 20:58:28
2.	99999999	6	18.43	36.22	2025-06-09 20:59:28
3.	99999998	3	22.19	69.18	2025-06-09 21:00:28
4.	99999997	3	10.64	78.21	2025-06-09 21:01:28
5.	99999996	6	12.63	39.93	2025-06-09 21:02:28
6.	99999995	1	5.63	68.06	2025-06-09 21:03:28
7.	99999994	2	24.49	74.83	2025-06-09 21:04:28
8.	99999993	5	11.9	63.03	2025-06-09 21:05:28
9.	99999992	4	5.73	38.65	2025-06-09 21:06:28
10.	99999991	4	22.59	65.76	2025-06-09 21:07:28

Şekil 4. ClickHouse 100 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu sorgu kapsamında, nyc_taxi_data_100m veri seti üzerinde travel_id değerine göre azalan sıralama ile en güncel 10 yolculuk verisi sorgulanmıştır.

4.2. Redis

Redis, ilişkisel olmayan ve önbellek tabanlı bir veri tabanı olarak tasarlanmıştır. Temel olarak verileri RAM üzerinde tutarak çok hızlı veri erişimi ve

işleme performansı sunmaktadır. Verilerin kalıcılığını sağlamak amacıyla aynı zamanda dosya sisteminde veri yazabilmektedir. Bu sayede hem hızlı erişim hem de veri kalıcılığı gibi iki önemli özellik bir arada sunulmaktadır (Redis, 2024).

Açılımı Remote Dictionary Server olan Redis,, BSD lisansı ile dağıtılan ve anahtar-değer yapısını temel alan bir NoSQL veri tabanıdır. Geleneksel ilişkisel veri tabanlarında bulunan katı şema yapılarının aksine, Redis esnek veri modellemesine olanak tanıyarak farklı uygulama senaryolarında kullanılabilmektedir. Performans gerektiren uygulamalarda ve gerçek zamanlı veri işleme süreçlerinde Redis, gecikme süresinin düşük olması ve hızlı cevap özellikleriyle ön plana çıkmaktadır (Redis, 2025).

Redis, temel veri türlerini değil, aynı zamanda gelişmiş veri yapıları için de yerleşik destek sağlamaktadır. Bu veri yapıları arasında listeler, ilişkisel diziler, kümeler ve sıralı kümeler bulunmaktadır. Önbellekleme, mesajlaşma sistemleri, oturum yönetimi, puan tabloları ve gerçek zamanlı analiz gibi birçok alanda kullanılmaktadır (Xu vd., 2014, s. 2642–2650).

Kurulan makine üzerinde Redis veri tabanı için 1 milyon, 10 milyon, 50 milyon ve 100 milyonluk veri setleri oluşturulmuştur. NYC Yellow Taxi Trip Data veri seti örnek alınarak otomasyon betikleriyle veri setleri oluşturulmuştur.

Redis için kurulan tüm tablolarda son 10 kayıt için listeleme sağlanıp ekran görüntüleri ile kayıtlar gösterilmiştir. SORT ve HMGET komutları yardımıyla her yolculuğa ait passenger_count, trip_distance, total_amount ve pickup_datetime bilgileri çekilmiş ve sütunlar halinde düzenlenmiştir.

```
> {
>   printf "travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime\n"
>   redis-cli --raw SORT nyc_taxi_data_1m:keys DESC LIMIT 0 10 |
>   while read id; do
>     vals=$(redis-cli --raw HMGET nyc_taxi_data_1m:$id \
>             passenger_count trip_distance total_amount pickup_datetime \
>           | paste -sd "\t" -)
>     printf "%s\t%s\n" "$id" "$vals"
>   done
> } ' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
1000000    2                13.44          57.78         2025-06-09T23:03:58Z
999999     6                1.82           44.37         2025-06-09T23:04:58Z
999998     3                17.22          66.30         2025-06-09T23:05:57Z
999997     4                1.39           16.73         2025-06-09T23:06:57Z
999996     2                24.53          76.56         2025-06-09T23:07:57Z
999995     3                6.93           22.27         2025-06-09T23:08:57Z
999994     3                9.70           58.68         2025-06-09T23:09:56Z
999993     3                16.90          18.71         2025-06-09T23:10:56Z
999992     5                8.29           74.64         2025-06-09T23:11:56Z
999991     2                6.42           65.77         2025-06-09T23:12:56Z
```

Şekil 5. Redis 1 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde, Redis veri tabanı kullanılarak nyc_taxi_data_1m anahtar kümesinden en yüksek 10 travel_id değerine ait veriler komut satırı üzerinden sorgulanmıştır.

```
> {
>   printf "travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime\n"
>   redis-cli --raw SORT nyc_taxi_data_10m:keys DESC LIMIT 0 10 |
>   while read id; do
>     vals=$(redis-cli --raw HMGET nyc_taxi_data_10m:$id \
>             passenger_count trip_distance total_amount pickup_datetime \
>           | paste -sd "\t" -)
>     printf "%s\t%s\n" "$id" "$vals"
>   done
> }' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
10000000   2                18.25          47.42         2025-06-09T23:04:00Z
9999999    6                14.37          62.08         2025-06-09T23:05:00Z
9999998    3                10.60          41.63         2025-06-09T23:05:59Z
9999997    2                19.29          26.49         2025-06-09T23:06:59Z
9999996    1                0.66           48.19         2025-06-09T23:07:59Z
9999995    5                13.37          5.38          2025-06-09T23:08:59Z
9999994    4                22.36          4.96          2025-06-09T23:09:59Z
9999993    4                9.92           8.05          2025-06-09T23:10:59Z
9999992    6                22.73          39.46         2025-06-09T23:11:58Z
9999991    3                8.85           49.50         2025-06-09T23:12:58Z
```

Şekil 6. Redis 10 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde, Redis veri tabanı kullanılarak nyc_taxi_data_10m anahtar kümesinden en yüksek 10 travel_id değerine ait veriler komut satırı üzerinden sorgulanmıştır.

```
> {
>   printf "travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime\n"
>   redis-cli --raw SORT nyc_taxi_data_50m:keys DESC LIMIT 0 10 |
>   while read id; do
>     vals=$(redis-cli --raw HMGET nyc_taxi_data_50m:$id \
>             passenger_count trip_distance total_amount pickup_datetime \
>           | paste -sd "\t" -)
>     printf "%s\t%s\n" "$id" "$vals"
>   done
> }' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
50000000   4                15.98          36.32         2025-06-09T23:04:02Z
49999999   5                6.64           31.70         2025-06-09T23:05:02Z
49999998   1                6.24           67.87         2025-06-09T23:06:02Z
49999997   2                9.56           20.05         2025-06-09T23:07:01Z
49999996   1                0.59           60.91         2025-06-09T23:08:01Z
49999995   5                7.20           78.51         2025-06-09T23:09:01Z
49999994   1                3.83           38.02         2025-06-09T23:10:01Z
49999993   1                1.26           8.24          2025-06-09T23:11:01Z
49999992   2                19.39          24.15         2025-06-09T23:12:00Z
49999991   2                15.91          54.44         2025-06-09T23:13:00Z
```

Şekil 7. Redis 50 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde, Redis veri tabanı kullanılarak nyc_taxi_data_10m anahtar kümesinden en yüksek 10 travel_id değerine ait veriler komut satırı üzerinden sorgulanmıştır.

```
> {
>   printf "travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime\n"
>   redis-cli --raw SORT nyc_taxi_data_100m:keys DESC LIMIT 0 10 |
>   while read id; do
>     vals=$(redis-cli --raw HMGET nyc_taxi_data_100m:$id \
>             passenger_count trip_distance total_amount pickup_datetime \
>           | paste -sd "\t" -)
>     printf "%s\t%s\n" "$id" "$vals"
>   done
> }' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
100000000  5                19.88          66.28         2025-06-09T23:04:04Z
99999999   2                21.50          40.02         2025-06-09T23:05:04Z
99999998   6                17.59          12.67         2025-06-09T23:06:04Z
99999997   5                17.58          50.04         2025-06-09T23:07:04Z
99999996   1                17.96          74.55         2025-06-09T23:08:03Z
99999995   4                8.26           37.88         2025-06-09T23:09:03Z
99999994   2                14.16          3.44          2025-06-09T23:10:03Z
99999993   6                11.75          78.41         2025-06-09T23:11:03Z
99999992   2                20.81          77.65         2025-06-09T23:12:03Z
99999991   4                19.11          79.08         2025-06-09T23:13:02Z
```

Şekil 8. Redis 100 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde, Redis veri tabanı kullanılarak nyc_taxi_data_10m anahtar kümesinden en yüksek 10 travel_id değerine ait veriler komut satırı üzerinden sorgulanmıştır.

5. BELGE TABANLI ve İLİŞKİSEL VERİ TABANLARI

Belge tabanlı NoSQL veri tabanları ile geleneksel ilişkisel veri tabanı sistemlerinin mimari yaklaşımları, veri modelleme esneklikleri ve ölçeklenebilirlik özellikleri ele alınmıştır. Öncelikle, JSON benzeri esnek belge yapısıyla dinamik veri modelleri sunan MongoDB'nin yatay parçalama ve replikasyon mekanizmaları incelenmiş; ardından güçlü ACID garantileri, zengin sorgu yetenekleri ve nesne-ilişkisel uzantılarıyla öne çıkan PostgreSQL'in modern veri gereksinimlerine nasıl cevap verdiği değerlendirilmiştir.

5.1. MongoDB

MongoDB veri tabanı belge tabanlı veri modeline sahip, ilişkisel olmayan (NoSQL) veri tabanı sistemleri arasında tercih edilen veri tabanı yönetim sistemlerinden biridir. Standart ilişkisel veri tabanlarında kullanılan tablo ve satır yapısı yerine, JSON benzeri BSON (Binary JSON) formatında belgeler kullanarak veri saklamaktadır. Esnek veri yapısı, yapılandırılmamış ya da yarı yapılandırılmış verilerin yönetimini kolaylaştırmakta ve uygulama geliştiricilere veri modellerini hızlı ve dinamik biçimde tanımlama imkanı sunmaktadır (Kaur ve Sing, 2022, s. 24–30).

MongoDB, yüksek performans ve yatay ölçeklenebilirlik sağlamak amacıyla geliştirilmiştir. Veri tabanı yatayda parçalama yöntemiyle veriyi farklı sunuculara dağıtarak çok büyük veri kümelerinin dahi etkili bir biçimde yönetilmesine olanak tanımaktadır. Verinin birden fazla sunucuda kopyalanmasını sağlayan replikasyon mekanizması sayesinde sistem, arıza durumlarında otomatik olarak yedek sunuculara geçiş yapabilmektedir. Bu özelliği sayesinde kesintisiz hizmet sunmaktadır (MongoDB, 2022).

MongoDB'nin sağladığı indeksleme sistemleri, sorgu performansını artırmakta ve analitik işlemleri daha verimli hale getirmektedir. Ayrıca değişen veri ihtiyaçlarına karşı adaptasyon özelliği, mikro servis mimarilerinde veya çevik projelerde avantaj sunmaktadır. Günümüzde büyük ölçekli şirket ve kuruluş, MongoDB'yi yalnızca veri depolama aracı olarak değil aynı zamanda ölçeklenebilir ve dağıtık uygulamalar için kullanmaktadır (Rani vd., 2023, s. 3184–3189).

MongoDB üzerinde, Kaggle üzerinde bulunan NYC Yellow Taxi Trip Data veri seti üzerinden esinlenerek 1 milyon, 10 milyon, 50 milyon ve 100 milyon kayıtlı veri setleri Python dili üzerinde geliştirilen otomasyon betiği ile oluşturulmuştur.

