T.C.
İNÖNÜ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İLİŞKİSEL VERİ TABANLARINDAN NOSQL DEPOLAMA SİSTEMLERİNE GEÇİŞ İÇİN KULLANILAN YÖNTEMLERİN İNCELENMESİ VE ŞEMA DÖNÜŞÜM UYGULAMASININ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Muhammed Mehdi ELÖMER

Bilgisayar Mühendisliği Ana Bilim Dalı

Tez Danışmanı: Doç. Dr. Ahmet Arif AYDIN

HAZİRAN 2024

T.C.
İNÖNÜ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İLİŞKİSEL VERİ TABANLARINDAN NOSQL DEPOLAMA SİSTEMLERİNE GEÇİŞ İÇİN KULLANILAN YÖNTEMLERİN İNCELENMESİ VE ŞEMA DÖNÜŞÜM UYGULAMASININ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Muhammed Mehdi ELÖMER
(36203619015)

Bilgisayar Mühendisliği Ana Bilim Dalı

Tez Danışmanı: Doç. Dr. Ahmet Arif AYDIN

HAZİRAN 2024

TEŞEKKÜR VE ÖN SÖZ

Bu tez çalışmamın her aşamasında, derin bilgisi ve değerli tecrübeleriyle beni yönlendiren, motive eden ve akademik mükemmeliyet yolunda cesaretlendiren saygıdeğer danışmanım Doç. Dr. Ahmet Arif AYDIN'a,

Hayatımın bu önemli döneminde bana verdikleri eşsiz destek ve sonsuz sevgileriyle her zaman yanımda olan değerli AİLEME,

Bu süreçte, motivasyonumu yüksek tutmamda ve hedeflerime ulaşmamda beni destekleyen, yorulduğum anlarda yanımda olan sevgili DOSTLARIMA

Teşekkür ederim.

ONUR SÖZÜ

Yüksek lisans tezi olarak sunduğum “İlişkisel Veri Tabanlarından NoSQL Depolama Sistemlerine Geçiş İçin Kullanılan Yöntemlerin İncelenmesi ve Şema Dönüşüm Uygulamasının Geliştirilmesi” başlıklı bu çalışmanın bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurmaksızın tarafımdan yazıldığına ve yararlandığım bütün kaynakların hem metin içinde hem de kaynakçada yöntemine uygun biçimde gösterilenlerden oluştuğunu belirtir, bunu onurumla doğrularım.

Muhammed Mehdi ELÖMER

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

SEMBOLLER VE KISALTMALAR

NoSQL : Not Only SQL / Sadece SQL Değil

RDBMS : Relational Database Management System / İlişkisel Veri Tabanı Yönetim Sistemleri

DBMS : Database Management System / Veri Taban Yönetim Sistemleri

SQL : Structured Query Language / Yapılandırılmış Sorgulama Dili

KV : Key-Value / Anahtar-Değer

WC : Wide Column / Geniş Kolon

CQL : Cassandra Query Language / Cassandra Sorgulama Dili

ER : Entity Relationship / Varlık İlişki

JSON : JavaScript Object Notation / JavaScript Nesne Gösterimi

CPU : Central Processing Unit / Merkezi İşlem Birimi

GPU : Graphics Processing Unit / Grafik İşlem Birimi

ÖZET

Yüksek Lisans Tezi

İLİŞKİSEL VERİ TABANLARINDAN NOSQL DEPOLAMA SİSTEMLERİNE GEÇİŞ İÇİN KULLANILAN YÖNTEMLERİN İNCELENMESİ VE ŞEMA DÖNÜŞÜM UYGULAMASININ GELİŞTİRİLMESİ

MUHAMMED MEHDİ ELÖMER

İnönü Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

47 + IV sayfa

2024

Danışman: Doç. Dr. Ahmet Arif AYDIN

Veri depolama sistemlerinin önemi günümüz büyük veri çağında önemli hale gelmiştir. İlişkisel veri tabanlarının katı olan ve esnek olmayan şema yapısı, NoSQL (Not Only SQL) veri depolama sistemlerinin ortaya çıkmasına zemin hazırlamıştır. İlişkisel veri tabanı yönetim sistemlerinin (RDBMS) katı şema yapısından NoSQL veri depolama sistemlerine dönüşüm, veri bütünlüğünü ve tutarlılığını sağlamak, uygun NoSQL veri depolama sisteminin seçimini yapmak ve geçiş sürecini doğru bir şekilde yürütmek için dikkatli bir değerlendirme gerektiren karmaşık bir işlemdir. Bu tez çalışmasında, ilişkisel veri tabanlarından NoSQL'e geçiş önemli bir konu olduğundan, ilişkisel veri tabanlarından istenilen kategorideki NoSQL'e geçiş imkânı sağlayan ve veriyi de yeni oluşturulan şemaya transfer edebilen RL2NoSQL metodu geliştirilmiştir. RL2NoSQL metodu, veri bütünlüğünü ve tutarlılığını korurken, veri modelleri arasındaki yapısal farklılıkları detaylıca ele almakta ve otomatik dönüşüm mekanizmaları sunmaktadır. Ayrıca, bu çözüm, karmaşıklığı yönetme ve veri depolama esnekliğini artırma konusunda önemli avantajlar sağlamaktadır. Bu sayede, veri kaybı veya veri bozulması gibi sorunlar yaşanmadan veri tabanı dönüşüm işlemi gerçekleştirilebilmektedir. Ek olarak, RL2NoSQL yaklaşımı tarafından üretilen şema önerilerini değerlendirmek için birkaç sorgu oluşturulmuştur. Oluşturulan sorgular her bir NoSQL veri depolama sisteminde çalıştırılmıştır. Elde edilen sonuçlar ve bulgular, veri sorgulama operasyonları için potansiyel iyileştirmeleri göstermiştir. RL2NoSQL yöntemi, NoSQL in her bir kategorisinde bulunan veri depolama sistemlerine geçiş imkânı sağladığından sektörde bulunan ve NoSQL'e geçiş yapmak isteyen firma ve organizasyonlara kritik bir katkı sağlamayı amaçlamakta ve çeşitli sektörlerdeki uygulamalar için yeni kapılar açmayı hedeflemektedir.

Anahtar Kelimeler: Veri Modelleme, İlişkisel Veri Tabanı Yönetim Sistemleri, NoSQL, Şema Dönüşümü

ABSTRACT

Master Thesis

EXAMINING THE METHODS USED FOR THE TRANSITION FROM RELATIONAL DATABASES TO NOSQL STORAGE SYSTEMS AND DEVELOPING THE SCHEMA TRANSFORMATION APPLICATION

MUHAMMED MEHDİ ELÖMER

Inonu University
Graduate School of Nature and Applied Sciences
Department of Computer Engineering

47 + IV pages

2024

Supervisor: Assoc. Prof. Ahmet Arif AYDIN

The rigid and inflexible schema structure of relational databases has paved the way for the emergence of NoSQL (Not Only SQL) data storage systems. The transition from Relational Database Management Systems (RDBMS) to NoSQL data storage systems is a complex process that requires careful evaluation to maintain data integrity and consistency, select the appropriate NoSQL system, and correctly manage the transition. In this thesis, given the importance of transitioning from relational databases to NoSQL, the RL2NoSQL method has been developed, which enables the transition to the desired category of NoSQL and also transfers the data to the newly created schema. The RL2NoSQL method meticulously addresses the structural differences between data models while providing automatic transformation mechanisms, offering significant advantages in managing complexity and enhancing storage flexibility. This ensures that the database transformation process can be conducted without data loss or corruption. Additionally, several queries have been created to evaluate the schema suggestions generated by the RL2NoSQL approach. These queries have been executed across various NoSQL storage systems. The results and findings indicate potential improvements for data querying operations. The RL2NoSQL method aims to provide a critical contribution to companies and organizations in the industry looking to transition to NoSQL, and it opens new doors for applications across various sectors.

Keywords: Data Modeling, RDBMS, NoSQL, Schema Transformation

1. GİRİŞ

1.1 Tezin Amacı

Dijital çağda, veri miktarı ve karmaşıklığı hızla artmaktadır, bu da veri yönetimi ve veri modellemesini iş dünyası ve teknoloji içinde stratejik konular haline getirmektedir (Aydin, 2023). Etkili veri yönetimi, organizasyonların rekabet gücünü artırırken, karar alma süreçlerinde veriye dayalı yaklaşımların önemini de ortaya çıkarmaktadır. Bu bağlamda, veri yönetim sistemlerinin gelişimi, modern işletmelerin karşılaştığı zorluklara yanıt vermelerini ve fırsatlardan yararlanmalarını sağlayan temel faktörlerden biri olarak belirlenmektedir (Aydin & Anderson, 2020).

Veri tabanı yönetim sistemlerinin (DBMS) gelişimi, bilgi teknolojisi alanında gelişime ve teknolojik ilerlemeye önemli ölçüde katkıda bulunmuştur. Farklı veri işleme ve analiz ihtiyaçlarını karşılamak için verileri depolama, yönetme, güncelleme ve erişme gibi kritik yetenekleri sunan DBMS, iş sektörlerinin neredeyse tümünde ve kişisel yaşamlarda vazgeçilmez hale gelmiştir (Aydin, 2023). DB-ENGINES'a[1] göre, şu anda çeşitli alanlarda 417 veri depolama sistemi aktif olarak kullanılmaktadır.

1970'ten bu yana, çeşitli sektörlerde yaygın olarak kullanılan İlişkisel Veri Tabanı Yönetim Sistemleri (RDBMS), şu anda farklı endüstrilerde 165'ten fazla RDBMS tarafından kullanılmaktadır. RDBMS'nin ilişkisel veri modeli, katı şemalar, yüksek maliyetli dikey ölçeklendirme ve merkezi depolama (Codd, 1970) gibi özellikleri zorunlu kılar. Bu özellikler, yapılandırılmamış ve yarı yapılandırılmış verilerin yönetimi için uygun değildir. Bu nedenle, büyük veri zorluklarını ele almak için esnek şema tasarımı, yatay ölçeklendirme ve verilerin birden fazla düğüm üzerinde çoğaltılmasını sağlayan dağıtılmış depolama özellikleriyle NoSQL veri depoları ortaya çıkmıştır. NoSQL depolama sistemlerinin gelişimi, büyük veri zorluklarıyla başa çıkmak için bu sistemlerin sunduğu yetenekler nedeniyle elzemdir. DB-ENGINES, büyük veri karmaşıklıklarını yönetmek için şu anda 182 NoSQL depolama sisteminin kullanıldığını bildirmektedir.