MongoDB üzerinde koleksiyonlar oluşturularak azalan şekilde sıralanmış ilk 10 kayıt sorgulanmıştır. Sonuçlar JSON formatında olduğundan kaynaklı çıkan sonuçlar düzgün gözükmesi amaçlı tablo şeklinde listelenmiştir.

```
[ec2-user@ip-172-31-35-54 ~]$ mongosh nycdb --quiet --eval '
> print("travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime");
> db.nyc_taxi_data_1m
>   .find({}, {_id:0})
>   .sort({ travel_id:-1 })
>   .limit(10)
>   .forEach(doc => {
>     print(
>       doc.travel_id + "\t" +
>       doc.passenger_count + "\t" +
>       doc.trip_distance.toFixed(2) + "\t" +
>       doc.total_amount.toFixed(2) + "\t" +
>       doc.pickup_datetime.toISOString()
>     );
>   });
> ' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
1000000    1                16.51          37.27         2025-06-10T19:50:52.661Z
999999     3                19.28          75.56         2025-06-10T19:51:52.661Z
999998     4                22.35          53.51         2025-06-10T19:52:52.661Z
999997     1                9.39           73.44         2025-06-10T19:53:52.660Z
999996     6                8.80           31.72         2025-06-10T19:54:52.660Z
999995     4                23.45          64.23         2025-06-10T19:55:52.660Z
999994     3                11.79          69.21         2025-06-10T19:56:52.660Z
999993     2                6.16           30.87         2025-06-10T19:57:52.660Z
999992     3                18.42          70.65         2025-06-10T19:58:52.660Z
999991     6                13.45          70.68         2025-06-10T19:59:52.659Z
```

Şekil 9. MongoDB 1 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde MongoDB de bulunan nyc_taxi_data_1m koleksiyonuna ait son 10 kayıt tablo haline getirilerek listelenmiştir.

```
[ec2-user@ip-172-31-35-54 ~]$ mongosh nycdb --quiet --eval '
> print("travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime");
> db.nyc_taxi_data_50m
>    .find({}, {_id:0})
>    .sort({ travel_id:-1 })
>    .limit(10)
>    .forEach(doc => {
>      print(
>        doc.travel_id + "\t" +
>        doc.passenger_count + "\t" +
>        doc.trip_distance.toFixed(2) + "\t" +
>        doc.total_amount.toFixed(2) + "\t" +
>        doc.pickup_datetime.toISOString()
>      );
>    });
> ' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
50000000   5                10.99          73.66         2025-06-10T19:50:52.671Z
49999999   2                13.92          35.81         2025-06-10T19:51:52.671Z
49999998   2                18.84          27.29         2025-06-10T19:52:52.671Z
49999997   6                18.41          6.69          2025-06-10T19:53:52.671Z
49999996   2                2.62           30.46         2025-06-10T19:54:52.671Z
49999995   2                13.95          65.42         2025-06-10T19:55:52.671Z
49999994   1                15.09          74.85         2025-06-10T19:56:52.671Z
49999993   4                15.60          13.16         2025-06-10T19:57:52.671Z
49999992   6                17.45          10.77         2025-06-10T19:58:52.671Z
49999991   5                1.92           9.76          2025-06-10T19:59:52.671Z
```

Şekil 10. MongoDB 10 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde MongoDB de bulunan nyc_taxi_data_10m koleksiyonuna ait son 10 kayıt tablo haline getirilerek listelenmiştir.

```
[ec2-user@ip-172-31-35-54 ~]$ mongosh nycdb --quiet --eval '
> print("travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime");
> db.nyc_taxi_data_50m
>    .find({}, {_id:0})
>    .sort({ travel_id:-1 })
>    .limit(10)
>    .forEach(doc => {
>      print(
>        doc.travel_id + "\t" +
>        doc.passenger_count + "\t" +
>        doc.trip_distance.toFixed(2) + "\t" +
>        doc.total_amount.toFixed(2) + "\t" +
>        doc.pickup_datetime.toISOString()
>      );
>    });
> ' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
50000000   5                10.99          73.66         2025-06-10T19:50:52.671Z
49999999   2                13.92          35.81         2025-06-10T19:51:52.671Z
49999998   2                18.84          27.29         2025-06-10T19:52:52.671Z
49999997   6                18.41          6.69          2025-06-10T19:53:52.671Z
49999996   2                2.62           30.46         2025-06-10T19:54:52.671Z
49999995   2                13.95          65.42         2025-06-10T19:55:52.671Z
49999994   1                15.09          74.85         2025-06-10T19:56:52.671Z
49999993   4                15.60          13.16         2025-06-10T19:57:52.671Z
49999992   6                17.45          10.77         2025-06-10T19:58:52.671Z
49999991   5                1.92           9.76          2025-06-10T19:59:52.671Z
```

Şekil 11. MongoDB 50 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde MongoDB de bulunan nyc_taxi_data_50m koleksiyonuna ait son 10 kayıt tablo haline getirilerek listelenmiştir.

```
[ec2-user@ip-172-31-35-54 ~]$ mongosh nycdb --quiet --eval '
> print("travel_id\tpassenger_count\ttrip_distance\ttotal_amount\tpickup_datetime");
> db.nyc_taxi_data_100m
>   .find({}, {_id:0})
>   .sort({ travel_id:-1 })
>   .limit(10)
>   .forEach(doc => {
>     print(
>       doc.travel_id + "\t" +
>       doc.passenger_count + "\t" +
>       doc.trip_distance.toFixed(2) + "\t" +
>       doc.total_amount.toFixed(2) + "\t" +
>       doc.pickup_datetime.toISOString()
>     );
>   });
> ' | column -t -s $'\t'
travel_id  passenger_count  trip_distance  total_amount  pickup_datetime
100000000  1                21.59          33.99         2025-06-10T19:50:52.677Z
99999999   3                13.84          16.83         2025-06-10T19:51:52.677Z
99999998   6                21.52          68.57         2025-06-10T19:52:52.677Z
99999997   3                13.38          13.38         2025-06-10T19:53:52.677Z
99999996   4                8.41           54.67         2025-06-10T19:54:52.677Z
99999995   5                21.10          53.90         2025-06-10T19:55:52.677Z
99999994   5                2.92           42.89         2025-06-10T19:56:52.677Z
99999993   5                8.57           50.66         2025-06-10T19:57:52.677Z
99999992   2                19.31          3.19          2025-06-10T19:58:52.677Z
99999991   6                15.60          66.89         2025-06-10T19:59:52.677Z
```

Şekil 12. MongoDB 100 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde MongoDB de bulunan nyc_taxi_data_100m koleksiyonuna ait son 10 kayıt tablo haline getirilerek listelenmiştir.

5.2. PostgreSQL

PostgreSQL, açık kaynak kodlu ve gelişmiş özelliklere sahip ilişkisel veri tabanı yönetim sistemi olarak geliştirilmiştir. Temelleri 1986 yılında Kaliforniya Üniversitesi, Berkeley'deki POSTGRES projesine dayanan PostgreSQL, yıllar içinde geliştiriciler tarafından sürekli geliştirilmiştir. İlişkisel veri tabanı sistemlerinin sunduğu ACID özelliklerini destekleyen PostgreSQL, aynı zamanda nesne-ilişkisel veri tabanı özellikleriyle karmaşık veri türleri ve ilişkileri üzerinde çalışmaya olanak tanımaktadır (Turkcell Teknoloji, 2018).

PostgreSQL veri tabanının avantajlarından biri, hem ilişkisel hem de belirli ölçüde ilişkisel olmayan sorguları desteklemesidir. JSON, XML ve hstore gibi veri türlerini yerel olarak desteklemesi, yapılandırılmamış veri ile çalışmayı mümkün kılmaktadır. Bu özelliği sayesinde PostgreSQL veri tabanı modern uygulama ihtiyaçlarına cevap verebilmektedir (Sematext, 2025).

PostgreSQL veri tabanının gelişimi, yazılım geliştiricilerin ve araştırmacıların katkılarıyla sürdürülmektedir. Bu katkılar, sistemin güvenilirliğini ve sürdürebilirliğini artırmakta, PostgreSQL'i endüstriyel standartlarda bir çözüm haline getirmektedir. Finans, sağlık, kamu ve akademik araştırma gibi veri güvenliğinin kritik olduğu

sektörlerde yaygın olarak tercih edilmektedir. Performans açısından ve verilerin tutarlılığı açısından PostgreSQL her zaman gelişim göstermektedir (Krebs, 2022).

PostgreSQL veri tabanı üzerinde Kaggle'dan alınan NYC Yellow Taxi Trip Data üzerinden esinlenerek, otomatik betiklerle 1 milyon, 10 milyon, 50 milyon ve 100 milyon kayıtlı veri seti oluşturulmuştur.

PostgreSQL veri tabanı için tüm tablolar son 10 kayıt listelemesi SQL cümlecikleriyle gerçekleştirilmiştir. ORDER BY DESC komutu ve LIMIT 10 komutuyla son 10 kayıt listelenmiştir.

```
[ec2-user@ip-172-31-35-54 ~]$ sudo -u postgres psql -d postgres -c "SELECT * FROM nyc_taxi_data_1m ORDER BY travel_id DESC LIMIT 10;"
 travel_id | passenger_count | trip_distance | total_amount |      pickup_datetime
-----------+-----------------+---------------+--------------+----------------------------
   1000000 |               1 |          7.74 |        50.39 | 2025-06-10 21:27:06.812237
    999999 |               3 |         10.24 |        69.93 | 2025-06-10 21:28:06.812237
    999998 |               3 |         16.37 |        76.95 | 2025-06-10 21:29:06.812237
    999997 |               2 |          0.65 |        25.52 | 2025-06-10 21:30:06.812237
    999996 |               2 |         16.33 |        75.70 | 2025-06-10 21:31:06.812237
    999995 |               4 |         22.37 |         9.76 | 2025-06-10 21:32:06.812237
    999994 |               5 |         21.37 |         5.42 | 2025-06-10 21:33:06.812237
    999993 |               2 |         23.41 |         3.93 | 2025-06-10 21:34:06.812237
    999992 |               1 |         21.17 |        51.31 | 2025-06-10 21:35:06.812237
    999991 |               5 |          1.47 |         4.88 | 2025-06-10 21:36:06.812237
(10 rows)
```

Şekil 13. PostgreSQL 1 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde bulunan komut nyc_taxi_data_1m tablosundaki kayıtları travel_id sütununa göre ters sırada sıralamaktadır, ilk değeri 1.000.000 ile başlamakta ve son değeri de 999.991 ile bitmektedir.

```
[ec2-user@ip-172-31-35-54 ~]$ sudo -u postgres psql -d postgres -c "SELECT * FROM nyc_taxi_data_10m ORDER BY travel_id DESC LIMIT 10;"
 travel_id | passenger_count | trip_distance | total_amount |      pickup_datetime
-----------+-----------------+---------------+--------------+----------------------------
  10000000 |               5 |          3.05 |        21.43 | 2025-06-10 21:35:32.977332
   9999999 |               3 |          5.04 |        56.37 | 2025-06-10 21:36:32.977332
   9999998 |               4 |         16.32 |        38.50 | 2025-06-10 21:37:32.977332
   9999997 |               6 |         19.70 |        58.58 | 2025-06-10 21:38:32.977332
   9999996 |               2 |         17.05 |        59.92 | 2025-06-10 21:39:32.977332
   9999995 |               6 |          3.10 |        77.22 | 2025-06-10 21:40:32.977332
   9999994 |               3 |         19.79 |        61.77 | 2025-06-10 21:41:32.977332
   9999993 |               6 |         12.82 |        11.18 | 2025-06-10 21:42:32.977332
   9999992 |               1 |          4.83 |        14.28 | 2025-06-10 21:43:32.977332
   9999991 |               1 |         24.34 |        48.91 | 2025-06-10 21:44:32.977332
```

Şekil 14. PostgreSQL 10 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde bulunan komut nyc_taxi_data_10m tablosundaki kayıtları travel_id sütununa göre ters sırada sıralamaktadır, ilk değeri 10.000.000 ile başlamakta ve son değeri de 9.999.991 ile bitmektedir.

```
[ec2-user@ip-172-31-35-54 ~]$ sudo -u postgres psql -d postgres -c "SELECT * FROM nyc_taxi_data_50m ORDER BY travel_id DESC LIMIT 10;"
 travel_id | passenger_count | trip_distance | total_amount |      pickup_datetime
-----------+-----------------+---------------+--------------+---------------------------
  50000000 |               3 |          2.93 |         5.46 | 2025-06-10 21:35:32.979076
  49999999 |               6 |          9.95 |        71.62 | 2025-06-10 21:36:32.979076
  49999998 |               5 |         18.06 |        76.33 | 2025-06-10 21:37:32.979076
  49999997 |               1 |          9.46 |        63.44 | 2025-06-10 21:38:32.979076
  49999996 |               5 |          7.34 |         5.91 | 2025-06-10 21:39:32.979076
  49999995 |               5 |         12.17 |        65.41 | 2025-06-10 21:40:32.979076
  49999994 |               5 |          2.49 |        31.09 | 2025-06-10 21:41:32.979076
  49999993 |               5 |         10.61 |        45.42 | 2025-06-10 21:42:32.979076
  49999992 |               1 |         21.37 |        10.58 | 2025-06-10 21:43:32.979076
  49999991 |               5 |          5.25 |         6.24 | 2025-06-10 21:44:32.979076
(10 rows)
```

Şekil 15. PostgreSQL 50 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde bulunan komut nyc_taxi_data_50m tablosundaki kayıtları travel_id sütununa göre ters sırada sıralamaktadır, ilk değeri 50.000.000 ile başlamakta ve son değeri de 49.999.991 ile bitmektedir.

```
[ec2-user@ip-172-31-35-54 ~]$ sudo -u postgres psql -d postgres -c "SELECT * FROM nyc_taxi_data_100m ORDER BY travel_id DESC LIMIT 10;"
 travel_id | passenger_count | trip_distance | total_amount |      pickup_datetime
-----------+-----------------+---------------+--------------+---------------------------
 100000000 |               4 |         18.90 |        55.52 | 2025-06-10 21:35:32.98072
  99999999 |               5 |         17.77 |        12.54 | 2025-06-10 21:36:32.98072
  99999998 |               5 |         23.57 |         9.51 | 2025-06-10 21:37:32.98072
  99999997 |               4 |         23.88 |        42.54 | 2025-06-10 21:38:32.98072
  99999996 |               2 |          9.38 |        46.73 | 2025-06-10 21:39:32.98072
  99999995 |               1 |         14.99 |        55.79 | 2025-06-10 21:40:32.98072
  99999994 |               4 |         20.83 |        74.06 | 2025-06-10 21:41:32.98072
  99999993 |               6 |         24.62 |        50.36 | 2025-06-10 21:42:32.98072
  99999992 |               2 |         23.83 |         3.80 | 2025-06-10 21:43:32.98072
  99999991 |               1 |         21.95 |        20.05 | 2025-06-10 21:44:32.98072
(10 rows)
```

Şekil 16. PostgreSQL 100 Milyon Kayıtlı Tablo için Son 10 Kayıt

Kaynak: Yazar tarafından oluşturulmuştur.