Bu tez çalışması, geleneksel İlişkisel veri tabanı yönetim sistemleri (RDBMS) tabanlı veri tabanlarından farklı NoSQL depolama sistemlerine geçişi kolaylaştıran ve şema ile veri

[1] https://db-engines.com/en/

transferini gerçekleştiren yenilikçi bir yöntem olan RL2NoSQL metodu geliştirmiştir. Bu method yardımıyla, veri modellemesi ve yönetimi alanında mevcut zorluklara çözümler sunarken, organizasyonların değişen veri gereksinimlerine uyum sağlamalarına yardımcı olmak amaçlanmıştır. RL2NoSQL yöntemi, PostgreSQL'den Anahtar-Değer, Belge, Geniş Kolon ve Çizge NoSQL depolama sistemlerine şema ve veri transferini sağlar. Bu tez çalışmasının odak noktası, veri depolama sistemlerinin dönüşüm sürecinde veri bütünlüğünün korunması ve veri erişim hızının optimize edilmesidir. Ayrıca bu tez çalışması, veri depolama sistemlerinin gelişimini ve bu süreçteki teknik zorlukları ele almakta, RDBMS'den NoSQL sistemlerine geçiş sırasında karşılaşılabilecek sorunlara pratik çözümler sunmaktadır. Bu çalışma, veri yönetimi ve modelleme alanlarında hem akademik hem de uygulamalı bakış açılarından önemli bir katkı sağlamayı hedeflemektedir.

1.2 Tezin Yapısı

Bu çalışmanın devamında, ikinci bölümde, büyük veri veri depolama sistemlerine genel bir bakış sunulmakta, bu sistemlerin temel özellikleri ve farklılıkları açıklanmaktadır. Üçüncü bölümde, gerçekleştirilen literatür taraması sunulmaktadır ve tez konusuyla ilgili mevcut çalışmalar detaylı bir şekilde incelenmektedir. Dördüncü bölümde, RL2NoSQL metodunun uygulanması ve RL2NoSQL metodunun detayları ele alınmaktadır. Beşinci bölümde, RL2NoSQL metodunun detaylı bir değerlendirilmesi sunulmaktadır, bu bölümde metodun etkinliği ve uygulama sonuçları analiz edilmektedir. Altıncı bölümde, RL2NoSQL metodunun aşamaları detaylandırılmakta ve uygulamada kullanılan kod örneklerine yer verilmektedir. Yedinci bölümde, elde edilen bulgular ve öneriler ayrıntılı bir şekilde tartışılmakta, sonuçların sektörel ve akademik etkileri üzerinde durulmaktadır. Sekizinci ve son bölümde ise genel sonuçlar sunulmaktadır.

2. BÜYÜK VERİ VE VERİ DEPOLAMA SİSTEMLERİNE GENEL BAKIŞ

Bu bölümde veri tabanı yönetim sistemleri ile ilgili önemli olan kavramlara yer verilmiştir. Ayrıca RDBMS ve NoSQL depolama teknolojilerinin genel özelliklerinden bahsedilmiştir. Veri tabanı yönetim sistemleri, bilgi teknolojileri alanındaki önemli bir gelişimin ve teknolojik gelişmelerin odak noktalarından biridir. Bu sistemler, günümüz büyük veri çağında veri depolama, yönetim, güncelleme ve sorgulama gibi önemli işlevleri sağlayarak iş dünyasında ve bireysel yaşamda büyük bir rol oynamaktadır (Aydin, 2023). DBMS'lerin tarihsel gelişimi, veri teknolojilerindeki yenilikler, kullanıcı ihtiyaçları, depolama ihtiyaçlarındaki artış, veri çeşitliliği, bulut bilişimi ve gerçek zamanlı işleme gibi faktörlerin bir yansıması olarak, 1960'lardan günümüze kadar sürekli bir gelişim süreci içindedir. DBMS alanında çeşitli veri modelleri geliştirilmiştir; bunlar arasında hiyerarşik, ağ, ilişkisel, nesne yönelimli, nesne-ilişkisel, NoSQL (Anahtar-Değer, Belge, Geniş Kolon ve Çizge), NewSQL, dizi ve çoklu model bulunmaktadır. Alt bölümlerde sırasıyla ilişkisel veri taban yönetim sistemleri ve NoSQL veri depolama teknolojilerinin genel karakteristikleri açıklanmıştır.

2.1 İlişkisel Veri Tabanı Yönetim Sistemleri

1960-1970 yılları arasında geliştirilen network ve hiyerarşik veri tabanların, yönetilmesi verinin eklenmesinin ortaya çıkardığı verinin yönetimi ile ilgili zorluklardan dolayı, zaman içinde ilişkisel veri tabanı yönetim sistemlerinin ortaya çıkmasına yol açmıştır. 1970'lerin başlarında Edgar F. Codd'un ilişkisel veri modeli kavramı ile ortaya çıkan RDBMS (Codd, 1970), Oracle, IBM DB2, Microsoft SQL Server gibi ticari ürünlerle ve MySQL, PostgreSQL, SQLite gibi açık kaynaklı alternatiflerle yaygınlaşmıştır. RDBMS, verileri tablolar, kolonlar ve satırlar şeklinde düzenleyen, böylece verileri yapılandırılmış ve sorgulanabilir hale getiren bir model benimsemiştir. SQL (Structured Query Language)[2], bu ilişkisel yapının etkili bir şekilde kullanılmasını sağlayan standart bir sorgulama dilidir. RDBMS işlem bütünlüğünü ve güvenilirliğini sağlamak için ACID (Atomicity, Consistency, Isolation, Durability) ilkelerini benimsemiştir. Atomicity (Bölünmezlik) veri tabanında yapalına bir işlemin (hareketin) tamamen bitmesini garanti eder eğer işlemlerde bir hata oluşursa yapılan değişiklikler geri alınır, Consistency (Tutarlılık) veri tabanı işlemlerinin

[2] https://www.ibm.com/docs/en/informix-servers/12.10?topic=sql-programming

tutarlı bir durumda olmasını sağlar, İsolation (İzolasyon) birden fazla işlemin birbirlerini etkilemeden yürütülmesini temin eder ve Durability (Dayanıklılık) ise sistemde herhangi bir hata meydana geldiğinde verilerin kaybolmamasını ve tekrar kullanılabilir hale getirilmesini garanti eder (Doguc & Aydın, 2019). Veri tabanı yönetim sistemleri, finans, sağlık, eğitim ve hükümet gibi çeşitli sektörlerde kritik veri yönetimi ihtiyaçlarını karşılamak üzere geniş çapta benimsenmiştir. Özellikle, veri bütünlüğünü ve işlem güvenliğini sağlama kapasiteleri, RDBMS'leri veri tabanlı uygulamalar için vazgeçilmez kılmıştır. DB-ENGINES[3], şu anda farklı sektörlerde 166 ilişkisel veri tabanı yönetim sisteminin kullanımda olduğunu bildirmektedir. Ancak, RDBMS'lerin sıkı şema tabanlı yapısı ve ölçeklenebilirlik sınırlamaları, hızla büyüyen veri hacimleri ve çeşitliliği karşısında zorluklar oluşturmaya başlamıştır. Bu durum, özellikle yapılandırılmamış ve yarı yapılandırılmış verilerin işlenmesinde, yeni veri tabanı yönetim sistemlerinin araştırılmasını zorunlu kılmıştır (Aydin, 2023).

2.2 Büyük Veri ve Karakteristikleri

Büyük veri, günümüzde birçok sektörde karar alma süreçlerini etkileyen, stratejik bir varlık olarak kabul edilmektedir. Bu kapsamlı veri kümeleri, işletmelerin müşteri davranışlarını daha iyi anlamalarını, operasyonel verimliliklerini artırmalarını ve pazar trendlerini öngörmelerini sağlayarak rekabet avantajı sunar. Büyük veri, hacim (Volume), hız (Velocity), çeşitlilik (Variety), doğruluk (Veracity) ve değer (Value) olmak üzere beş temel karakteristiğe sahiptir. Bu "beş V" kavramı, büyük verinin anlaşılmasını ve yönetilmesini kolaylaştırmak için tasarlanmıştır. Şekil 2.1'de büyük verinin karakteristikleri bir görsel olarak sunulmuştur.

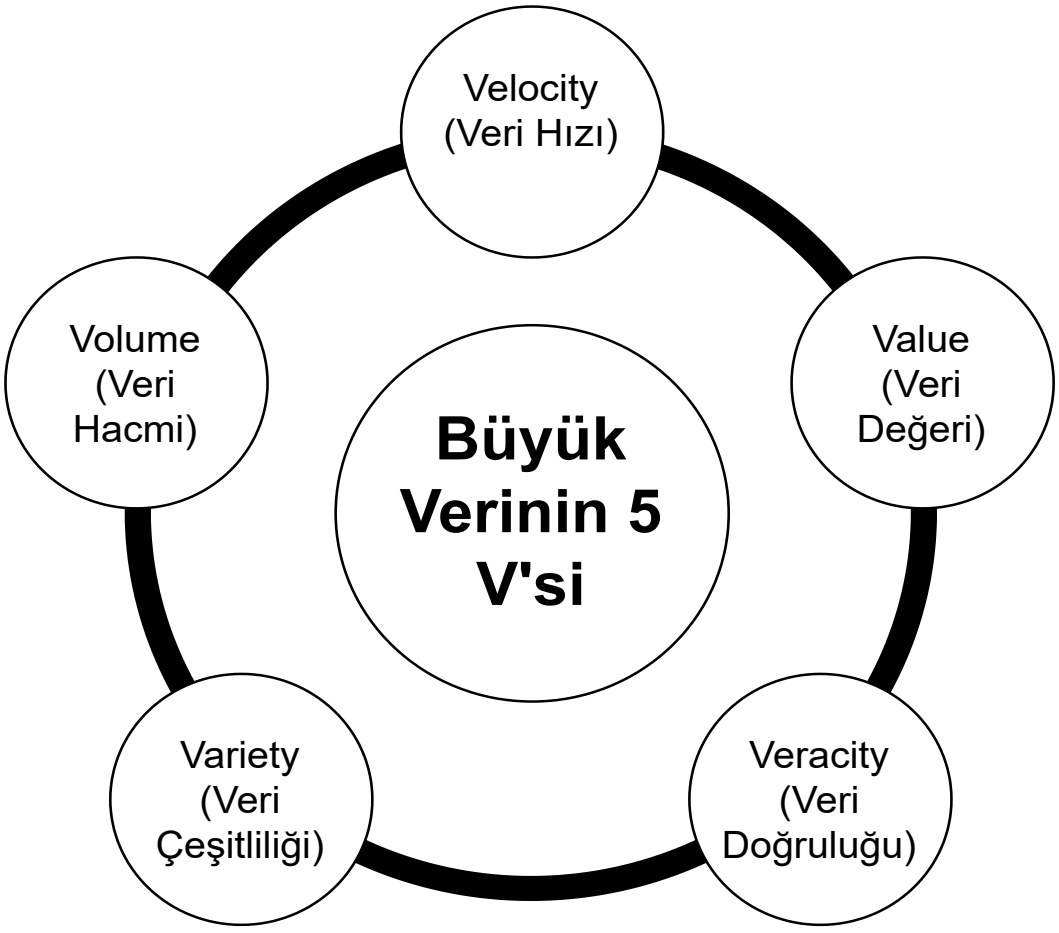
Hacim, büyük verinin en belirgin özelliğidir ve verinin boyutunu ifade eder. Günümüzde üretilen veri miktarı, geçmişteki toplam veri miktarını hızla aşmaktadır. Bu verilerin saklanması, işlenmesi ve analiz edilmesi, geleneksel veri tabanı yönetim sistemleri için büyük bir meydan okuma teşkil etmektedir. Hız, verinin üretim hızını ve işleme hızını belirtir. Gerçek zamanlı veri işleme ve anında karar verme kapasitesi, özellikle finans ve online hizmetler gibi hız odaklı sektörler için kritik öneme sahiptir. Çeşitlilik, büyük verinin farklı biçimlerde gelmesiyle ilişkilidir; yapılandırılmış, yarı yapılandırılmış ve