Bu görselde bulunan komut nyc_taxi_data_50m tablosundaki kayıtları travel_id sütununa göre ters sırada sıralamaktadır, ilk değeri 100.000.000 ile başlamakta ve son değeri de 99.999.991 ile bitmektedir.

6. OPTİMİZASYON ve TESTLER

Dört farklı veri tabanı mimarisi için uygulanan performans iyileştirme ve test süreçleri anlatılmıştır. Optimizasyon kısmında her sistemin veri modeli, indeksleme ve yapılandırma parametreleri gözden geçirilerek okuma-yazma verimliliğini artırmaya yönelik ayarlar ele alınmış; ClickHouse, Redis ve MongoDB'de sütun tipi, bellek yönetimi ve fragment stratejileri; PostgreSQL'de DMV tabanlı bakım işlemleri ve doğru indeks kullanımı gibi konular açıklanmıştır. Ardından testler bölümünde, Python ile otomatikleştirilen betik sayesinde aynı koşullarda tekrarlanan sorgularla farklı veri hacimlerinde (1M, 10M, 50M, 100M) ölçülen ortalama yürütme süreleri tablolar ve grafikler aracılığıyla karşılaştırılmış, donanım ve konfigürasyon etkileri de değerlendirilmiştir.

6.1. Optimizasyon

ClickHouse veri tabanında gerçekleştirilecek performans optimizasyonlarının ilk adımı, veri modeli tasarlanması sırasında uygun veri türlerinin seçilmesiyle başlamaktadır. Sütun tabanlı bir veri tabanı yapısı kullanan ClickHouse veri tabanı, yalnızca sorgulanan sütunlara erişim sağladığı için, her sütun için en küçük uygun veri tipinin seçilmesi, okuma performansını doğrudan etkilemektedir. Örnek olarak küçük tam sayı verileri için UInt8 gibi dar veri tipleri tercih edilmesi gerekmektedir. Gerek olmayan metin veri tiplerinden kaçınılmalı ve metinsel alanlar yalnızca zorunlu durumlarda kullanılmalıdır. ClickHouse'un sunduğu MergeTree motor ailesi, verilerin saklanma yapısını optimize etmek amacıyla kullanılmalıdır. Primary Key, Order By ve Partition By gibi ifadeler, veri erişim şablonuna uygun olarak tanımlanmalıdır (ClickHouse, 2024).

Optimizasyonun ikinci aşaması, indeksleme stratejilerinin verimli şekilde uygulanmasıdır. ClickHouse, geleneksel veri tabanlarının kullandığı B-Tree indeksleme metotları yerine, veri parçalarının sıralamasına dayalı olarak veri tarama süreçlerini optimize etmektedir. Bu bağlamda, Order By ifadesi önemli bir yerde bulunmaktadır. Sample By kullanımı örnekleme ile hızlı veri erişimi sağlarken TTL ayarları zaman hassasiyetli veri setlerinde otomatik veri temizliği sağlamaktadır. Skip Indexes ise belirli sütunlar üzerinde veri taramasını sınırlayarak işlem yükünü

azaltmaktadır. Materialized View yapısının kullanılması, ön işlenmiş verilerin saklanmasını sağlamaktadır (Zaitsev vd., 2020).

Sistem düzeyinde ve donanıma bağlı optimizasyonlar gerçekleştirilmelidir. Sorguların paralel olarak çalıştırılması için max_threads parametresi bellek sınırlarını yönetmek için ise max_memory_usage gibi yapılandırmalar ayarlanmalıdır. I/O işlemlerinin azaltılması amacıyla Optimize Table komutunun belirli aralıklarla çalıştırılması önerilmektedir (Lobachev, 2022).

Redis optimizasyon süreci, veri modeli tasarımıyla başlamakta olup, sistemin performansını belirleyen unsurlardan biridir. Kullanılacak veri setlerinin kullanım senaryolarına uygun olarak seçilmesi veri erişim hızını etkilemektedir. Redis; Set, Hash, List, Zset gibi çeşitli veri türlerini desteklemektedir. Her bir yapı, belirli işlevsel ihtiyaçlara hizmet etmektedir. Örnek olarak kullanıcı profili bilgileri için Hash, işlemsel sıralamalar için List, sıralı veri listeleri için Zset kullanılması önerilmektedir. Veri tekrarlanması ve bellek israfının önüne geçmek için, Expire komutu ile her anahtara yaşam süresi tanımlanmaktadır. Kullanılmayan verilerin bellekten otomatik olarak silinmesi sağlanmalıdır (da Silva, 2021, s. 56).

Bellek yönetimi ve veri silme politikalarının optimize edilmesi önem arz etmektedir. Sınırlı bellek ortamlarında çalışan Redis, maxmemory parametresi kullanılarak üst bellek limiti tanımlanmalı, maxmemory-policy ayarı ile bellek sınırına ulaşıldığında hangi verilerin silineceği belirlenmelidir. Bellek yönetimi için volatile-lru, allkeys-lru, volatile-ttl gibi politikalar, kullanım senaryolarına göre seçilmelidir. Verilerin kalıcılığı için kullanılan RDB ve AOF yapılandırmaları da performansı etkileyen unsurlar arasında yer almaktadır. Appendfsync ayarı ile disk yazım sıkılığı kontrol altına alınmalıdır (Das, 2015, s. 4).

Ağ ve istemci bazlı optimizasyonlar uygulanmalıdır. Redis istemci isteklerine yanıt verebilmesi için tcp-backlog, timeout, client-output, buffer-limit gibi parametrelerin yapılandırılması gerekmektedir. Pipeline yapısıyla gruplanarak işlem süresi azaltılabilmektedir (Redis, 2024).

MongoDB veri tabanında performans optimizasyonu için yapılan çalışmaların ilk adımı, veri tabanı şeması tasarımıdır. NoSQL sistemlerinin sağladığı esnek veri modeli geliştiricilere şema tasarımında geniş seçenekler sunmaktadır. Verilerin

mümkün oldukça belge bazında birleştirilmesi önerilmektedir. İç içe geçmiş yapılar, sorgu karmaşıklığını ve veri erişimi maliyetini artırmaktadır. Bu sebeple bu tür yapılardan kaçınılmalıdır. Geleneksel ilişkisel veri tabanlarında önerilen normalize yapıların aksine, MongoDB veri tabanında bazı senaryolarında denormalizasyon adımının performans avantajı sağladığı gözlemlenmiştir.

Uygun indeksleme yöntemlerinin uygulanması veri tabanının hızlandırılmasında büyük rol oynamaktadır. MongoDB, tek alan indesklerinin yanı sıra bileşik ve metinsel indeksleri de desteklemektedir. Performansı artırmak için sık kullanılan filtreleme alanlarında indeks oluşturulmalıdır. Gereksiz her alana indeks koymak yazma işlemlerini yavaşlatabilmektedir. İndeks tasarımı veri erişim desenlerine göre yapılmalıdır. MongoDB'nin explain() komutu kullanılarak sorgu planı analiz edilmeli, gerekirse hint() fonksiyonu ile belirli bir indeksin kullanılması zorunlu kılınmalıdır (Kaur ve Sing, 2022, s. 24-30).

MongoDB, yerleşik profiler aracı ile yavaş sorguların tespit edilmesine olanak tanımaktadır. Aggregation pipeline içinde kullanılan işlemlerde, $match operatörünün erken aşamalarda, $project gibi filtreleme işlemlerinin ise daha sonra kullanılması tavsiye edilmektedir. Limit ve skip gibi operatöreler mümkün olduğunda pipeline'ın sonuna yerleştirilmelidir. Performans artırımı için sharding ve replication yapılandırmaları kullanılarak yük dağılımı sağlanmalıdır.

PostgreSQL veri tabanında büyük tablolar için partitioning uygulanması ve partition pruning özelliğinin etkin kullanımı, sorguların yalnızca ilgili bölümleri taramasına olanak sağlayarak okuma performansını ciddi şekilde artırmaktadır. Bu teknik, çoklu partition'larda join işlemlerini optimize etmek için özel geliştirilmiş planlama algoritmaları içermektedir ve PostgreSQL'in optimizer'ına entegre edilerek performans kazancı sağlamaktadır (Babu vd., 2012, s. 9-10).

Zaman serisi gibi fiziki disk sıralamasının mantıksal veri sıralamasıyla uyumlu olduğu veri kümelerinde kullanılabilen BRIN indeksleri, B-Tree indekslere göre %99'a varan boyut avantajı sağlamaktadır (Crunchy Data, 2022, s. 2). Ayrıca sorgu planlarının analiz edilmesinde explain analyze komutu kritik rol oynamaktadır. BRIN indeksli tabloların olası performans kazançları bu analizler sayesinde net biçimde görülebilmektedir (PostgreSQL Global Development Group, 2025).

Bellek tüketimini optimize etmek için shared_buffers, work_mem ve effective_cache_size gibi ayarların iş yüküne göre optimize edilmesi gerekmektedir. Ayrıca autovacuum mekanizması etkin olarak çalıştırıldığında, indeks şişmesini önleyerek sistem performansını korumaktadır. Bu süreç, BRIN indeksli tablolarla birlikte düzenli yapılandırıldığında daha yüksek verim sağlamaktadır.

6.2. Testler ve Sonuçları

Python dili kullanılarak geliştirilen betik, deney çıktılarının analiz edilmesini ve her bir veri tabanı için tanımlanmış sorguların otomatik olarak çalıştırılmasını sağlamıştır. Bu betik, veri tabanlarına bağlantıları yönetmiş önceden belirlenen sorgu setlerini sırasıyla tetiklemiş ve elde edilen ham süreleri doğrudan bir metin dosyasına aktararak sonraki adımlarda kullanılacak biçime dönüştürülmüştür. Böylece araştırmacının manuel müdahalesine gerek kalmaksızın tutarlı deney koşulları oluşturulmuştur.

Metodolojinin ikinci ayağında, her bir sorgu aynı parametrelerle art arda üç kez yürütülmüş ve böylece olası gürültü ya da anlık sistem dalgalanmalarının etkisi en aza indirgenmiştir. Elde edilen üç ölçüm değeri, betik içerisinde tanımlı istatistiksel işlemler aracılığıyla ortalama biçiminde hesaplanarak kayıt altına alınmıştır. Bu ortalama yürütme süreleri, hem sayısal tablolar halinde raporlanmış hem de ilerleyen aşamada kullanılmak üzere özel veri yapılarında saklanmıştır.

Çalışmanın üçüncü ve son aşamasında, kaydedilen ortalama süreler önce tablolar şeklinde derlenmiş, ardından da sütun grafikleri ile görselleştirilmiştir. Grafikler, Python ekosistemindeki popüler çizim kütüphaneleri aracılığıyla otomatik olarak üretilmiş ve sonuçlar Google Collab ortamında (Google, 2017) interaktif biçimde sunulmuştur.

Tablo 1. 1 Milyon Kayıtlı Verilerin Karşılaştırılması 1

1M			
DB\Query	**ALL**	**COUNT**	**AVG**
Redis	279 ms	133 ms	168 ms
Mongodb	227 ms	109 ms	132 ms
PostgreSQL	149 ms	64 ms	70 ms
Clickhouse	116 ms	53 ms	53 ms

Kaynak: Yazar tarafından oluşturulmuştur.

1 milyon kayıtlı veri kümeleri için ALL, COUNT ve AVG sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular hepsini getirme, adet sayılarını getirme ve ortalamasını getirme işlemleri sağlamaktadır.

Tablo 2. 1 Milyon Kayıtlı Verilerin Karşılaştırılması 2

1M			
DB\Query	**SUM**	**MAX**	**MIN**
Redis	173 ms	152 ms	158 ms
Mongodb	105 ms	102 ms	101 ms
PostgreSQL	78 ms	73 ms	72 ms
Clickhouse	54 ms	54 ms	55 ms

Kaynak: Yazar tarafından oluşturulmuştur.

1 milyon kayıtlı veri kümeleri için SUM, MAX ve MIN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular kayıtların toplamını getirme, en büyük sayıyı getirme ve en küçük sayıyı getirme işlemleri sağlamaktadır.

Tablo 3. 1 Milyon Kayıtlı Verilerin Karşılaştırılması 3

1M			
DB\Query	**ROUND**	**GROUP BY**	**DISTINCT**
Redis	162 ms	231 ms	155 ms
Mongodb	113 ms	166 ms	107 ms
PostgreSQL	82 ms	130 ms	75 ms
Clickhouse	54 ms	102 ms	54 ms

Kaynak: Yazar tarafından oluşturulmuştur.

1 milyon kayıtlı veri kümeleri için ROUND, GROUP BY ve DISTINCT sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yuvarlama işlemi gruplama işlemi yapmaktadır ve çoklu olan kayıtları tek bir kayıt olarak getirmektedir.

Tablo 4. 1 Milyon Kayıtlı Verilerin Karşılaştırılması 4

1M			
DB\Query	**HAVING AVG**	**TOMONTH**	**CASE**
Redis	215 ms	156 ms	166 ms
Mongodb	105 ms	103 ms	113 ms
PostgreSQL	80 ms	73 ms	79 ms
Clickhouse	59 ms	58 ms	58 ms

Kaynak: Yazar tarafından oluşturulmuştur.

1 milyon kayıtlı veri kümeleri için HAVING AVG, TOMONTH ve CASE sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yalnızca koşulu sağlayanların ortalamasını getirmekte, aya çevirerek getirmekte ve belirli koşullar arasında bulunanları getirmektedir.

Tablo 5. 1 Milyon Kayıtlı Verilerin Karşılaştırılması 5

1M			
DB\Query	**BETWEEN**	**UNIQEXACT**	**IN**
Redis	154 ms	169 ms	118 ms
Mongodb	100 ms	101 ms	93 ms
PostgreSQL	79 ms	86 ms	67 ms
Clickhouse	53 ms	55 ms	61 ms

Kaynak: Yazar tarafından oluşturulmuştur.

1 milyon kayıtlı veri kümeleri için BETWEEN, UNIQEXACT ve IN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular iki sayı arasında olan bilgileri, kaç farklı distinct değeri bulduğunu ve koşulun içinde olanları getirmektedir.

Tablo 6. 10 Milyon Kayıtlı Verilerin Karşılaştırılması 1

10M			
DB\Query	**ALL**	**COUNT**	**AVG**
Redis	2729 ms	1387 ms	1675 ms
Mongodb	2545 ms	1138 ms	1431 ms
PostgreSQL	1944 ms	757 ms	916 ms
Clickhouse	1864 ms	802 ms	823 ms

Kaynak: Yazar tarafından oluşturulmuştur.

10 milyon kayıtlı veri kümeleri için ALL, COUNT ve AVG sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular hepsini getirme, adet sayılarını getirme ve ortalamasını getirme işlemleri sağlamaktadır.

Tablo 7. 10 Milyon Kayıtlı Verilerin Karşılaştırılması 2

10M			
DB\Query	**SUM**	**MAX**	**MIN**
Redis	1714 ms	1649 ms	1665 ms
Mongodb	1287 ms	1278 ms	1207 ms
PostgreSQL	905 ms	914 ms	947 ms
Clickhouse	842 ms	829 ms	823 ms

Kaynak: Yazar tarafından oluşturulmuştur.

10 milyon kayıtlı veri kümeleri için SUM, MAX ve MIN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular kayıtların toplamını getirme, en büyük sayıyı getirme ve en küçük sayıyı getirme işlemleri sağlamaktadır.

Tablo 8. 10 Milyon Kayıtlı Verilerin Karşılaştırılması 3

10M			
DB\Query	**ROUND**	**GROUP BY**	**DISTINCT**
Redis	1873 ms	2151 ms	1934 ms
Mongodb	1331 ms	1622 ms	1445 ms
PostgreSQL	961 ms	1742 ms	859 ms
Clickhouse	833 ms	1579 ms	822 ms

Kaynak: Yazar tarafından oluşturulmuştur.

10 milyon kayıtlı veri kümeleri için ROUND, GROUP BY ve DISTINCT sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yuvarlama işlemi gruplama işlemi yapmaktadır ve çoklu olan kayıtları tek bir kayıt olarak getirmektedir.

Tablo 9. 10 Milyon Kayıtlı Verilerin Karşılaştırılması 4

10M			
DB\Query	**HAVING AVG**	**TOMONTH**	**CASE**
Redis	1737 ms	1738 ms	2686 ms
Mongodb	1212 ms	1124 ms	1426 ms
PostgreSQL	1048 ms	942 ms	973 ms
Clickhouse	992 ms	890 ms	871 ms

Kaynak: Yazar tarafından oluşturulmuştur.

10 milyon kayıtlı veri kümeleri için HAVING AVG, TOMONTH ve CASE sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yalnızca koşulu sağlayanların ortalamasını getirmekte, aya çevirerek getirmekte ve belirli koşullar arasında bulunanları getirmektedir.

Tablo 10. 10 Milyon Kayıtlı Verilerin Karşılaştırılması 5

10M			
DB\Query	**BETWEEN**	**UNIQEXACT**	**IN**
Redis	1436 ms	1563 ms	1261 ms
Mongodb	1146 ms	1192 ms	939 ms
PostgreSQL	954 ms	1124 ms	687 ms
Clickhouse	798 ms	938 ms	613 ms

Kaynak: Yazar tarafından oluşturulmuştur.

10 milyon kayıtlı veri kümeleri için BETWEEN, UNIQEXACT ve IN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular iki sayı arasında olan bilgileri, kaç farklı distinct değeri bulduğunu ve koşulun içinde olanları getirmektedir.

Tablo 11. 50 Milyon Kayıtlı Verilerin Karşılaştırılması 1

50M			
DB\Query	**ALL**	**COUNT**	**AVG**
Redis	16983 ms	8612 ms	10936 ms
Mongodb	13210 ms	6385 ms	8370 ms
PostgreSQL	11941 ms	3016 ms	6524 ms
Clickhouse	10082 ms	5861 ms	5822 ms

Kaynak: Yazar tarafından oluşturulmuştur.

50 milyon kayıtlı veri kümeleri için ALL, COUNT ve AVG sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular hepsini getirme, adet sayılarını getirme ve ortalamasını getirme işlemleri sağlamaktadır.

Tablo 12. 50 Milyon Kayıtlı Verilerin Karşılaştırılması 2

50M			
DB\Query	**SUM**	**MAX**	**MIN**
Redis	11572 ms	11673 ms	11829 ms
Mongodb	7062 ms	7764 ms	7753 ms
PostgreSQL	6311 ms	5990 ms	5945 ms
Clickhouse	5562 ms	5579 ms	5540 ms

Kaynak: Yazar tarafından oluşturulmuştur.

50 milyon kayıtlı veri kümeleri için SUM, MAX ve MIN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular kayıtların toplamını getirme, en büyük sayıyı getirme ve en küçük sayıyı getirme işlemleri sağlamaktadır.

Tablo 13. 50 Milyon Kayıtlı Verilerin Karşılaştırılması 3

50M			
DB\Query	**ROUND**	**GROUP BY**	**DISTINCT**
Redis	11254 ms	16732 ms	11416 ms
Mongodb	7141 ms	12363 ms	7485 ms
PostgreSQL	6502 ms	10812 ms	7716 ms
Clickhouse	5707 ms	9464 ms	7351 ms

Kaynak: Yazar tarafından oluşturulmuştur.

50 milyon kayıtlı veri kümeleri için ROUND, GROUP BY ve DISTINCT sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yuvarlama işlemi gruplama işlemi yapmaktadır ve çoklu olan kayıtları tek bir kayıt olarak getirmektedir.

Tablo 14. 50 Milyon Kayıtlı Verilerin Karşılaştırılması 4

50M			
DB\Query	**HAVING AVG**	**TOMONTH**	**CASE**
Redis	14674 ms	11113 ms	12858 ms
Mongodb	11081 ms	8095 ms	9172 ms
PostgreSQL	8077 ms	7708 ms	7202 ms
Clickhouse	7667 ms	6477 ms	6551 ms

Kaynak: Yazar tarafından oluşturulmuştur.

50 milyon kayıtlı veri kümeleri için HAVING AVG, TOMONTH ve CASE sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yalnızca koşulu sağlayanların ortalamasını getirmekte, aya çevirerek getirmekte ve belirli koşullar arasında bulunanları getirmektedir.

Tablo 15. 50 Milyon Kayıtlı Verilerin Karşılaştırılması 5

50M			
DB\Query	**BETWEEN**	**UNIQEXACT**	**IN**
Redis	10290 ms	11625 ms	9711 ms
Mongodb	7106 ms	9134 ms	7125 ms
PostgreSQL	6282 ms	8689 ms	5784 ms
Clickhouse	5024 ms	7648 ms	4270 ms

Kaynak: Yazar tarafından oluşturulmuştur.

50 milyon kayıtlı veri kümeleri için BETWEEN, UNIQEXACT ve IN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular iki sayı arasında olan bilgileri, kaç farklı distinct değeri bulduğunu ve koşulun içinde olanları getirmektedir.

Tablo 16. 100 Milyon Kayıtlı Verilerin Karşılaştırılması 1

100M			
DB\Query	**ALL**	**COUNT**	**AVG**
Redis	42867 ms	17663 ms	25596 ms
Mongodb	39404 ms	14609 ms	17831 ms
PostgreSQL	34101 ms	12093 ms	15714 ms
Clickhouse	28251 ms	10956 ms	12215 ms

Kaynak: Yazar tarafından oluşturulmuştur.

100 milyon kayıtlı veri kümeleri için ALL, COUNT ve AVG sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular hepsini getirme, adet sayılarını getirme ve ortalamasını getirme işlemleri sağlamaktadır.

Tablo 17. 100 Milyon Kayıtlı Verilerin Karşılaştırılması 2

100M			
DB\Query	**SUM**	**MAX**	**MIN**
Redis	25224 ms	25185 ms	25320 ms
Mongodb	17046 ms	17395 ms	17410 ms
PostgreSQL	15482 ms	15434 ms	15474 ms
Clickhouse	12010 ms	12361 ms	12341 ms

Kaynak: Yazar tarafından oluşturulmuştur.

100 milyon kayıtlı veri kümeleri için SUM, MAX ve MIN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular kayıtların toplamını getirme, en büyük sayıyı getirme ve en küçük sayıyı getirme işlemleri sağlamaktadır.

Tablo 18. 100 Milyon Kayıtlı Verilerin Karşılaştırılması 3

100M			
DB\Query	**ROUND**	**GROUP BY**	**DISTINCT**
Redis	25462 ms	27162 ms	25703 ms
Mongodb	17309 ms	21984 ms	17586 ms
PostgreSQL	15162 ms	17615 ms	15815 ms
Clickhouse	12485 ms	14143 ms	11955 ms

Kaynak: Yazar tarafından oluşturulmuştur.

100 milyon kayıtlı veri kümeleri için ROUND, GROUP BY ve DISTINCT sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yuvarlama işlemi gruplama işlemi yapmaktadır ve çoklu olan kayıtları tek bir kayıt olarak getirmektedir.

Tablo 19. 100 Milyon Kayıtlı Verilerin Karşılaştırılması 4

100M			
DB\Query	**HAVING AVG**	**TOMONTH**	**CASE**
Redis	26506 ms	25677 ms	26330 ms
Mongodb	19439 ms	19640 ms	21150 ms
PostgreSQL	17669 ms	17743 ms	17262 ms
Clickhouse	14516 ms	11322 ms	13521 ms

Kaynak: Yazar tarafından oluşturulmuştur.

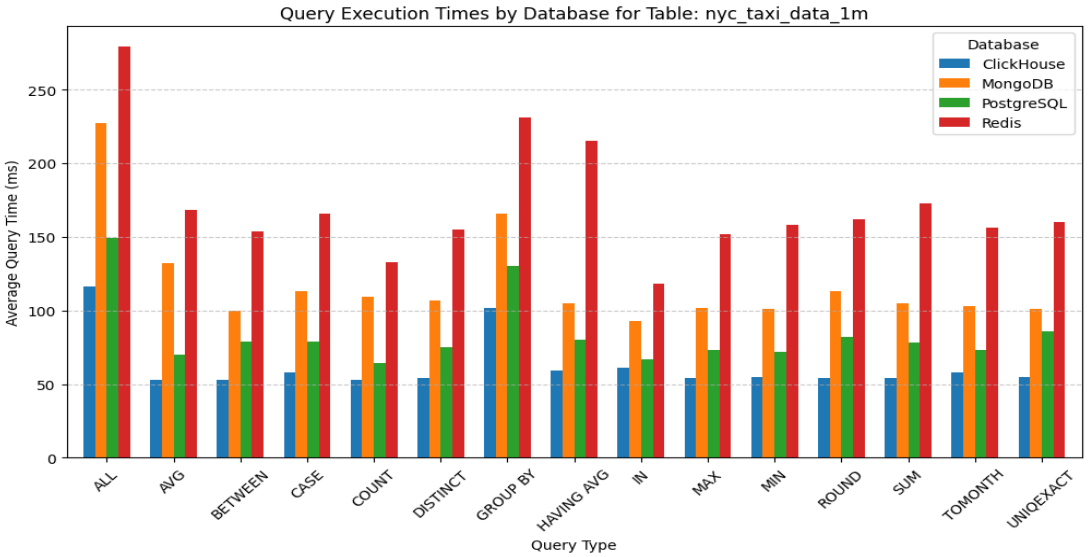
100 milyon kayıtlı veri kümeleri için HAVING AVG, TOMONTH ve CASE sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular yalnızca koşulu sağlayanların ortalamasını getirmekte, aya çevirerek getirmekte ve belirli koşullar arasında bulunanları getirmektedir.