[3] https://db-engines.com/en/

yapılandırılmamış verileri içerir. Verinin çeşitliliği, analiz süreçlerini karmaşıklaştırsada, doğru araçlarla işlendiğinde daha derin içgörüler sunar.

Doğruluk, veri setlerinin kalitesini ve güvenilirliğini ifade eder. Doğru olmayan veya eksik veriler, yanlış sonuçlar doğurabilir ve işletmeler için risk oluşturabilir. Değer, büyük veriden elde edilebilecek faydayı belirtir. Verinin işlenmesi ve analiz edilmesiyle elde edilen bilgiler, işletmelerin stratejik kararlar almasına yardımcı olur ve değer oluşturur.

Büyük verinin bu beş temel özelliği, işletmelerin ve organizasyonların veri yönetimi stratejilerini şekillendirir. Etkili bir büyük veri yönetimi, bu özelliklere uygun araçlar ve yöntemler gerektirir. Bu kapsamda, NoSQL veri tabanları gibi esnek şema tasarımı, yatay ölçeklenebilirlik ve dağıtılmış mimariler sunan teknolojiler, büyük veri yönetimi için vazgeçilmez hale gelmiştir. NoSQL sistemleri, büyük veri setlerini etkin bir şekilde yönetmek için gerekli performansı ve esnekliği sağlayarak, işletmelerin büyük veriden maksimum fayda sağlamasına olanak tanır.



Şekil 2.1 : Büyük verinin beş temel karakteristiği.

2.3 CAP Teoremi

Dağıtık sistemlerde veri yönetimi, son yıllarda artan veri hacimleri ve işlem ihtiyaçları nedeniyle giderek daha karmaşık hale gelmiştir. Bu karmaşıklığı yönetmek için 2000 yılında Eric Brewer tarafından öne sürülen CAP teoremi, veri tabanı tasarımında önemli bir kılavuz olmuştur. CAP, Tutarlılık (Consistency), Erişilebilirlik (Availability) ve Bölümleme Toleransı (Partition Tolerance) olmak üzere üç temel özellikten oluşur ve bu teoreme göre, bir dağıtık sistem en fazla bu özelliklerin iki tanesini aynı anda sağlayabilir (Doguc & Aydın, 2019). Şekil 2.2'de CAP teoreminin özellikleri sunulmuştur.

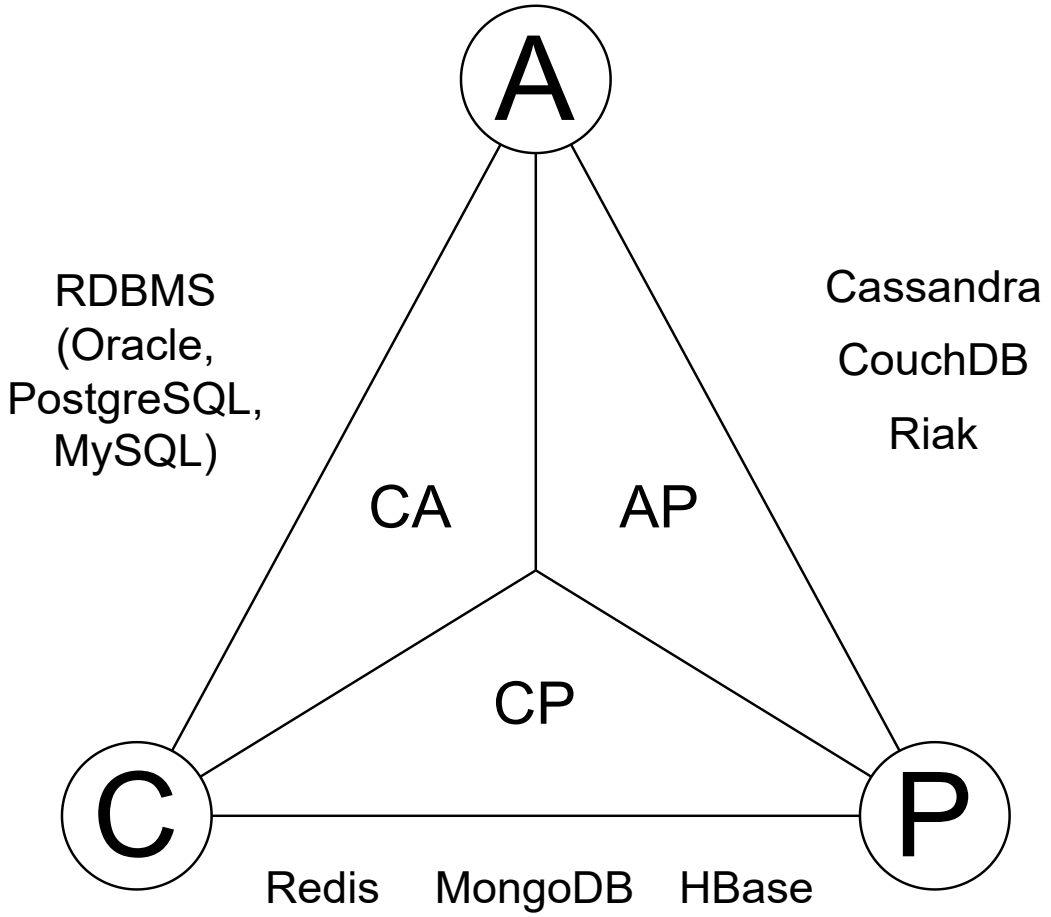
Tutarlılık, veri tabanının tüm düğümlerinde verinin aynı anda güncel ve tutarlı olmasını gerektirir. Yani, bir düğümde yapılan bir güncelleme, tüm düğümlerde neredeyse anında yansıtılmalıdır. İlişkisel veri tabanları genellikle bu özelliği garanti ederken, NoSQL veri tabanları genellikle nihai tutarlılık (eventual consistency) sunar. Nihai tutarlılık, tüm düğümlerin belirli bir zaman dilimi sonunda tutarlı hale geleceğini, ancak işlemler sırasında verinin farklı düğümlerde farklı olabileceğini kabul eder.

Erişilebilirlik, her türlü ağ bölümlemesinde sistemin kullanıcı isteklerine yanıt verme kabiliyetini ifade eder. Erişilebilirlik odaklı sistemler, özellikle web hizmetleri ve e-ticaret gibi kesintisiz hizmet gerektiren uygulamalar için tercih edilir. Örneğin, bir alışveriş sitesi, veri tabanı düğümlerinden biri çöktüğünde bile kullanıcıların alışveriş yapabilmesini sağlamak için yüksek erişilebilirlik gerektirir.

Bölümleme toleransı, sistem bileşenlerinin bir kısmı çöktüğünde bile sistemin çalışmaya devam edebilmesini sağlar. Özellikle büyük ölçekli dağıtık sistemlerde, ağ hataları kaçınılmaz olduğundan, sistemlerin bu tür kesintilere dayanıklı olması gerekir.

İlişkisel veri tabanları, CAP teoreminde tutarlılık (Consistency) ve erişilebilirlik (Availability) özelliklerini önceliklendirirken, bölüm toleransını (Partition Tolerance) daha az öncelikli olarak ele alır. Özellikle, ACID özellikleriyle desteklenen tutarlılık, tüm düğümlerde veri tabanı değişikliklerinin sürekli ve eş zamanlı olarak yansıtılmasını garanti eder. Ancak, ağ bölünmeleri gibi durumlarda ilişkisel veri tabanları, veri tutarlılığını korumak adına operasyonları durdurabilir. Bu, özellikle yüksek işlem bütünlüğü gerektiren karmaşık işlemleri destekleyen Microsoft SQL Server veya Oracle Database gibi sistemlerde görülebilir. Bu sistemler, yüksek derecede tutarlılık ve veri bütünlüğü sunar, ancak bu bazen erişilebilirliğin feda edilmesi anlamına gelebilir. Bu tür ilişkisel veri tabanları genellikle

kritik iş uygulamaları için tercih edilir çünkü veri doğruluğu ve güvenilirliği bu veri tabanı bağlamında üstün bir öncelik taşır.



Şekil 2.2 : CAP teoremine genel bakış.

NoSQL veri tabanları, genellikle bölümleme toleransı ve erişilebilirliği ön plana çıkarırken, tutarlılığı daha az öncelikli olarak ele alır. Örneğin, Amazon'un DynamoDB'si ve Apache Cassandra gibi sistemler, yüksek erişilebilirlik ve bölüm toleransını destekleyerek büyük ölçekli uygulamalar için idealdir. Bu sistemler, veri kaybı olmaksızın ölçeklenebilirlik ve sürekli erişilebilirlik sunar.