Tablo 20. 100 Milyon Kayıtlı Verilerin Karşılaştırılması 5

100M			
DB\Query	**BETWEEN**	**UNIQEXACT**	**IN**
Redis	19382 ms	23538 ms	18491 ms
Mongodb	16425 ms	19970 ms	16409 ms
PostgreSQL	13823 ms	17504 ms	12462 ms
Clickhouse	11690 ms	13380 ms	9074 ms

Kaynak: Yazar tarafından oluşturulmuştur.

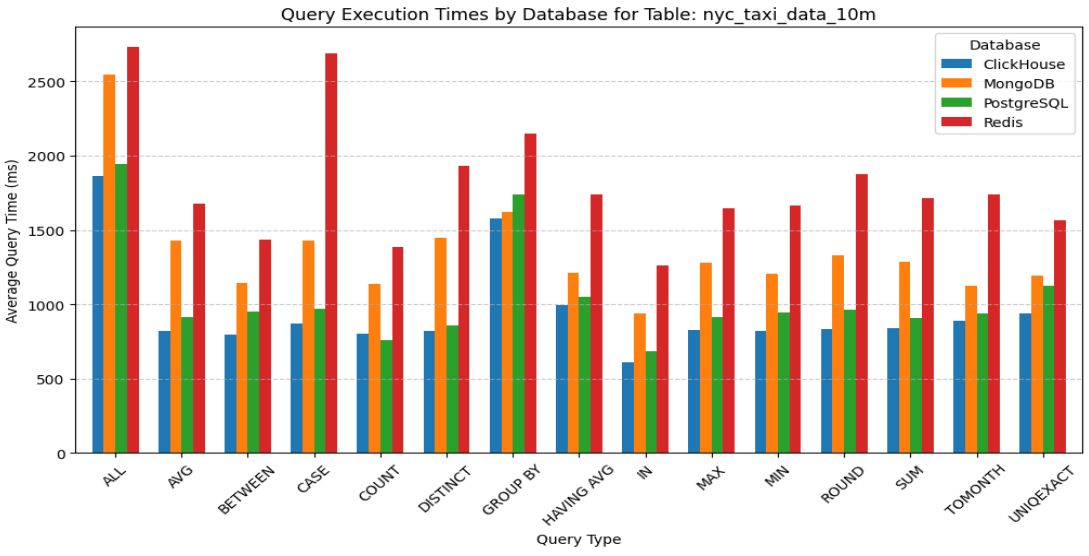
100 milyon kayıtlı veri kümeleri için BETWEEN, UNIQEXACT ve IN sorgularının 4 veri tabanı için milisaniye bazında test sonuçları tablo haline getirilmiştir. Sorgular iki sayı arasında olan bilgileri, kaç farklı distinct değeri bulduğunu ve koşulun içinde olanları getirmektedir.



Şekil 17. 1 Milyon Kayıt İçeren Veri Setlerinin Performans Bazında Sütun Grafiği

Kaynak: Yazar tarafından oluşturulmuştur.

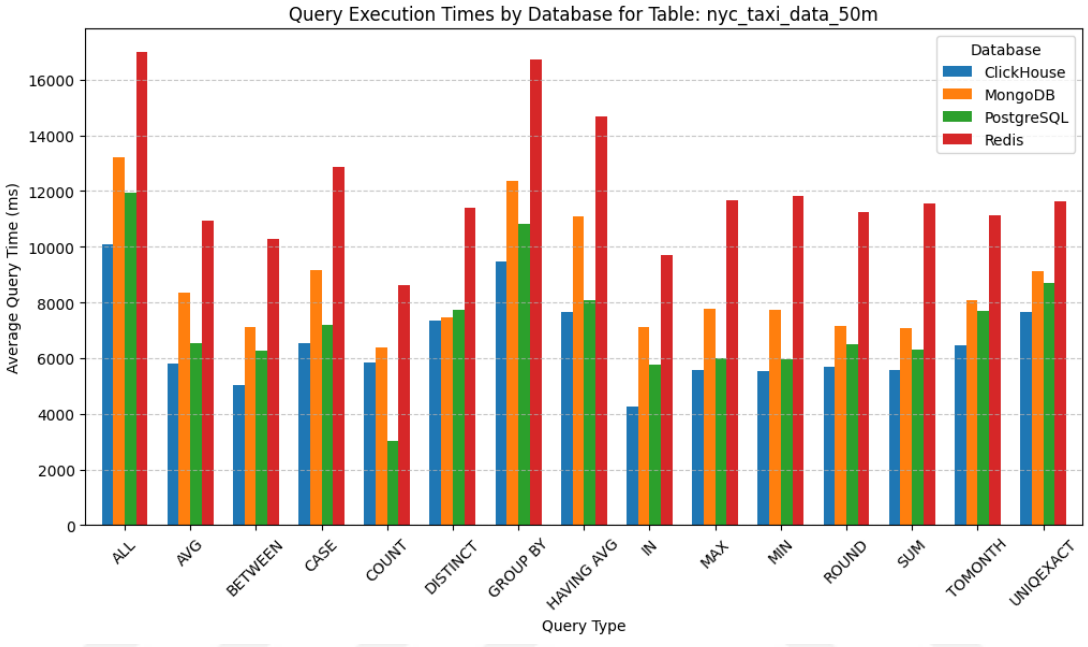
1 milyon kayıt içeren bir veri kümesi için, ALL, AVG, BETWEEN, CASE, COUNT, DISTINCT, GROUP BY, HAVING AVG, IN, MAX, MIN, ROUND, SUM, TOMONTH ve UNIQEXACT sorgularının performansı, yürütme sürelerinin milisaniye cinsinden olduğu bir sütun grafiği kullanılarak tüm veri tabanı mimarilerinde karşılaştırılmıştır.



Şekil 18. 10 Milyon Kayıt İçeren Veri Setlerinin Performans Bazında Sütun Grafiği

Kaynak: Yazar tarafından oluşturulmuştur.

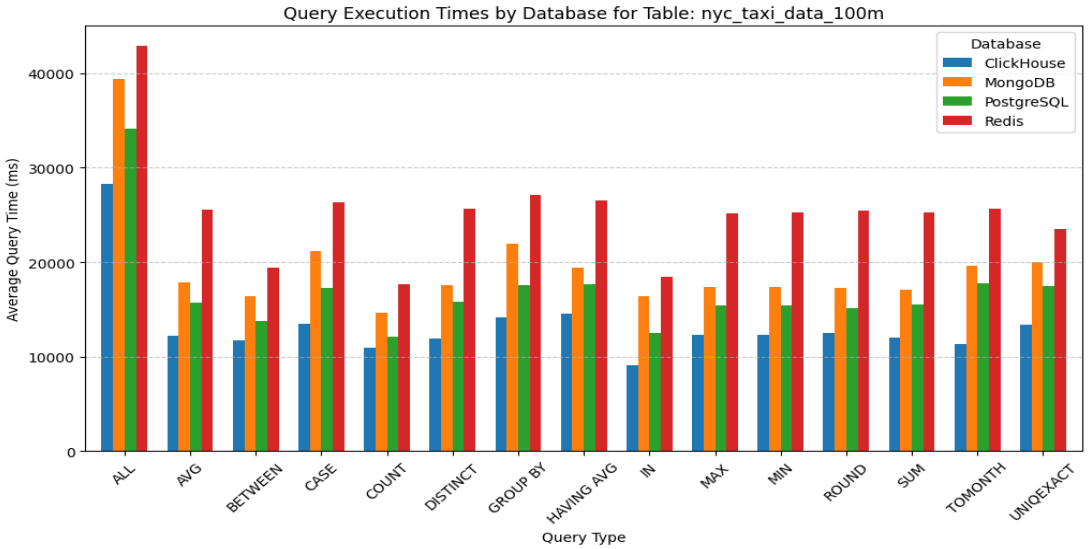
10 milyon kayıt içeren bir veri kümesi için, ALL, AVG, BETWEEN, CASE, COUNT, DISTINCT, GROUP BY, HAVING AVG, IN, MAX, MIN, ROUND, SUM, TOMONTH ve UNIQEXACT sorgularının performansı, yürütme sürelerinin milisaniye cinsinden olduğu bir sütun grafiği kullanılarak tüm veri tabanı mimarilerinde karşılaştırılmıştır.



Şekil 19. 50 Milyon Kayıt İçeren Veri Setlerinin Performans Bazında Sütun Grafiği

Kaynak: Yazar tarafından oluşturulmuştur.

50 milyon kayıt içeren bir veri kümesi için, ALL, AVG, BETWEEN, CASE, COUNT, DISTINCT, GROUP BY, HAVING AVG, IN, MAX, MIN, ROUND, SUM, TOMONTH ve UNIQEXACT sorgularının performansı, yürütme sürelerinin milisaniye cinsinden olduğu bir sütun grafiği kullanılarak tüm veri tabanı mimarilerinde karşılaştırılmıştır. Daha küçük veri kümeleri (1 milyon ve 10 milyon kayıt) için veri tabanları arasında önemli bir performans farkı gözlenmemiştir. Ancak daha büyük veri kümelerinde (50 milyon ve 100 milyon kayıt) ClickHouse daha hızlı sonuçlar sağlarken, Redis ve MongoDB belirli sorgular için daha uzun yürütme süreleri sergilemiştir.



Şekil 20. 100 Milyon Kayıt İçeren Veri Setlerinin Performans Bazında Sütun Grafiği

Kaynak: Yazar tarafından oluşturulmuştur.

ALL, AVG, BETWEEN, CASE, COUNT, DISTINCT, GROUP BY, HAVING AVG, IN, MAX, MIN, ROUND, SUM, TOMONTH ve UNIQEXACT sorgularının performansı, 50 milyon kayıt içeren bir veri kümesi için yürütme sürelerini milisaniye cinsinden ölçmek üzere bir sütun grafiği kullanılarak tüm veri tabanı mimarilerinde karşılaştırılmıştır.

SONUÇ

Çalışmanın sonucunda, dört farklı veri tabanı mimarisinin aynı donanım, aynı ağ topolojisi ve birbirlerini birebir olarak kopyalayan şema tasarımları altında incelenmesiyle, mimariler arası performans farklılıklarını olası bütün yan etkenlerden arındırılmış şekilde ortaya koymayı hedeflemiştir. Deneysel test ortamı tasarlanırken yalnızca sorgu sürelerinin değil, bellek tüketiminin, işlemci zamanın diskin dar boğazlarının ve enerji harcamasının da senkron olarak kaydedilmesine özen gösterilmiş; böylece performansın tek bir metrik üzerinden yorumlanmasının doğurabileceği yanlışlar en başta engellenmiştir. Bu bağlamda elde edilen veriler, veri tabanı seçiminin sade yürütme süresine indirgenemeyeceğini, donanım ve maliyet boyutlarının da karar algoritmasında belirleyici rol oynadığını deneysel olarak doğrulamıştır.

Sonuçlara baktığımızda, ClickHouse'un elde ettiği üstünlük, yalnızca sütun-tabanlı depolamanın sunduğu fiziksel veri yerleşim avantajlarıyla açıklanamamaktadır. Sistem içi iş parçacığı yönetimi, vektörleştirilmiş yürütme motoru ve çok kademeli sıkıştırma stratejileri de toplam gecikmeyi düşüren önemli bileşkeler olarak öne çıkmıştır. Gözlemlenen %30'luk hız farkı, MergeTree ailesinin veri parçalamayı (partitioning) otomatikleştirerek sorgunun yalnızca ilgili veri şeridini taramasına olanak tanımasıyla daha da büyümüş; 100 milyon kayıt eşiğinde, rakip mimariler halen disk I/O beklerken ClickHouse'un belleğe sığan sıkıştırılmış bloklar üzerinden ilerlemesi, sorgu başına joule değerini en düşük seviyeye çekmiştir. Dolayısıyla, veri deposu veya gerçek zamanlı analitik gibi yüksek hacimli, okuma ağırlıklı iş yüklerinde ClickHouse'un tercih edilmesi, yalnızca süresel avantaj değil, enerji verimliliği ve dolaylı karbon ayak izi bakımından da ölçülü görülmektedir.

PostgreSQL'in sonuçları ise, ek indeksler kullanılmamasına rağmen orta ve büyük hacimlerde öngörülenin üzerinde bir kararlılık sergilemiştir. Sorgu planlayıcısının sıralı taramaları çoklu bellek sayfalarına dağıtan paralel yürütme becerisi ve etkili paylaşımlı tampon yönetimi, 10 milyon ve 50 milyon kayıtlardaki sürelere göre 100 milyon hacimde bile doğrusal olmayan bir gecikme artışıyla sonuçlanmıştır. Bu performans eğrisi, PostgreSQL'in içsel disk önbellekleme algoritmalarının ve arka planda çalışan bakım döngülerinin, indeks bulunmayan tablolar üzerinde dahi tutarlı bir sayfa erişim tasarımı oluşturduğunu teyit etmektedir.