CAP teoremi, sistem tasarımcılarına, özellikle de NoSQL veri tabanları kullanılarak gerçekleştirilen uygulamalar için, hangi özelliklerin öncelikli olduğunu seçme konusunda yardımcı olur. Bu kararlar, uygulamanın doğası ve gereksinimleri göz önünde bulundurularak yapılmalıdır. Örneğin, bir finans uygulaması için tutarlılık hayati öneme

sahipken, bir içerik dağıtım ağı (CDN) için erişilebilirlik ve bölüm toleransı daha önemli olabilir.

CAP teoremi, veri tabanı tasarımında ve dağıtık sistemlerin yönetiminde kritik bir rol oynar. Tutarlılık, erişilebilirlik ve bölüm toleransı arasındaki denge, uygulamanın başarısını doğrudan etkileyebilir. NoSQL çözümleri, bu üç özelliği dengeli bir şekilde sunarak, modern uygulamaların ihtiyaçlarına uyum sağlamada önemli bir rol oynamaktadır.

2.4 NoSQL Paradigması

NoSQL (Not Only SQL) veri depolama sistemleri, RDBMS veri tabanlarının ACID prensipleriyle beraber ortaya çıkan kısıtlamaları ve oluşturduğu sınırlamaları ortadan kaldırmak için yanıt olarak ortaya çıkmıştır. Yapılandırılmamış ve yarı yapılandırılmış verilerle çalışabilme, yüksek düzeyde ölçeklenebilirlik ve şema esnekliği gibi özellikleriyle, NoSQL sistemleri, büyük veri ve web ölçekli uygulamaların taleplerini karşılamak üzere geleneksel ilişkisel veri tabanlarından farklı bir yaklaşım sunarak tasarlanmıştır. Veri bütünlüğünü ve tutarlılığını sağlamak için ACID'den farklı olarak BASE (Basically Available, Soft state, Eventually consistent) modelini benimsemektedirler. "Basically Available" sistemin her zaman erişilebilir olmasını ifade eder, "Soft state" sistemin sürekli değişim halinde olabileceğini kabul eder ve bu değişikliklerin zamanla stabil hale geleceğini belirtir, "Eventually consistent" ise, belirli bir zaman geçtikten sonra, verinin tutarlı bir duruma ulaşacağını garantiler (Doguc & Aydın, 2019). NoSQL depolama sistemleri, dört ana kategoride sınıflandırılmaktadır: Anahtar-Değer (Key-Value), Belge tabanlı (Document), Geniş kolon (Wide-Column) ve Çizge (Graph) depolama sistemleri (Ali et al., 2019). Her bir kategori, farklı veri modellerine ve kullanım senaryolarına odaklanarak çeşitli avantajlar sunar (Aydin, 2023).

Belge tabanlı depolama sistemleri, veriyi belgeler halinde depolayan ve genellikle JSON veya BSON formatlarını kullanan sistemlerdir. Her bir belge, semantik olarak ilişkilendirilmiş anahtar-değer çiftlerini içerir. Bu yapı, çok çeşitli ve karmaşık veri tiplerini desteklemelerine olanak tanır. Büyük veri setleri ve değişken veri yapılarıyla çalışmak için idealdir. Örneğin, MongoDB, CouchDB, Couchbase, Databricks ve Realm popüler Belge tabanlı NoSQL depolama sistemleridir (Ali et al., 2019) (Aydin, 2023).

Anahtar-Değer tabanlı veri depolama sistemleri, basit bir anahtar-değer çifti ile ilişkilendirilmiş sistemlerdir. Bu yapı, basit ve hızlı veri erişimi gerektiren uygulamalar için

uygundur. Bu kategoride olan NoSQL, sistemin hafızasını etkin biçimde kullanma imkânı sundugundan çok hızlı sorgulama işlemleri sağlamaktadır (KEKEVİ & AYDIN, 2022). Redis, Memcached, Riak, Hazelcast ve Ignite gibi Anahtar-Değer depolama sistemleri, bu kategorinin popüler örnekleridir (Aydin, 2023).

Geniş Kolon tabanlı depolama sistemleri, yüksek ölçeklenebilirlik ve büyük veri setlerinin etkin yönetimi için tasarlanmıştır. Bu kategoride sınıflandırılan veri depolama sistemleri veriyi kolonlar halinde depolama imkânı sağlar. Her bir kolon, birleşik bir anahtar ile ilişkilendirilmiş bir veri ailesini temsil eder. Bu yapı, geniş ve dağıtık veri setlerini etkili bir şekilde depolamak ve sorgulamak için uygundur. Apache Cassandra, Apache HBase, Accumulo ve ScyllaDB geniş kolon depolama sistemlerinin popüler örnekleridir (Aydin, 2023).

Çizge tabanlı NoSQL depolama sistemleri, düğümler ve bu düğümleri birbirine bağlayan kenarlar aracılığıyla veriyi depolayan sistemlerdir. Bu yapı, sosyal ağ analizleri, öneri sistemleri ve ağ topolojileri gibi alanlar için idealdir. Sosyal ağ analitiği veya ağ haritalama gibi uygulamalar için kullanılır. Örneğin, Neo4j, OrientDB, Virtuoso, ArangoDB ve JanusGraph popüler olan bazı çizge tabanlı NoSQL teknolojilerindendir (Aydin, 2023).

NoSQL depolama sistemlerinin bu çeşitliliği, modern veri yönetimi gereksinimlerinin karşılanmasında önemli bir rol oynamaktadır. Ancak, bu sistemlerin de kendi içlerinde avantajları ve sınırlamaları bulunmaktadır. Örneğin, bazı NoSQL sistemleri, veri bütünlüğü ve tutarlılığı konusunda RDBMS veri tabanları kadar katı olmayabilir. Bu nedenle, uygulama gereksinimlerine ve veri özelliklerine uygun doğru veri tabanı seçimi, projenin başarısı için kritik öneme sahiptir.

3. LİTERATÜR ARAŞTIRMASI

Bu bölümde tez çalışmasıyla birinci dereceden ilişkili olan ve şema dönüşümü gerçekleştirilen yayınlara yer verilmiştir. Ayrıca bu çalışmaların özeti Çizelge 3.1'de sunulmuştur.

Averbuch ve arkadaşları tarafından yayımlanan makalede (Averbuch et al., 2013), ilişkisel veri tabanlarını çizge veri tabanlarına dönüştürmek için bir yöntem sunulmaktadır. Bu süreçte, PostgreSQL gibi bir ilişkisel veri tabanı sisteminden başlayarak, Neo4J ve OrientDB gibi çizge veri tabanı yönetim sistemlerine veri aktarımı gerçekleştirilmektedir. Dönüşüm, ilişkisel veri tabanının şemasını ve verilerini doğrudan sorgulayarak, veri tabanı şemasını bir çizge şemasına ve verileri de çizge veri tabanındaki düğümler ve kenarlar olarak dönüştüren bir Java sistemi kullanılarak yapılmaktadır. Yapılan deneylerde, dönüştürme işlemi sonrası sorgu performansının değerlendirilmesi, çizge veri tabanlarında ilişkisel veri tabanlarına kıyasla bazı durumlarda daha iyi performans sergilediğini göstermektedir. Bu dönüşüm, özellikle ilişkisel veri tabanlarında karmaşık "join" işlemleri gerektiren sorguların çizge veri tabanlarında daha verimli bir şekilde gerçekleştirilebilmesini sağlamaktadır.

Singh ve Kaur makalelerinde (M. Singh & Kaur, 2015), sağlık hizmetleri veri tabanlarını ilişkisel veri tabanlarından çizge veri tabanlarına dönüştüren SQL2Neo yöntemini tanıtmaktadırlar. Yöntem, ilişkisel veri tabanı şemalarını ve kısıtlamalarını kullanarak, MySQL gibi popüler bir ilişkisel veri tabanından, Neo4j gibi yaygın bir çizge veri tabanına veri aktarımını gerçekleştirir. Dönüşüm süreci, ilişkisel veri tabanındaki tabloları ve dış anahtar ilişkilerini düğümlere ve kenarlara dönüştürerek, sağlık hizmetleri veri tabanı örneği üzerinden somutlaştırılmıştır. Deneysel sonuçlar, çizge veri tabanlarında sorgu performansının iyileştiğini ve veri ilişkilerinin daha etkin yönetildiğini göstermektedir. Bu çalışma, özellikle ilişki yoğun veri setlerinin yönetiminde çizge veri tabanlarının önemini vurgulamaktadır.

Çizelge 3.1 : İlişkisel veri tabanlarından NoSQL e geçiş yapan yayınlar ve yöntemleri.

Referans	Yöntem & Programlama Dili	Kaynak Veri Tabanı	Hedef NoSQL Depolama Sistemi			
			Belge Tabanlı	WC	KV	Çizge
(Averbuch et al., 2013)	Özel & Java	PostgreSQL	-	-	-	Neo4j, Orient DB
(M. Singh & Kaur, 2015)	SQL2Neo & Java	MySQL	-	-	-	Neo4j
(Gopalan et al., 2017)	Özel & Python	MySQL	-	Cassandra	-	-
(Hamouda & Zainol, 2017)	DODS & Belirtilmemiş	RDBMS	MongoDB	-	-	-
(Unal & Oguztuzun, 2018)	Özel & Java	MySQL	-	-	-	Neo4j
(Mahmood, 2018)	Özel & Belirtilmemiş	MS SQL	MongoDB	-	-	-
(Nan & Bai, 2019)	Özel & Java	ER	-	-	-	Neo4j
(Alotaibi & Pardede, 2019)	Özel & Belirtilmemiş	Oracle	MongoDB	Cassandra	-	Neo4j
(A. Singh, 2019)	Özel & Belirtilmemiş	RDBMS	MongoDB	-	-	-
(Al Mahruqi et al., 2019)	Özel & Java	MySQL	MongoDB	-	-	-
(Chen & Lee, 2020)	Özel & Belirtilmemiş	RDBMS	-	Cassandra, HBase	-	-
(Namdeo & Suman, 2021b)	SDAM & Java	RDBMS	MongoDB	-	-	-

(Namdeo & Suman, 2021a)	SLSDMM & Java	MySQL	MongoDB	-	-	-
(Belkadi & Esbai, 2021)	Özel & Belirtilmemiş	RDBMS	MongoDB	-	-	-
(Feng & Huang, 2022)	Özel & Python	MySQL	-	-	-	Neo4j
(Zaidi et al., 2022)	Özel & Java	MySQL	-	HBase	-	-
(Abdelhedi et al., 2022)	RDBToNoSDW & Java	MySQL, PostgreSQL	-	-	-	Orient DB
(Revathi et al., 2023)	Auto JSON & Python	MySQL	MongoDB	-	-	-
(Erraji et al., 2023)	TMSDRDND & JavaScript	MySQL	MongoDB	-	-	-
(Tandya & Azizah, 2023)	Özel & Belirtilmemiş	MySQL	MongoDB	-	-	-
(Hamouda, 2023)	Özel & Belirtilmemiş	MySQL	MongoDB	-	-	-
(Eldrrat & Maatuk, 2023)	Özel & Python, JAVA,	RDBMS	-	-	-	Neo4j
Bu tez çalışması	***RL2NoSQL & Python***	***PostgreSQL***	***MongoDB***	***Cassandra***	***Redis***	***Neo4j***

Gopalan ve arkadaşları tarafından yayımlanan makale (Gopalan et al., 2017), MySQL gibi ilişkisel veri tabanlarından Cassandra gibi NoSQL veri tabanlarına veri göçü sürecini ele almaktadır. Yazarlar, bu göçün iki aşamadan oluştuğunu belirtmektedir: şema ve veri göçü. Makalede, Python programlama dili kullanılarak MySQL şemalarının ve verilerinin Cassandra Query Language (CQL)[4] formatına otomatik olarak nasıl dönüştürüleceği

[4] https://cassandra.apache.org/doc/stable/cassandra/cql/

açıklanmaktadır. Bu sürecin, veri bütünlüğünü korurken veri erişim hızını artırabileceği ve ölçeklenebilirliği iyileştirebileceği vurgulanmaktadır.