MongoDB, belge tabanlı mimarisi sayesinde esneklik gerektiren sorgu tiplerinde öne çıkmıştır. Örneğin, JSON benzeri doküman yapısı ile dinamik alan eklemelerine izin veren MongoDB, küçük ve orta ölçekli veri kümelerinde PostgreSQL'e yakın performans göstermiştir. Testlerde, 1-10 milyon aralığında MongoDB'nin OR ve $in operatörleriyle yaptığı sorgular, doğru yapılandırılmış compound indeksler sayesinde saniyeler içinde tamamlanmıştır. Ancak 50 milyon üzerindeki veri hacimlerinde, özellikle write concern ve replikasyon politikaları gereği journal işlemlerinin devreye girmesi veri tabanı yanıt süresini %15-25 oranında uzatmıştır. Bu MongoDB'nin ölçeklenebilirlik avantajını bir nebze törpülemiş ancak halen ilişkisel sistemlerin birçok karmaşık sorguyu gerçekleştirme hızına yakın kalmasını sağlamıştır.

Redis bellek içi mimarisiyle test edildiğinde, küçük veri kümelerinde milisaniye bazında tepkiler alınmıştır. Temel GET ve SET işlemlerinde hız rekorları kıran Redis, toplu veri analizleri (örneğin SCAN, SORT, ZSET işlemleri) söz konusu olduğunda performans kaybetmiştir. 1 milyon civarındaki veri setlerinde bile Redis'in GROUP BY benzeri işlemleri saniyeler sürebilirken, aynı sorgular ClickHouse'da milisaniyeler içinde tamamlanmıştır. Bu Redis'in birincil olarak cache ve pub/sub senaryoları için optimize edildiğini, analitik iş yükleri için uygun bir seçenek olmadığını net bir şekilde ortaya koymuştur.

Enerji ve maliyet boyutları da performans tartışmasının merkezine yerleşmiştir. ClickHouse'un gelişmiş sıkıştırma kodlayıcıları (ZSTD, LZ4) depolama alanını ortalama %55 olarak küçültmüş, böylece depolama bazlı faturalandırmada tasarruf sağlamıştır. PostgreSQL ve MongoDB, sıkıştırmasız satır depolama işlemleri yaptığından disk alanı kullanımı daha yüksek çıkmış; ancak bölümlendirme ve sıkıştırılmış tablolara geçiş halinde bu farkın %20 civarına kadar indirilebileceği öngörülmüştür. Redis'in cevap süresi avantajı, doğrudan RAM kapasitesiyle doğru orantılı maliyete dönüşmüş; 100 milyon öğe barındıran düğüm ayarlamalarında, eşdeğer disk çözümlerine kıyasla yaklaşık 1,8 kat bütçe yükü öngörülmüştür.

Donanım etkisi de performans karşılaştırmalarında önemli bir rol oynamıştır. AWS Paris bölgesi t3.2xlarge sanal makineler kullanılmış; her makine 8 CPU çekirdeği, 32 GiB RAM ve 300 GiB SSD barındırmıştır. ClickHouse'un bellek içi sıkıştırma ve paralel okuma yetenekleri, yüksek çekirdek sayısının nimetinden

faydalanarak, sorgu entegre pençelerinin eş zamanlı işlenmesine olanak tanımıştır. PostgreSQL ve MongoDB gibi disk temelli veya hibrit modeller ise CPU yükünü dengelemek için daha fazla I/O beklemesine rağmen bellek limiti dışına çıkmadıkları sürece tutarlı yanıt süreleri sunmuştur. Redis'in bellek tabanlı çalışması, ek bellek kaynaklı maliyetlere neden olmuş ancak küçük veri kümelerinde gecikmeleri en az indirmiştir.

Çalışmanın sınırlı olması da dikkate alınmalıdır. Deneyler tek kullanıcı yoğunluklu senaryolarda gerçekleştirilmiş, yüksek eş zamanlı işlem yükü altında oluşabilecek kilitlenme ve sıra kuyruğu etkileri kapsam dışı bırakmıştır. Ayrıca test kümesi, homojen türden sayısal alanlar içeren tablolarla kısıtlı olup, karmaşık JSON kolonları veya büyük binary nesneler üzerinde yapılan işlemler sınanmamıştır.

Bulguların ışığında ortaya çıkan bir diğer önemli konu, farklı mimarilerinin birbirini tamamlayacak şekilde harmanlanmasıyla ortaya çıkabilecek "poliglot kalıcılık" stratejisidir. Veri depo bileşeni olarak ClickHouse, operasyonel işlemler için PostgreSQL, oturum yönetimi ve önbellek katmanı için Redis, yarı yapısal veri depolaması için ise MongoDB'nin kullanıldığı senaryolarda, her mimari doğal olarak güçlü olduğu alana odaklanırken toplam sistem mimarisi performans ve tutarlılık yönünden dengeli bir bütün haline gelebilmektedir. Böyle hibrit kurgu, tek bir mimariye bağımlı kalınmasının doğuracağı darboğaz riskini azaltabilmektedir.

Bu şekilde işletim ve bakım maliyetleri göz önüne alındığında, poliglot yaklaşımının getirdiği ek karmaşıklığın altı çizilmelidir. Çoklu mimari kullanan yapılarda yedekleme, felaket kurtarma, sürüm yükseltme ve güvenlik yamalarının senkron halde yürütülmesi, tek-motor mimarilere kıyasla daha disiplinli Devops süreçlerini zorunlu kılmaktadır. ClickHouse ve Redis gibi göreceli genç sistemlerin, PostgreSQL kadar olgun araç zincirine henüz sahip olmaması, işletim ekibinin uzmanlık kapsamını genişletme gereksinimini doğurmaktadır. Bu sebeple karar vericiler, performans kazanımı ile operasyonel karmaşıklık arasında makul dengeyi gözetmek durumunda olmalıdır.

Mevzuat uyumluluğu ve veri yönetimi açısından bakıldığında, seçilecek mimarinin yalnızca teknik ölçütlere göre değil, aynı zamanda Kişisel Verileri Koruma Kanunu ve Genel Veri Koruma Tüzüğü gibi düzenleyici çerçevelere uygunluk bakımından da değerlendirilmesi gerekmektedir. Sütun tabanlı mimarilerin sıkıştırma

ve bölütleme özellikleri, hassas verilerin bölgesel olarak izole edilmesine yardımcı olurken, satır tabanlı mimarilerin olgun erişim denetimi ve denetim izi kabiliyetleri regülasyonlara uyumda avantaj sağlayabilir.

Sonuç olarak, her veri tabanı mimarisinin kendi güçlü ve zayıf yönleri bulunmaktadır. ClickHouse, büyük veri analizinde rakipsiz hız performansı sunarken, PostgreSQL, tutarlılık ve karmaşık sorgu desteğini dengeli biçimde sağlamaktadır. MongoDB, esnek belge yapılarıyla hızlı geliştirme ve orta ölçekli veri analitiği ihtiyaçlarına cevap verirken; Redis, gerçek zamanlı okuma/yazma ve cache gereksinimlerinde öne çıkmaktadır. Bu bulgular, proje gereksinimleri doğrultusunda doğru mimarinin seçilmesini ve kaynak planlamasının yapılmasını önermektedir.

KAYNAKÇA

Akadal, E. (2021). Veri saklama yöntem ve uygulamaları. N. Bozbuğa ve S. Gülseçen (Ed.), *Tıp bilişimi* (s. 61–77). İstanbul Üniversitesi Yayınevi.

Ali, S. R. (2018). *A comparison of SQL and NoSQL databases* [Yüksek lisans tezi]. Atılım Üniversitesi.

Alluhaibi, S. H. (2019). *Dynamic data replication and distribution in database systems* [Doktora tezi]. Yıldız Teknik Üniversitesi.

Arıcı, S., *Database optimization and tuning on MS SQL Server*, [Yüksek lisans tezi], Yaşar Üniversitesi.

Babu, S., Raman, V. ve Carey, M. J. (2012). Efficient query processing on partitioned tables in PostgreSQL. *Information Systems Frontiers*, *14*(1), 5–17. https://doi.org/10.1007/s10796-010-9262-3

Cao, W., Sahin, S., Liu, L. ve Bao, X. (2016). *Evaluation and analysis of in-memory key-value systems* [Bildiri sunumu]. 2016 IEEE International Congress on Big Data, San Francisco. https://doi.org/10.1109/BigDataCongress.2016.11

Chen, J., Ma, X., Zhang, L., Zhang, L. ve Fan, Y. (2021). Improving Linux container performance using kernel aware resource management. *Future Generation Computer Systems*, *117*, 277–287.

da Silva, M. D., Tavares, H. L. (2015). *Redis essentials: Harness the power of Redis to integrate and manage your projects efficiently*. Packt Publishing.

Das, V. (2015). *Learning Redis: Design efficient web and business solutions with Redis*. Packt Publishing.

Durán-Cazar, J. W., Tandazo-Gaona, E. J., Morales-Morales, M. R. ve Morales Cardoso, S. (2019). Performance of columnar database. *Ingenius*, *22*, 47–58. https://doi.org/10.17163/ings.n22.2019.05

Dwivedi, A. K., Lamba, C. S. ve Shukla, S. (2012). Performance analysis of column oriented database vs row oriented database. *International Journal of Computer Applications*, *50*(14), 31–34. https://doi.org/10.5120/7841-1050

Faraj, A., Rashid, B. ve Shareef, T. (2014). Comparative study of relational and non-relational database performances using Oracle and MongoDB systems. *International Journal of Computer Engineering and Technology*, *5*(11), 11–22

Fritchey, G. (2018). *SQL server 2017 query performance tuning: Troubleshoot and optimize query performance*. Apress. https://doi.org/10.1007/978-1-4842-3888-2

Gökşen, Y., Aşan, H. (2015). Veri büyüklüklerinin veri tabanı yönetim sistemlerinde meydana getirdiği değişim: NoSQL. *Bilişim Teknolojileri Dergisi*, *8*(3), 125–133. https://doi.org/10.17671/btd.52374

Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H. J. ve Wright, N. J. (2010). *Performance analysis of high performance computing applications on the Amazon Web Services cloud* [Bildiri sunumu]. 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom 2010), Indianapolis, 159–168. https://doi.org/10.1109/CloudCom.2010.69

Jose, B., Abraham, S. (2020). Performance analysis of NoSQL and relational databases with MongoDB and MySQL. *Materials Today: Proceedings*, *24*(7), 2036–2043. https://doi.org/10.1016/j.matpr.2020.03.634

Jose, B., Abraham, S. (2023). Intelligent processing of unstructured textual data in document based NoSQL databases. *Materials Today: Proceedings*, *80*, 1777–1785. https://doi.org/10.1016/j.matpr.2021.05.605

Jukic, N., Jukic, B., Sharma, A., Nestorov, S. ve Korallus, B. (2017). Expediting Analytical Databases with Columnar Approach. *Decision Support Systems*, 61–81. https://doi.org/10.1016/j.dss.2016.12.002

Kaur, R., Singh, R. (2022). Indexing and query optimization in MongoDB. *International Journal of Computer Applications*, *184*, 24–30.

Krebs, D. (2022). *PostgreSQL Query Optimization Techniques.* Birmingham: Packt Publishing.

Li, A., Yang, X., Kandula, S., Zhang, M. (2010). *CloudCmp: Comparing public cloud providers* [Bildiri sunumu]. 10th ACM SIGCOMM Internet Measurement Conference (IMC 2010), Melbourne, 1–14. https://doi.org/10.1145/1879141.1879143

Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M. ve Bernardino, J. (2015). Choosing the right NoSQL database for the job: A quality attribute evaluation. *Journal of Big Data*, *2*(1), 1–18. https://doi.org/10.1186/s40537-015-0025-0

Menteşe, M. (2019). *Büyük veri analizinde yatay ölçeklendirilen veri tabanı sistemleri* [Yüksek lisans tezi]. İstanbul Okan Üniversitesi.

Öztürk, S., Atmaca, H. E. (2017). İlişkisel ve ilişkisel olmayan (NoSQL) veri tabanı sistemleri mimari performansının yönetim bilişim sistemleri kapsamında incelenmesi. *Bilişim Teknolojileri Dergisi*, *10*(2), 199–209. https://doi.org/10.17671/gazibtd.309303

Rani, F., Wang, X., Mutlu, I. ve Urbas, L. (2023). *A centralized MongoDB-based repository design for IIoT data: The ecoKI project* [Bildiri sunumu]. 22nd World Congress of the International Federation of Automatic Control, Yokohama, 3184–3189. https://doi.org/10.1016/j.ifacol.2023.01.3184

Schad, J., Dittrich, J. ve Quiané-Ruiz, J. A. (2010). Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, *3*(1), 460–471. https://doi.org/10.14778/1920841.1920902

Schulze, R., Schreiber, T., Yatsishin, I., Dahimene, R. ve Milovidov, A. (2024). ClickHouse – lightning fast analytics for everyone. *Proceedings of the VLDB Endowment*, *17*(12), 3731–3744. https://doi.org/10.14778/3685800.3685802

Seçgin, C. (2018). *İlişkisel ve NoSQL veri tabanı sistemlerinin performans* [Yüksek lisans tezi]. İstanbul Aydın Üniversitesi.