Hamouda ve Zainol makalelerinde (Hamouda & Zainol, 2017), ilişkisel veri tabanlarından belge yönelimli NoSQL veri tabanlarına karmaşık veri tabanları ile ilgili göç sorunlarını ele alan bir Document-Oriented Data Schema (DODS) önermektedirler. Önerilen yöntem, ER modelinin özelliklerini temel alarak, ilişkisel veri tabanı özelliklerini, tabloları, ilişkileri, işlevleri ve kısıtlamaları dikkate alarak veri normalleştirme ve denormalleştirme süreçlerini içerir. Bu çalışma, karmaşık veri tabanlarının göçü için uyumlu bir yöntem sunmayı amaçlamaktadır.

Unal ve Oguztuzun makalelerinde (Unal & Oguztuzun, 2018), MySQL'de bulunan bir veri tabanının, belirli adımları izleyerek (önce şemayı SchemaCrawler aracılığı ile aktarıp, daha sonra verileri Java ile yazılan özel bir yazılım kullanarak aktarmak) Neo4j çizge depolama sistemine aktarılması ve ardından yapılan sorgulama testleri sonucunda, Yazarlara göre Neo4j'nin belli sorgularda (özellikle ilişkisel veri tabanlarında "join" gerektiren sorgularda) daha performanslı olduğu gözlemlenmiştir. Bu deneyimler ışığında, many-to-many ilişkisi bulunan, sık şema değişikliği gerektiren veya yapılandırılmamış verilerin depolandığı durumlarda çizge veri tabanlarının tercih edilmesinin önemi vurgulanmaktadır.

Mahmood makalesinde (Mahmood, 2018), ilişkisel veri tabanlarından NoSQL depolama sistemlerine geçiş yapmak için otomatize bir algoritma önermektedir. Algoritma, veri tabanı şemalarını ve tablolar arası ilişkileri analiz ederek, dönüşüm için Gömülü, Bağlantılı ve Otomatik olmak üzere üç mod sunmaktadır. Bu modlar, NoSQL depolama sisteminde veri yapılandırmasını etkinleştirmekte ve veri transfer sürecini kolaylaştırmaktadır. Geliştirilen VB.NET uygulaması aracılığıyla, bir MS SQL Server veri tabanından alınan örnek verilerle MongoDB NoSQL depolama sistemi başarıyla oluşturulmuştur. Sonuçlar, bu yöntemin, veri taşımada güçlü ve esnek bir çözüm sunduğunu göstermektedir.

Nan ve arkadaşları tarafından yayımlanan makale (Nan & Bai, 2019), büyük veri bağlamında, ilişkisel veri tabanlarından çizge veri tabanlarına veri dönüşümüne odaklanmaktadır. İlişkisel veri tabanlarının depolama kapasitesi ve sorgulama verimliliği gibi bazı sorunları ele alarak, NoSQL veri tabanlarına, özellikle çizge veri tabanlarına geçişi önermektedir. Yazarlar, Varlık-İlişki (ER) diyagramlarını temel alarak, çizge modeline dönüştürme kuralları geliştirmişler ve bu kuralları kullanarak veri göçünü otomatikleştiren bir algoritma sunmuşlardır. Makalede, model farklılıklarının etkisini azaltma, veri

bütünlüğünü koruma ve göç işlemini otomatik olarak tamamlama üzerine vurgu yapılmaktadır. Ayrıca, dönüşüm yönteminin geçerliliği ve doğruluğunu gösteren deneysel sonuçlar da sunulmuştur.

Alotaibi ve Pardede makalelerinde (Alotaibi & Pardede, 2019), ilişkisel veri tabanı şemalarını çeşitli NoSQL depolama sistemi şemalarına dönüştürmek için bir dizi kural önermektedirler. Önerilen dönüşüm kuralları, birincil anahtarların birleşiminden oluşan süper kolon ailesi oluşturarak ilişkisel veri tabanındaki çoktan çoğa ilişkileri kolon tabanlı NoSQL'e dönüştürmeyi; ilişkisel veri tabanında bir koleksiyona başka bir koleksiyonun ID'sini referans olarak ekleyerek doküman tabanlı NoSQL'e dönüştürmeyi ve her bir varlık için bir düğüm oluşturarak ve ilişkileri düğümler arasında ilişki özelliği olarak kullanarak çizge tabanlı NoSQL'e dönüştürmeyi içermektedir. Bu dönüşüm süreci, Oracle, Cassandra, MongoDB ve Neo4j olmak üzere dört farklı veri tabanı üzerinde test edilmiş ve altı farklı sorgu çalıştırılarak dönüşüm sonuçları doğrulanmıştır. Önerilen kuralların tamamlılığı, mevcut çalışmalarla karşılaştırılarak analiz edilmiş ve daha kapsamlı ilişki türlerini içerdiği gösterilmiştir.

Singh makalesinde (A. Singh, 2019), ilişkisel veri tabanlarından MongoDB'ye veri göçü sürecini ele almakta ve bu süreçte tablo formundaki verilerin yapılandırılmamış belgelere nasıl dönüştürüldüğünü göstermektedir. Makalede, XAMPP ve MongoDB yönetim sistemleri kullanılarak dönüşümün performansının doğrulandığı belirtilmektedir. Ayrıca, MongoDB'nin özellikle konum tabanlı veriler üzerinde ACID işlemleri kullanarak veri bütünlüğünü koruma konusunda etkili olduğu vurgulanmaktadır. Bu göç sürecinin, bulut tabanlı depolama çözümlerinin maliyet etkinliği ve MongoDB'nin yerleşik sharding çözümüyle yüksek ölçeklenebilirliği sunması gibi avantajları da göz önünde bulundurularak değerlendirilmesi gerektiği belirtilmektedir.

Al Mahruqi ve arkadaşları tarafından yayımlanan makale (Al Mahruqi et al., 2019), web uygulamalarının MySQL gibi ilişkisel veri tabanlarından MongoDB gibi belge yönelimli NoSQL veri tabanlarına yarı otomatik bir şekilde nasıl geçirileceğini ele almaktadır. Yazarlar, bu geçiş sürecinin iki aşamadan oluştuğunu belirtmektedir: şema ve veri göçü ile uygulama kaynak kodunun göçü. Makalede, gömülü SQL sorgularının NoSQL veri tabanı API'si ile etkileşim kuracak şekilde nasıl taşınacağı ve uygulama kodunun bu yeni sorguları kullanacak şekilde nasıl değiştirileceği açıklanmaktadır. Bu yaklaşımın, PHPBB3 gibi büyük ölçekli web uygulamalarının göç edilmesinde başarılı olduğu ve göç edilen sistemin

performansının optimize edilmesiyle orijinal uygulamaya yakın bir performans sergilediği gösterilmektedir.

Chen ve Lee makalelerinde (Chen & Lee, 2020), RDB'den Wide Column Store Database'e (WCSDB) veri ve şema dönüşümü için bir algoritma sunmaktadırlar. Bu algoritma, veri modellerinin dönüştürülmesi ve optimize edilmiş sorgu performansı için verilerin nasıl yeniden yapılandırılacağını açıklamaktadır. Özellikle, ilişkisel tabloları ve ilişkileri WCSDB'nin kolon ailesi yapısına uyacak şekilde nasıl haritalayacağını detaylandırmaktadır. Algoritma, veri modelinin dönüştürülmesi, ilişkisel tabloların kolon ailelerine nasıl bölüneceği ve bu süreçte verimlilik ve performansı nasıl maksimize edeceği üzerine odaklanmaktadır.

Namdeo ve Suman tarafından yazılan makalede (Namdeo & Suman, 2021b), RDBMS'den NoSQL veri tabanlarına geçişi kolaylaştıran bir Schema Design Advisor Modeli (SDAM) sunmaktadır. Bu model, mevcut ilişkisel veri tabanı şemalarını ve sorgularını analiz ederek, bunları NoSQL (özellikle belge tabanlı veri tabanları için) için optimize edilmiş şemalara dönüştürmeyi amaçlamaktadır. Makale, bu dönüşüm sürecinin her adımını ayrıntılı olarak ele almakta ve bir maliyet modeli sunarak farklı şema tasarımlarının etkinliğini değerlendirmektedir. Makalede, bu modelin uygulanabilirliği ve performansı gerçek dünya veri tabanları üzerinde yapılan deneylerle test edilip değerlendirilmektedir.

Namdeo ve Suman tarafından sunulan makalede (Namdeo & Suman, 2021a), ilişkisel veri tabanlarından (RDBMS) NoSQL veri tabanlarına (özellikle MongoDB'ye) veri aktarımı için geliştirilen ve hibrit bir model olan Snapshot-Live Stream Veri Tabanı Migrasyon Modelini (SLSDMM) sunmaktadır. MySQL'den MongoDB'ye gerçek zamanlı ve eski verilerin paralel olarak taşınmasını sağlayan bu model, hem anlık görüntü (snapshot) verilerini hem de canlı veri akışını etkili bir şekilde yönetir.

Belkadi ve Esbai tarafından yayınlanan makalede (Belkadi & Esbai, 2021), büyük veri bağlamında ilişkisel veri tabanlarından (RDBMS) belge yönelimli NoSQL veri tabanlarına (özellikle MongoDB) geçişi otomatize etmek için Model Yönlendirilmiş Mühendislik (Model-Driven Engineering-MDE) yaklaşımını kullanmaktadır. Makale, MOF 2.0 QVT (Meta-Object Facility 2.0 Query-View-Transformation) ve Acceleo dönüşüm dillerini kullanarak, ilişkisel veri tabanı şemalarından NoSQL belge yönelimli veri tabanları için JSON dosyaları üreten dönüşüm kurallarını tanımlamaktadır.

Feng ve Huang tarafından yazılan makalede (Feng & Huang, 2022), ilişkisel veri tabanlarından çizge veri depolama sistemlerine veri göçü sürecini ele alır ve bu süreci üç ana adımda inceler: ER diyagramlarının çizge veri depolama sistemlerinde tasarlanması, ilişkilerin çizge veri depolama sistemlerinde modellenmesi ve verilerin ilişkisel veri tabanından çizge veri depolama sistemlerine taşınması. Makalede sunulan yöntem, ilişkisel veri tabanında bulunan verilerin ER diyagramı aracılığıyla tasarlanıp çizge veri depolama sistemlerine aktarılmasını içerir. Yazarlar, önerdikleri metodolojinin çeşitli ilişkisel veri tabanlarının dönüşümünü başarıyla gerçekleştirdiğini ve verilerin tutarlılığını koruduğunu belirtirler. Ancak, bazı sorguların daha az verimli olduğunu ve yöntemin bazı eksiklikler içerdiğini kabul ederler.

Zaidi ve arkadaşları tarafından yayımlanan makalede (Zaidi et al., 2022), ilişkisel veri tabanlarından kolon tabanlı NoSQL veri tabanlarına veri migrasyonu için bir şema dönüşüm tekniği sunmaktadır. Teknik, denormalizasyon, veri erişim desenleri ve çok katmanlı şemaları birleştirmektedir. MySQL'den MongoDB'ye dönüşümü örnek alarak, bu yöntemin sorgulama süresini ve depolama alanını iyileştirdiğini gösteren deney sonuçları sunulmuştur. Bu çalışma, veri tekrarını azaltarak ve sorgulama performansını artırarak veri migrasyon sürecini optimize etmeyi amaçlamaktadır.

Abdelhedi ve arkadaşları tarafından yayımlanan makalede (Abdelhedi et al., 2022), ilişkisel veri tabanlarından NoSQL depolama sistemlerine veri aktarımı sürecini ele almaktadır. Özellikle, MySQL ve PostgreSQL gibi ilişkisel veri tabanlarından alınan verilerin, OrientDB gibi belge tabanlı bir NoSQL depolama sistemine nasıl dönüştürüleceğini açıklamaktadır. Dönüşüm için Model Driven Architecture (MDA) kullanılarak, ilişkisel veri tabanlarından alınan verilerin NoSQL depolama sistemlerine aktarımı için üç aşamalı bir süreç (CreateDW, ConvertLinks ve MergeClasses modülleri) tanımlanmaktadır.

Revathi ve arkadaşları tarafından yayımlanan makalede (Revathi et al., 2023), MySQL veri tabanlarından MongoDB'ye otomatik veri transfer sağlayan "Auto JSON" adlı bir modeli tanıtmaktadır. Model, MySQL veri tabanı şemalarını ve verilerini JSON belgelerine dönüştürerek, bu verileri MongoDB koleksiyonlarına taşıyan bir süreci kapsar. Makalede, hastane yönetim sistemi örneğinden alınan veri tabanı üzerinden yapılan deneysel testlerle, modelin etkinliği ve performansı değerlendirilmiştir. Bu modelin özellikle tablo boyutu, veri türü ve ilişki yöntemi gibi faktörleri göz önünde bulundurarak dönüşüm stratejisi seçimini optimize ettiği belirtilmiştir. Makalenin sonucunda, bu otomatik dönüşüm yaklaşımının,

ilişkisel veri tabanlarından NoSQL veri tabanlarına geçişte verimlilik ve hız açısından önemli avantajlar sunduğu vurgulanmıştır.

Erraji ve arkadaşları tarafından yayımlanan makalede (Erraji et al., 2023), MySQL'den MongoDB'ye veri transferini ele alan ve bu süreci on adımda gerçekleştiren kapsamlı bir yaklaşımı tanıtmaktadır. Makale, MongoDB'nin JSON tabanlı veri depolama yapısını ve Mongoose JavaScript kütüphanesini kullanarak ilişkisel veri tabanı öğelerinin nasıl dönüştürüleceğini ayrıntılı olarak açıklar. Bu süreç, ilişkisel veri tabanlarındaki tablolar, kolonlar, kısıtlamalar ve ilişkisel bağlar gibi bileşenlerin MongoDB'nin belge tabanlı yapısına nasıl entegre edilebileceğini kapsar. Ayrıca, veri tabanı şemaları, veri bütünlüğü ve sorgulama performansı gibi faktörlere odaklanarak, MongoDB için uygun şema dönüşüm kurallarını geliştirir. Makale, MongoDB'ye geçişi kolaylaştıracak pratik yöntemler ve dönüşüm stratejileri sunmakta ve bu sürecin etkinliğini vurgulamaktadır.

Tandya ve Azizah makalelerinde (Tandya & Azizah, 2023), ilişkisel veri tabanlarından NoSQL belge tabanlı depolama sistemlerine veri dönüşümünü gerçekleştiren bir algoritma geliştirmektedirler. Bu süreçte, belge yönlü veri tabanlarının gömme ve referanslama yetenekleri önemli avantajlar sunmaktadır. Geliştirilen algoritma, MySQL'den MongoDB'ye veri aktarımı için ilişkisel veri tabanının meta verilerini kullanmaktadır. Performans değerlendirmesi, algoritmanın genel olarak diğer algoritmalara göre daha iyi sonuçlar sunduğunu göstermektedir, ancak gömülü belgelere sık erişim gerektiren durumlarda performans düşüşleri yaşanabilmektedir. Yazarlar, algoritmanın daha kapsamlı gerçek dünya veri tabanı senaryolarında test edilmesini önermektedirler.

Hamouda makalesinde (Hamouda, 2023), ilişkisel veri tabanlarından belge tabanlı NoSQL depolama sistemlerine geçişi kolaylaştıran bir algoritma sunmaktadır. Bu algoritma, ilişkisel veri tabanlarının karmaşık yapılarını analiz ederek, veri tiplerini ve ilişkilerini detaylı bir şekilde ele alır. Özellikle, ilişkisel veri tabanlarında yer alan tablo yapıları ve ilişkiler, belge tabanlı NoSQL veri tabanlarına uygun bir formata dönüştürülmektedir. Veri kaybı veya çoğalmasını önleyen bu yöntem, veri tabanı özelliklerinin tamamını etkin bir şekilde belge tabanlı yapıya aktarabilmektedir. Çalışma, NoSQL'e geçiş sırasında karşılaşılan zorlukları gidermekte ve bu süreci sistematik bir şekilde yönlendirmektedir.

Eldrrat ve Maatuk tarafından yazılan makalede (Eldrrat & Maatuk, 2023), ilişkisel veri tabanlarından (RDB) çizge veri tabanlarına veri aktarımı için yeni bir yaklaşım sunmaktadır. Önerilen yöntem, veri tabanı meta verilerini ve ilişkisel yapıları analiz ederek, bu bilgileri

çizge veri tabanı şemasına dönüştürmeye yönelik algoritmalar geliştirmektedir. Bu süreç, veri bütünlüğünü ve tutarlılığını koruyarak karmaşık veri tabanı yapılarını ve ilişkilerini etkili bir şekilde yönetmeyi amaçlamaktadır. Makalede, bu yöntemin etkinliği ve doğruluğu, gerçek veri tabanları üzerinde yapılan deneylerle test edilip değerlendirilmektedir.

RL2NoSQL yaklaşımı, Çizelge 3.1'deki diğer araştırma makalelerinden farklı olarak, NoSQL depolama sistemlerinin dört kategorisine odaklanmaktadır. Kullanıcılar, hedef NoSQL şemasını oluşturmadan ve verileri taşımadan önce, kaynak veri tabanının mevcut özelliklerini, ilişkilerini ve yapısını ara JSON temsillerini kullanarak gözden geçirebilirler. Çalışmamız, Çizelge 3.1'deki diğer çalışmalardan öne çıkmaktadır çünkü çoğu araştırma makalesi yalnızca bir NoSQL kategorisi için şema taşıması gerçekleştirebilirken, Alotaibi ve Pardede (Alotaibi & Pardede, 2019) çalışmalarını üç NoSQL kategorisine kadar yürütmüştür. Ek olarak, RL2NoSQL yöntemi, Çizelge 3.1'de gösterildiği gibi, NoSQL depolama sistemlerinin dört kategorisi için de şema oluşturmayı ve veri taşımayı içermektedir.

4. YÖNTEM VE METOD

Bu bölüm, mevcut bir ilişkisel veri tabanını okuma, ara JSON temsili oluşturma, şema tasarımı önerme ve verileri geleneksel RDBMS'den sunulan NoSQL depolama sistemlerinin kategorilerine aktarma işlemlerinin içinde yer alan adımlar da dahil olmak üzere, RL2NoSQL[5] yönteminin nasıl çalıştığına dair detaylı bir sekilde sunmaktadır. Ayrıca RL2NoSQL yöntemi, kaynak ilişkisel veri tabanlarından hedef NoSQL veri depolama sistemlerine şema üretimi ve veri dönüşümünü gerçekleştirmek için Python programlama dilinden yararlanır. Ayrıca, Şekil 4.1'de sunulan sözde kod, bu dönüşüm sürecinde RL2NoSQL yönteminin temel çalışma mantığını ve algoritmasını açıklar.