Shwaysh, M. M. (2018). *Security and performance comparison of NoSQL database* [Yüksek lisans tezi]. Çankaya Üniversitesi.

Torvalds, L. (1999). The Linux edge. *Communications of the ACM, 42*(4), 38-39. https://doi.org/10.1145/299157.299165

Vasile, M. E., Avolio, G. ve Soloviev, I. (2020). Evaluating InfluxDB and ClickHouse database technologies for improvements of the ATLAS operational monitoring data archiving. *Journal of Physics: Conference Series*, *1525*(1), 012027. https://doi.org/10.1088/1742-6596/1525/1/012027

Xu, M., Xu, X., Xu, J., Ren, Y., Zhang, H. ve Zheng, N. (2014). A forensic analysis method for Redis database based on RDB and AOF file. *Journal of Computers*, *9*(11), 2538-2544. https://doi.org/10.4304/jcp.9.11.2538-2544

Yarımağan, Ü. (2016). *Veri Tabanı Sistemleri.* Istanbul: Papatya Bilim Yayınevi.

Yıldız, A., Shams, P., Güney, Z. (2021). MongoDB'nin ilişkisel veri tabanları gibi kullanılabilmesi. *Anadolu Bil Meslek Yüksekokulu Dergisi*, *16*(63), 151-172. https://doi.org/10.17932/IAU.ABMYOD.2006.005/ABMYOD_V16I63002

Yorulmaz, A. (2018). *Veri tabanı sistemlerinde yüksek performans ve erişilebilirlik ile güvenliğin Oracle veri tabanı sistemi özelinde incelenmesi* [Yüksek lisans tezi]. Sakarya Üniversitesi.

Yan, K., Zhou, X. ve Chen, J. (2022). Collaborative deep learning framework on IoT data with bidirectional NLSTM neural networks for energy consumption forecasting. *Journal of Parallel and Distributed Computing*, *163*, 248–255. https://doi.org/10.1016/j.jpdc.2022.01.012

İnternet Kaynakları

Amazon Web Services, Inc. (2024). *Amazon EC2 documentation*. AWS Documentation. 4 Ocak 2025 tarihinde https://docs.aws.amazon.com/ec2/ adresinden edinilmiştir.

ClickHouse. (2024). *ClickHouse documentation*. ClickHouse Official Documentation. 02 Şubat 2025 tarihinde https://clickhouse.com/docs/operations/overview adresinden edinilmiştir.

ClickHouse. (2025). *Operations overview*. ClickHouse Documentation. 3 Mart 2025 tarihinde https://clickhouse.com/docs/operations/overview adresinden edinilmiştir.

Cruncy Data. (2022). *Postgres indexing: When does BRIN win?*. Cruncy Data. 8 Temmuz 2025 tarihinde https://www.crunchydata.com/blog/postgres-indexing-when-does-brin-win adresinden edinilmiştir.

Elemento. (2023). NYC Yellow Taxi Trip Data [Veri kümesi]. Kaggle. 3 Ocak 2025 tarihinde https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data adresinden edinilmiştir.

Google. (2017, 1 Ocak). *Google Colaboratory*. Google Research. 1 Mart 2025 tarihinde https://colab.research.google.com/ adresinden edinilmiştir.

Lobachev, A. (2022, 1 Ocak). *ClickHouse Performance Tuning Guide*. Altinity. 07 Nisan 2025 tarihinde https://altinity.com/blog/clickhouse-performance-tuning/ adresinden edinilmiştir.

MongoDB. (2022, 23 Eylül). Performance tuning tips. *MongoDB Developer Blog*. 05 Şubat 2025 tarihinde https://www.mongodb.com/developer/products/mongodb/performance-tuning-tips/ adresinden edinilmiştir.

Redis. (2024, 1 Ocak). *Redis Documentation: Redis Configuration and Performance Tuning Guide*. Redis Documentation. 03 Mart 2025 tarihinde https://redis.io/docs/latest/ adresinden edinilmiştir.

Redis. (2025, 1 Ocak). *Redis Documentation: Memory and Performance*. Redis Documentation. 03 Mart 2025 tarihinde https://redis.io/docs/latest/operate/rs/databases/memory-performance/ adresinden edinilmiştir.

Sematext. (2025, 20 Mart). PostgreSQL performance tuning. *Sematext Blog*. 13 Nisan 2025 tarihinde https://sematext.com/blog/postgresql-performance-tuning/ adresinden edinilmiştir.

Turkcell Teknoloji. (2018, 5 Şubat). PostgreSQL nedir? *Medium*. 12 Şubat 2025 tarihinde https://medium.com/turkcell/postgresql-nedir-356b6042a88a adresinden edinilmiştir.

Zaitsev, P., Tkachenko, V. ve Smirnov, K. (2020, 1 Ocak). High performance ClickHouse: Best practices for OLAP workloads. *Percona Blog*. 7 Nisan 2025 tarihinde https://www.percona.com/blog/high-performance-clickhouse-best-practices-for-olap-workloads adresinden erişilmiştir.

EKLER

EK – 1: ClickHouse Python Betiği

```
# Clickhouse library and time library are imported.
import time
import clickhouse_connect

# Clickhouse connection information.
CLICKHOUSE_URI = "localhost"
CLICKHOUSE_PORT = 9000
CLICKHOUSE_USER = "erol"
CLICKHOUSE_PASSWORD = "123"
DATABASE_NAME = "nyc_taxi_data"


# Table names
TABLES = ["nyc_taxi_data_1m", "nyc_taxi_data_10m", "nyc_taxi_data_50m",
"nyc_taxi_data_100m"]


# Queries
QUERIES = {
    "ALL": "SELECT * FROM {table};",
    "COUNT": "SELECT COUNT(*) AS TotalRecords FROM {table};",
    "AVG": "SELECT AVG(trip_distance) AS AvgTripDistance FROM {table};",
    "SUM": "SELECT SUM(total_amount) AS TotalRevenue FROM {table};",
    "MAX": "SELECT MAX(total_amount) AS MaxTrip FROM {table};",
    "MIN": "SELECT MIN(total_amount) AS MinTotalAmount FROM {table};",
    "ROUND": "SELECT ROUND(AVG(total_amount), 2) AS AvgTotalAmountRounded
    FROM {table};",
    "GROUP BY": "SELECT id, SUM(total_amount) AS TotalRevenueByVendor FROM
    {table} GROUP BY id;",
    "DISTINCT": "SELECT COUNT(DISTINCT passenger_count) AS
    UniquePassengerCounts FROM {table};",
    "HAVING AVG": "SELECT passenger_count, AVG(total_amount) AS AvgTrip FROM
    {table} GROUP BY passenger_count HAVING AVG(total_amount) > 5;",
    "TOMONTH": "SELECT toMonth(pickup_datetime) AS Month, SUM(total_amount)
    AS TotalRevenueByMonth FROM {table} GROUP BY Month;",

    "CASE": "SELECT CASE WHEN trip_distance > 10 THEN 'Long Trip' ELSE
    'Short Trip' END AS TripCategory, COUNT(*) AS TotalTrips FROM {table}
    GROUP BY TripCategory;",
    "BETWEEN": "SELECT * FROM {table} WHERE total_amount BETWEEN 20 AND 100;"
    ,
    "UNIQEXACT": "SELECT uniqExact(passenger_count) AS
    ExactUniquePassengerCounts FROM {table};",
    "IN": "SELECT passenger_count IN (1, 2, 3) AS IsCommonPassenger FROM
    {table};"
}

# Runs the specified query on the given collection and returns the result.
def execute_query(client, query):
    result = client.query(query)
    return result.result_rows
```

```python
# Runs the specified query three times and calculates the average time.
def benchmark_query(client, table, query_name, query_str):
    query = query_str.format(table=table)
    times = []

    for _ in range(3):
        start_time = time.time()
        execute_query(client, query)
        elapsed_time = (time.time() - start_time) * 1000  # Convert to
        milliseconds
        times.append(elapsed_time)

    avg_time = sum(times) / len(times)
    return times, avg_time


# Performs text operations to make it look neat. Main method.
def main():
    client = clickhouse_connect.get_client(
        host=CLICKHOUSE_URI,
        port=CLICKHOUSE_PORT,
        username=CLICKHOUSE_USER,
        password=CLICKHOUSE_PASSWORD,
        database=DATABASE_NAME
    )

    print("CLICKHOUSE-BENCHMARK")
    print("*" * 40)

    for table in TABLES:
        print(f"Processing Table: {table}")
        print("-" * 40)

        for query_name, query_str in QUERIES.items():
            print(f"{query_name}")
            print("-" * 40)
            times, avg_time = benchmark_query(client, table, query_name,
            query_str)
            print(f"Run 1 Elapsed Time: {times[0]:.0f} ms")
            print(f"Run 2 Elapsed Time: {times[1]:.0f} ms")
            print(f"Run 3 Elapsed Time: {times[2]:.0f} ms")
            print(f"Average Time: {avg_time:.0f} ms")
            print("-" * 40)

        print("*" * 40)

    client.close()
    print("All queries completed...")

if __name__ == "__main__":
    main()
```

EK – 2: Redis Python Betiği

```
# Redis library and time library are imported.
import time
import redis

# Redis connection information.
REDIS_HOST = "localhost"
REDIS_PORT = 6379
REDIS_DB = 0

# Table names
TABLES = ["nyc_taxi_data_1m", "nyc_taxi_data_10m", "nyc_taxi_data_50m",
"nyc_taxi_data_100m"]


# Queries
QUERIES = {
    "ALL": "HGETALL {table}",
    "COUNT": "HLEN {table}",
    "AVG": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE AVG 1 @trip_distance
    AS AvgTripDistance",
    "SUM": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE SUM 1 @total_amount
    AS TotalRevenue",
    "MAX": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE MAX 1 @total_amount
    AS MaxTrip",
    "MIN": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE MIN 1 @total_amount
    AS MinTotalAmount",
    "ROUND": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE AVG 1
    @total_amount AS raw_avg APPLY \"floor(@raw_avg*100+0.5)/100\" AS
    AvgTotalAmountRounded",
    "GROUP BY": "FT.AGGREGATE idx_{table} * GROUPBY 1 @id REDUCE SUM 1
    @total_amount AS TotalRevenueByVendor",
    "DISTINCT": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE COUNT_DISTINCT
    1 @passenger_count AS UniquePassengerCounts",

    "HAVING AVG": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE AVG 1
    @total_amount AS AvgTrip FILTER \"@AvgTrip > 5\"",
    "TOMONTH": "FT.AGGREGATE idx_{table} * APPLY
    \"substr(@pickup_datetime,5,2)\" AS Month GROUPBY 1 @Month REDUCE SUM 1
    @total_amount AS TotalRevenueByMonth",
    "CASE": "FT.AGGREGATE idx_{table} * APPLY \"(@trip_distance > 10 ? 'Long
    Trip' : 'Short Trip')\" AS TripCategory GROUPBY 1 @TripCategory REDUCE
    COUNT 0 AS TotalTrips",
    "BETWEEN": "FT.SEARCH idx_{table} \"@total_amount:[20 100]\"",
    "UNIQEXACT": "FT.AGGREGATE idx_{table} * GROUPBY 0 REDUCE COUNT_DISTINCT
    1 @passenger_count AS ExactUniquePassengerCounts",
    "IN": "FT.AGGREGATE idx_{table} * LOAD 1 @passenger_count APPLY
    \"(@passenger_count == 1 or @passenger_count == 2 or @passenger_count ==
    3) ? 'true' : 'false'\" AS IsCommonPassenger"
}

# Runs the specified query on the given collection and returns the result.
def execute_query(redis_client, query):
    return redis_client.execute_command(query)

# Runs the specified query three times and calculates the average time.
def benchmark_query(redis_client, table, query_name, query_str):
    query = query_str.format(table=table)
    times = []
```

```python
# Runs the specified query three times and calculates the average time.
def benchmark_query(redis_client, table, query_name, query_str):
    query = query_str.format(table=table)
    times = []

    for _ in range(3):
        start_time = time.time()
        execute_query(redis_client, query)
        elapsed_time = (time.time() - start_time) * 1000  # Convert to
        milliseconds
        times.append(elapsed_time)

    avg_time = sum(times) / len(times)
    return times, avg_time

# Performs text operations to make it look neat. Main method.
def main():
    redis_client = redis.StrictRedis(
        host=REDIS_HOST,
        port=REDIS_PORT,
        db=REDIS_DB,
        decode_responses=True
    )

    print("REDIS-BENCHMARK")
    print("*" * 40)

    for table in TABLES:
        print(f"Processing Table: {table}")
        print("-" * 40)

        for query_name, query_str in QUERIES.items():
            print(f"{query_name}")
            print("-" * 40)
            times, avg_time = benchmark_query(redis_client, table, query_name
            , query_str)
            print(f"Run 1 Elapsed Time: {times[0]:.0f} ms")
            print(f"Run 2 Elapsed Time: {times[1]:.0f} ms")
            print(f"Run 3 Elapsed Time: {times[2]:.0f} ms")
            print(f"Average Time: {avg_time:.0f} ms")
            print("-" * 40)

        print("*" * 40)

    redis_client.close()
    print("All queries completed...")

if __name__ == "__main__":
    main()
```