Algoritma: RL2NoSQL

Girdi: RDBMS veritabanı
Çıktı: Dönüştürülmüş NoSQL şema

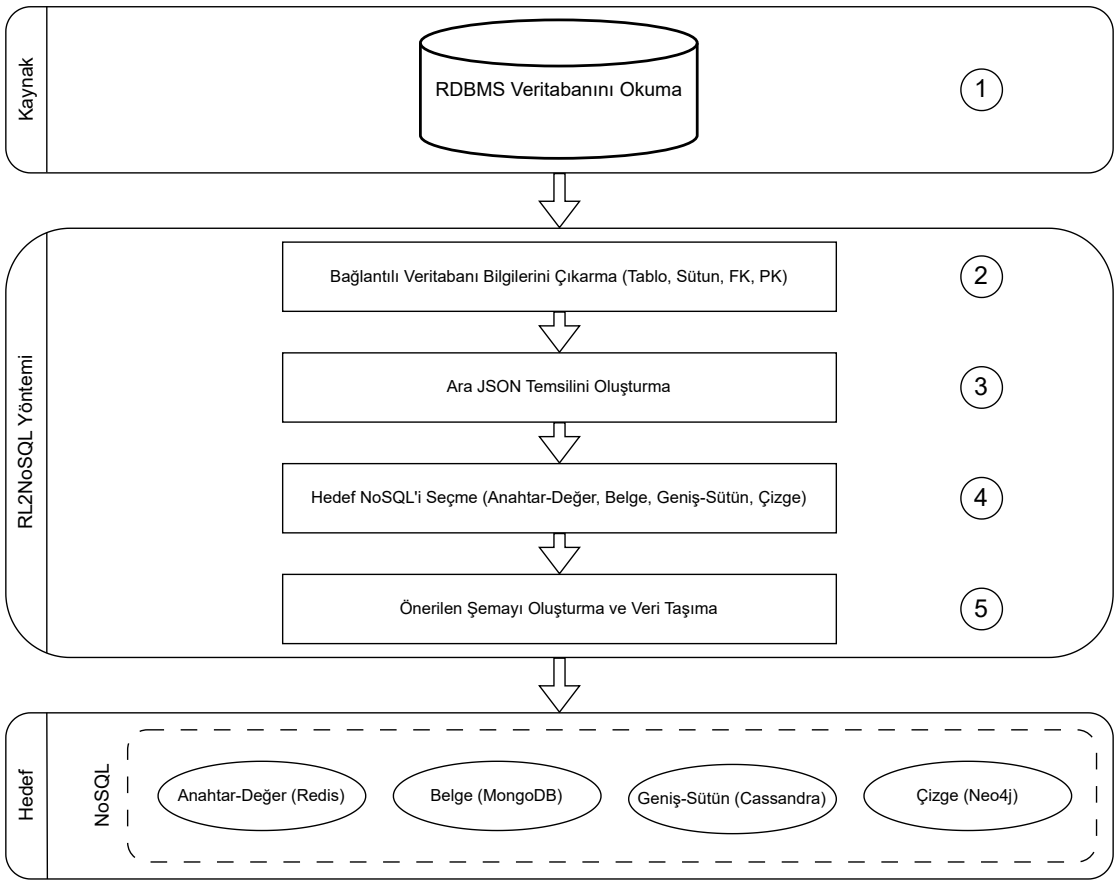
```
1 başla
2     Sağlanan kimlik bilgileri kullanılarak RDBMS'ye bağlan
3     Veritabanı bilgilerini oku ve çıkar (tablo, sütun, yabancı anahtar, birincil anahtar)
4     Çıkarılan şemayı ara JSON temsilcisine dönüştür
5     Bir hedef NoSQL türü seç (Anahtar-Değer, Belge, Geniş Sütun, Çizge)
6     Önerilen şemayı oluştur
7     Verileri taşı
8 bitir
```

Şekil 4.1 : RL2NoSQL şema dönüştürme algoritması için sözde kod.

Şekil 4.1'de, 2. ve 3. satırların RDBMS'ye bağlantı kurduğunu ve kaynak veri tabanı şeması hakkında özel bilgileri, tabloları, kolonları, yabancı anahtarları ve birincil anahtarları da içerecek şekilde aldığını göstermektedir. 4. satır ara JSON temsilinin oluşturulmasını belirtir. 5. satırda, belirli NoSQL depolama sistemlerinin seçimini vurgular. 6. satırda, seçilen hedef NoSQL'e göre, daha önce oluşturulan genel JSON yapısı, seçilen hedef NoSQL depolama sistemi için uygun esnek bir şemaya dönüştürülür. Son olarak, 7. satırda, kaynak RDBMS'de saklanan bilgiler seçilen NoSQL depolama sistemine aktarılır. Bu süreçte, NoSQL veri tabanına özgü veri tipleri ve anahtar-değer, belge, geniş kolon veya çizge gibi

[5] https://github.com/MehdiALOMER/RL2NoSQLMethodology

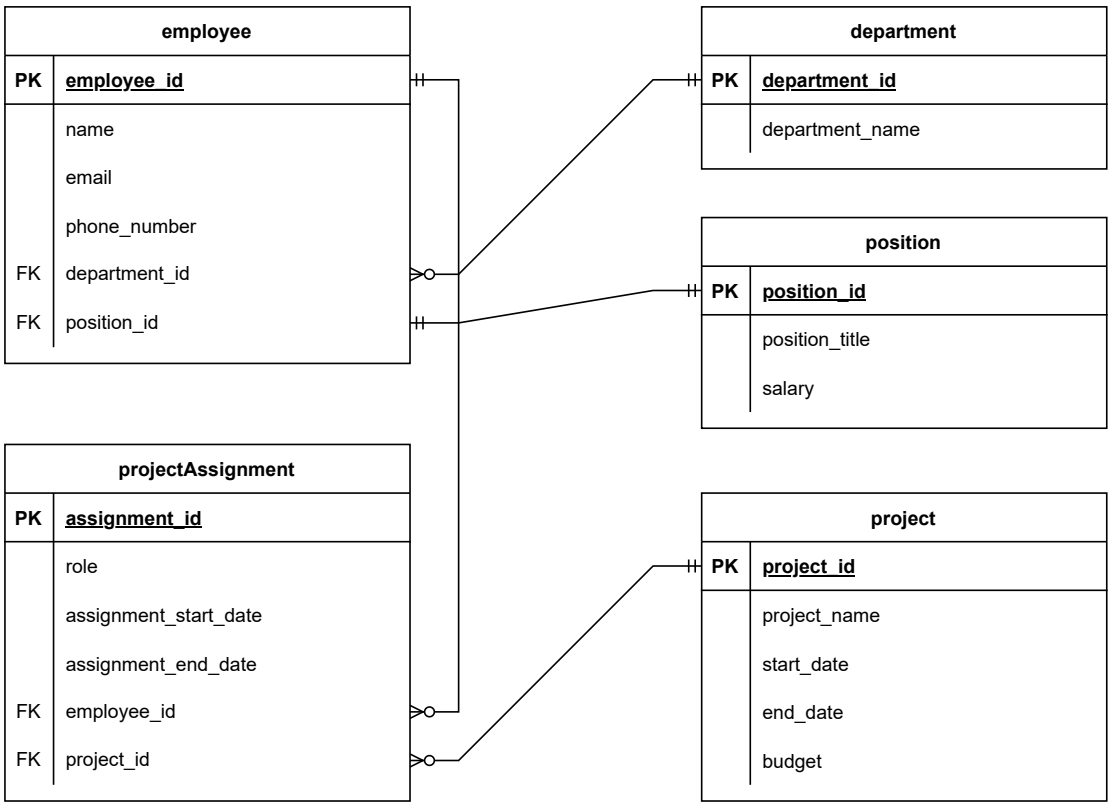
esnek yapısal özellikler her NoSQL sistemi için uygun özelleştirilmiş bir şema oluşturmak için dikkatlice kullanılır.



Şekil 4.2 : RL2NoSQL yöntemine genel bakış.

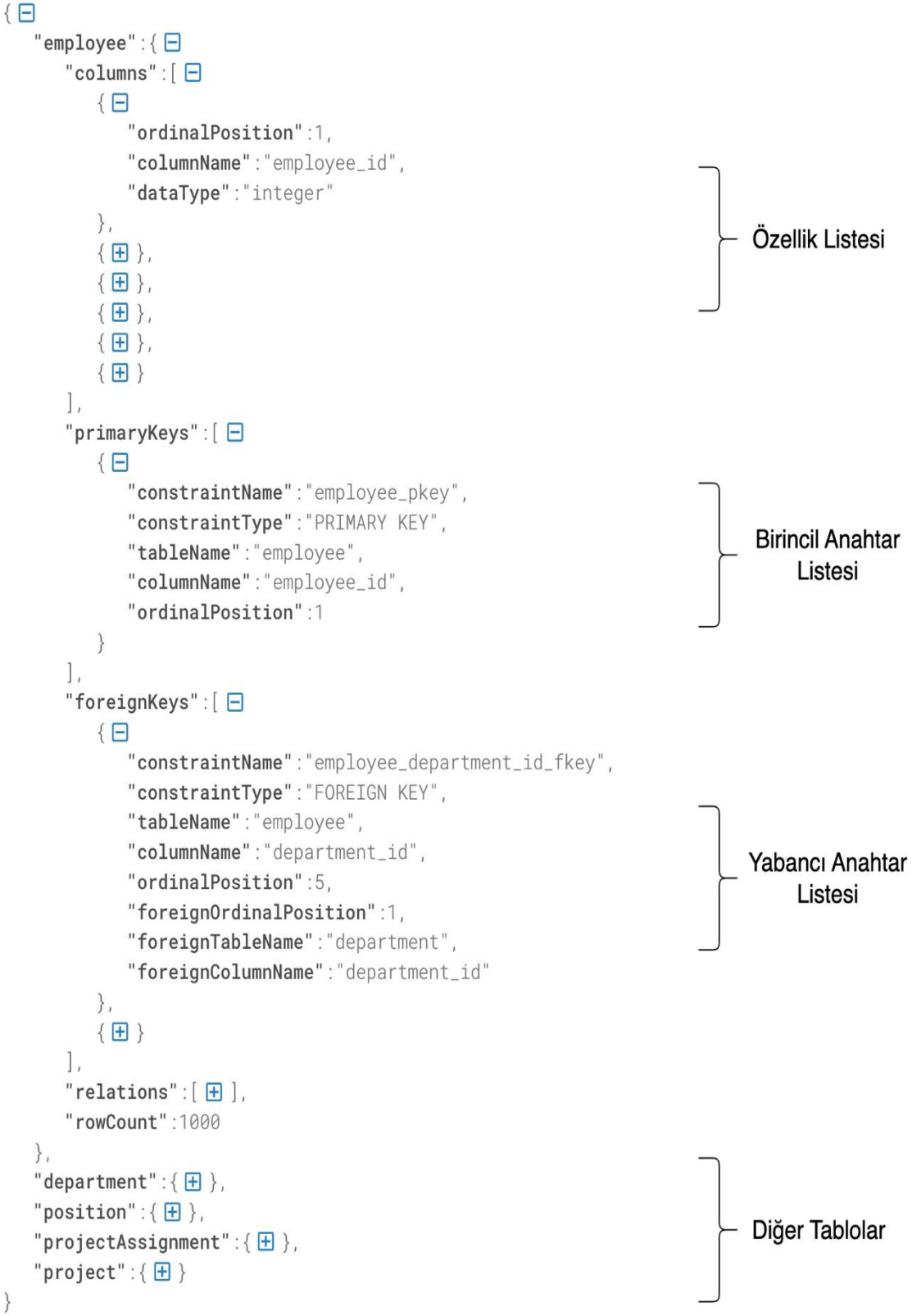
Şekil 4.2, RDBMS ile bağlantı kurmayı, mevcut veri tabanını ve ilişkisel şemasını okumayı, NoSQL veri teknolojileri için esnek şema tasarımları sunmayı ve RDBMS'den NoSQL'e veri aktarımını basitleştirmeyi amaçlayan RL2NoSQL metodunun aşamalarını sunmaktadır. RL2NoSQL'un çalışma mantığını açıklamak amacıyla, adım adım ve detaylı bir çalışma örneği sunulmaktadır. Sürecin açıklığını artırmak için, her adım için derinlemesine açıklamalar ve yorumlar bulunmaktadır.