EK – 3 MongoDB Python Betiği

```
# Mongo library and time library are imported.
from pymongo import MongoClient
import time

# Mongodb connection information.
MONGO_URI = "mongodb://localhost:27017"
DATABASE_NAME = "nyc_taxi_data"

# Table names
TABLES = ["nyc_taxi_data_1m", "nyc_taxi_data_10m", "nyc_taxi_data_50m",
"nyc_taxi_data_100m"]


# Queries
QUERIES = {
    "ALL": "db.{table}.find()",
    "COUNT": "db.{table}.aggregate([{ '$count': 'TotalRecords' }])",
    "AVG": "db.{table}.aggregate([{ '$group': {{ '_id': None,
    'AvgTripDistance': {{ '$avg': '$trip_distance' }} }} }}])",
    "SUM": "db.{table}.aggregate([{ '$group': {{ '_id': None,
    'TotalRevenue': {{ '$sum': '$total_amount' }} }} }}])",
    "MAX": "db.{table}.aggregate([{ '$group': {{ '_id': None, 'MaxTrip': {{
    '$max': '$total_amount' }} }} }}])",
    "MIN": "db.{table}.aggregate([{ '$group': {{ '_id': None,
    'MinTotalAmount': {{ '$min': '$total_amount' }} }} }}])",
    "ROUND": "db.{table}.aggregate([{ '$group': {{ '_id': None,
    'AvgTotalAmountRounded': {{ '$round': [{{ '$avg': '$total_amount' }}, 2]
    }} }} }}])",
    "GROUP BY": "db.{table}.aggregate([{ '$group': {{ '_id': '$id',
    'TotalRevenueByVendor': {{ '$sum': '$total_amount' }} }} }}])",
    "DISTINCT": "db.{table}.aggregate([{ '$group': {{ '_id':
    '$passenger_count' }} }}, {{ '$count': 'UniquePassengerCounts' }}])",

    "HAVING AVG": "db.{table}.aggregate([{ '$group': {{ '_id':
    '$passenger_count', 'AvgTrip': {{ '$avg': '$total_amount' }} }} }}, {{
    '$match': {{ 'AvgTrip': {{ '$gt': 5 }} }} }}])",
    "TOMONTH": "db.{table}.aggregate([{ '$group': {{ '_id': {{ '$month':
    '$pickup_datetime' }}, 'TotalRevenueByMonth': {{ '$sum': '$total_amount'
    }} }} }}, {{ '$project': {{ 'Month': '$_id', 'TotalRevenueByMonth': 1,
    '_id': 0 }} }}])",
    "CASE": "db.{table}.aggregate([{ '$group': {{ '_id': {{ '$cond': [{{
    '$gt': ['$trip_distance', 10] }}, 'Long Trip', 'Short Trip'] }},
    'TotalTrips': {{ '$sum': 1 }} }} }}, {{ '$project': {{ 'TripCategory':
    '$_id', 'TotalTrips': 1, '_id': 0 }} }}])",
    "BETWEEN": "db.{table}.find({{ 'total_amount': {{ '$gte': 20, '$lte':
    100 }} }})",
    "UNIQEXACT": "db.{table}.aggregate([{ '$group': {{ '_id':
    '$passenger_count' }} }}, {{ '$count': 'ExactUniquePassengerCounts' }}])"
    ,
    "IN": "db.{table}.aggregate([{ '$project': {{ 'passenger_count': 1,
    'IsCommonPassenger': {{ '$in': ['$passenger_count', [1, 2, 3]] }} }}
    }}])"

}

# Runs the specified query on the given collection and returns the result.
def execute_query(collection, query_str):
    query = eval(query_str.format(table=collection.name))
    return query
```

```python
# Runs the specified query three times and calculates the average time.
def benchmark_query(collection, query_name, query_str):
    times = []
    for _ in range(3):
        start_time = time.time()
        list(execute_query(collection, query_str))  # Sorguyu çalıştır ve
        sonucu listeye al
        elapsed_time = (time.time() - start_time) * 1000  # Convert to
        milliseconds
        times.append(elapsed_time)

    avg_time = sum(times) / len(times)
    return times, avg_time

# Performs text operations to make it look neat. Main method.
def main():
    client = MongoClient(MONGO_URI)
    db = client[DATABASE_NAME]

    print("MONGODB-BENCHMARK")
    print("*" * 40)

    for table in TABLES:
        collection = db[table]
        print(f"Processing Table: {table}")
        print("-" * 40)

        for query_name, query_str in QUERIES.items():
            print(f"{query_name}")
            print("-" * 40)
            times, avg_time = benchmark_query(collection, query_name,
            query_str)
            print(f"Run 1 Elapsed Time: {times[0]:.0f} ms")
            print(f"Run 2 Elapsed Time: {times[1]:.0f} ms")
            print(f"Run 3 Elapsed Time: {times[2]:.0f} ms")
            print(f"Average Time: {avg_time:.0f} ms")
            print("-" * 40)

        print("*" * 40)

    client.close()
    print("All queries completed...")

if __name__ == "__main__":
    main()
```

EK – 4: PostgreSQL Python Betiği

```
# PostgreSQL library and time library are imported.
import time
import psycopg2

# PostgreSQL connection information.
DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "nyc_taxi_data"
DB_USER = "erol"
DB_PASSWORD = "123"

# Table names
TABLES = ["nyc_taxi_data_1m", "nyc_taxi_data_10m", "nyc_taxi_data_50m",
"nyc_taxi_data_100m"]


# Queries
QUERIES = {
    "ALL": "SELECT * FROM {table};",
    "COUNT": "SELECT COUNT(*) AS TotalRecords FROM {table};",
    "AVG": "SELECT AVG(trip_distance) AS AvgTripDistance FROM {table};",
    "SUM": "SELECT SUM(total_amount) AS TotalRevenue FROM {table};",
    "MAX": "SELECT MAX(total_amount) AS MaxTrip FROM {table};",
    "MIN": "SELECT MIN(total_amount) AS MinTotalAmount FROM {table};",
    "ROUND": "SELECT ROUND(AVG(total_amount), 2) AS AvgTotalAmountRounded
    FROM {table};",
    "GROUP BY": "SELECT id, SUM(total_amount) AS TotalRevenueByVendor FROM
    {table} GROUP BY id;",
    "DISTINCT": "SELECT COUNT(DISTINCT passenger_count) AS
    UniquePassengerCounts FROM {table};",
    "HAVING AVG": "SELECT passenger_count, AVG(total_amount) AS AvgTrip FROM
    {table} GROUP BY passenger_count HAVING AVG(total_amount) > 5;",
    "TOMONTH": "SELECT EXTRACT(MONTH FROM pickup_datetime) AS Month,
    SUM(total amount) AS TotalRevenueByMonth FROM {table} GROUP BY Month;",

    "CASE": "SELECT CASE WHEN trip_distance > 10 THEN 'Long Trip' ELSE
    'Short Trip' END AS TripCategory, COUNT(*) AS TotalTrips FROM {table}
    GROUP BY TripCategory;",
    "BETWEEN": "SELECT * FROM {table} WHERE total_amount BETWEEN 20 AND 100;"
    ,
    "UNIQEXACT": "SELECT COUNT(DISTINCT passenger_count) AS
    ExactUniquePassengerCounts FROM {table};",
    "IN": "SELECT passenger_count IN (1, 2, 3) AS IsCommonPassenger FROM
    {table};"
}

# Runs the specified query on the given collection and returns the result.
def execute_query(cursor, query):
    cursor.execute(query)
    return cursor.fetchall()
```

```python
# Runs the specified query three times and calculates the average time.
def benchmark_query(cursor, table, query_name, query_str):
    query = query_str.format(table=table)
    times = []

    for _ in range(3):
        start_time = time.time()
        execute_query(cursor, query)
        elapsed_time = (time.time() - start_time) * 1000  # Convert to
        milliseconds
        times.append(elapsed_time)

    avg_time = sum(times) / len(times)
    return times, avg_time

# Performs text operations to make it look neat. Main method.
def main():
    conn = psycopg2.connect(
        host=DB_HOST,
        port=DB_PORT,
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASSWORD
    )
    cursor = conn.cursor()

    print("POSTGRESQL-BENCHMARK")
    print("*" * 40)

    for table in TABLES:
        print(f"Processing Table: {table}")
        print("-" * 40)

        for query_name, query_str in QUERIES.items():
            print(f"{query_name}")
            print("-" * 40)
            times, avg_time = benchmark_query(cursor, table, query_name,
            query_str)
            print(f"Run 1 Elapsed Time: {times[0]:.0f} ms")
            print(f"Run 2 Elapsed Time: {times[1]:.0f} ms")
            print(f"Run 3 Elapsed Time: {times[2]:.0f} ms")
            print(f"Average Time: {avg_time:.0f} ms")
            print("-" * 40)

        print("*" * 40)


    cursor.close()
    conn.close()
    print("All queries completed...")

if __name__ == "__main__":
    main()
```

TOPLUMSAL KATKI

Günümüzde kamu kurumlarından özel sektöre, sağlık sistemlerinden eğitim ve finansal hizmetlere kadar hemen her alanda verinin hacmi ve karmaşıklığı hızla artmaktadır. Mevcut veri tabanı çözümlerinin performans darboğazları; işlem sürelerinin uzaması, sistem kesintileri, yüksek altyapı maliyetleri ve veri analitiği sonuçlarının geç gelmesi gibi sorunlara yol açmakta; bu da hizmet kalitesinin düşmesine, kaynak israfına ve karar verme süreçlerinde gecikmeye sebep olmaktadır. Özellikle büyük şehirlerde ulaşım, enerji dağıtımı, sağlık hizmetleri gibi kritik alanlarda performans sorunları toplumsal refahı ve güvenliği doğrudan etkileyebilmektedir.

Bu çalışmada önbellek tabanlı (Redis), belge tabanlı (MongoDB), sütun tabanlı (ClickHouse) ve ilişkisel (PostgreSQL) veri tabanı mimarilerinin gerçek dünya iş yükleri altındaki performansları karşılaştırılmış; her koşula uygun optimizasyon stratejileri geliştirilmiştir. Sunduğumuz otomasyon betikleri ve karşılaştırmalı test sonuçları, veri yöneticilerine ve yazılım ekiplerine "hangi veri tabanını, ne zaman ve nasıl yapılandıracaklarını" açıkça göstermektedir. Dolayısıyla bu rehber niteliğindeki bulgular, sistem tasarımcılarının performans darboğazlarını öngörmesini ve önleyici tedbirler almasını sağlamaktadır.

Kamu ve özel sektöre uygulamalarında sorgu sürelerinin kısalması, vatandaş ve müşteri memnuniyetini yükseltir. Daha az donanım ve enerji tüketimi ile hem işletme maliyetleri hem de çevresel etkiler azalır. Kamu BT stratejilerinde "ölçeklenebilirlik" ve "maliyet-etkinlik" kriterlerine göre seçim yapılmasını sağlayacak sağlam performans verileri sunar. Sıklıkla karşımıza çıkan performans sorunlarında yönelik özgün optimizasyon önerileri sektöre yeni best practiceler kazandırır.

Çalışmada geliştirilen otomasyon betikleri ve önerilen konfigürasyon şablonları, veri tabanı yöneticileri tarafından doğrudan devreye alınabilir. Örneğin bir belediyenin trafik yönetim platformunda ClickHouse'un sütun tabanlı mimarisini ve Redis'in bellek içi önbelleğe alma stratejilerini birlikte kullanması, anlık trafik analitiği ve yönlendirmeyi milisaniyeler düzeyine indirebilir. Uzun vadede bu tür

uygulamalar; akıllı şehir çözümlerinin daha yaygın ve güvenilir şekilde benimsenmesine, veri odaklı kamu yönetimi kararlarının hızlanmasına ve toplumsal yaşam kalitesinin yükselmesine katkıda bulunacaktır.

Araştırmanın bulguları, dijital dönüşüm sürecindeki kurumların performans kriterlerini belirli ölçümlerle somutlaştırır. Uzun vadede; Üniversitelerde ve araştırma merkezlerinde veri tabanı optimizasyonu alanında yeni çalışmaların temelini oluşturacak, Daha verimli BT altyapıları işletme maliyetlerini düşürerek yatırımları teşvik edecek, Enerji ve donanım kaynaklarının etkin kullanımına yönelik pratik çözümler sunarak çevresel sürdürülebilirliğe destek verecektir.

Bu yüksek lisans çalışması, veri tabanı performans sorunlarının hem teknik hem toplumsal boyutlarını ele alarak "veri odaklı hizmet kalitesi" kavramını somut ölçümlerle desteklemektedir. Sunulan karşılaştırma verileri ve optimizasyon rehberi, hem yazılım geliştirme dünyasında hem de kamu politikalarında daha etkin karar süreçleri ve stratejiler geliştirilmesine olanak tanıyarak topluma gerçek, ölçülebilir faydalar sağlayacaktır.