RL2NoSQL yöntemini gerçekleştirmek için, PostgreSQL'de saklanan bir örnek çalışan veri tabanı kullanılmaktadır. Şekil 4.3, kaynak ilişkisel veri tabanının varlık ilişki (ER) modelini göstermektedir. Bu, okuyucularımızın, bu örnekteki dönüşüm sürecinin sonunda ilişkisel şemanın başlangıç yapısını son esnek NoSQL şemalarıyla karşılaştırabilmeleri için kaynak veri tabanının başlangıç ilişkisel yapısını gösterir.



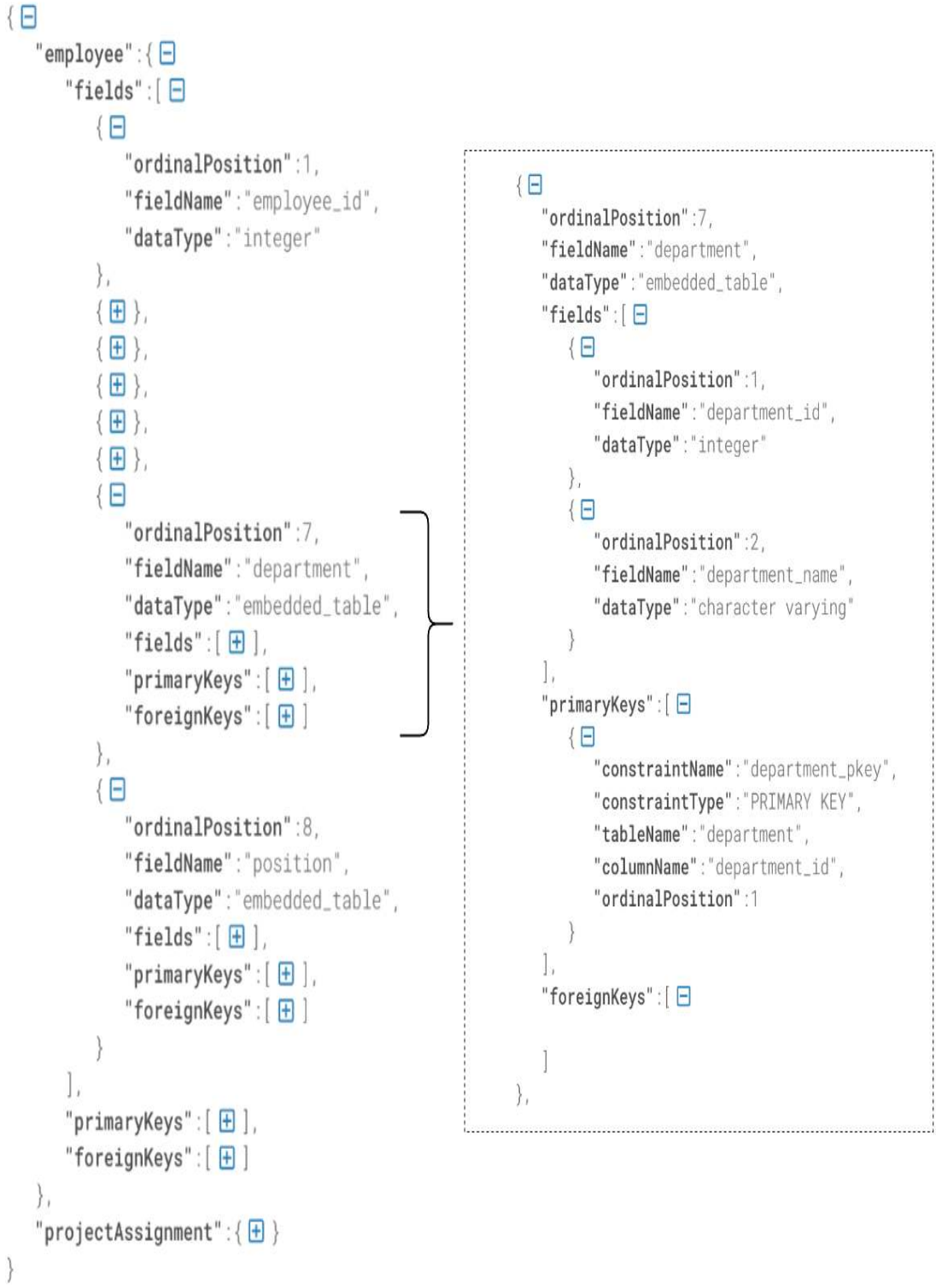
Şekil 4.3 : Örnek Çalışan veri tabanının ER modeli temsili.

İlk adımda, API'lar oluşturmak için yüksek performanslı bir web çatısı olan FastAPI kullanılarak yazılan bir Python kodunda, PostgreSQL ilişkisel veri tabanıyla başarılı bir şekilde bağlantı kuruldu. İkinci adımda, kaynak veri tabanındaki her tablonun tablo listesini, birincil ve yabancı anahtarlarını ve kolon türlerini almak için dikkatlice hazırlanmış bir dizi SQL sorgusu kullanıldı. Bu adımda, kaynak veri tabanının temel yapısal bilgileri, tablolar arasındaki bağlantıları tanımlamak ve belirlemek için kullanıldı. Şekil 4.3'te gösterilen çalışan veri tabanını okuduktan sonra, Şekil 4.4'te sunulan JSON temsili oluşturuldu. Şekil 4.4'te gösterildiği gibi, oluşturulan JSON temsilinde her tablo, bir anahtar ve değer çifti ile temsil edilmekte ve çalışan veri tabanı hakkındaki tüm bilgiler, kolonlar, her kolon için veri türleri, birincil anahtarlar, yabancı anahtarlar ve ilişkili tablolar da dahil edilmektedir.



Şekil 4.4 : Çalışan veri tabanını okuduktan sonraki JSON temsili.

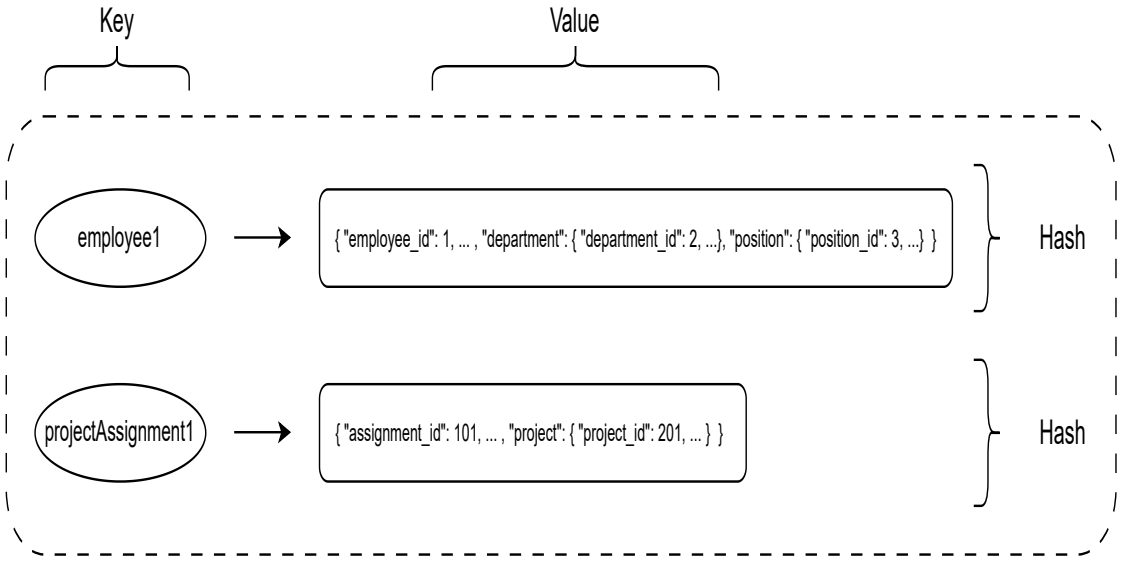
3. adımda, RL2NoSQL metodu, önceki adımda kaynak veri tabanından elde edilen bilgileri kullanarak ara JSON temsili oluşturmayı mümkün kılar. RL2NoSQL yönteminde ara JSON temsillerinin oluşturulması kritik öneme sahiptir çünkü bu temsiller daha sonraki NoSQL şema oluşturma işlemlerinde kullanılır. Bu adımda, NoSQL depolama sistemleri için uygun veri transferi için RL2NoSQL tarafından önceden tanımlanmış kurallar seti uygulanmıştır. Bu kurallar şu şekilde özetlenebilir: Eğer bir tablo ilişkisel veri tabanında yalnızca bir diğer tablo ile ilişkili ise, bu ilişki hedef NoSQL sistemlerinde "gömülü" olarak işaretlenir. Bir tablo birden fazla tablo ile ilişkili olduğunda, ilişkinin türüne bakılmaksızın "referanslama" modeli kullanılır. Bu ana kurallar ve stratejik analiz ile çeşitli NoSQL depolama sistemlerine geçişi kolaylaştırmak için bir ara JSON temsili oluşturulmuştur. Şekil 4.5, Şekil 4.3'te sağlanan ER modelinin, Şekil 4.4'te sağlanan JSON yapısı oluşturulduktan sonra ara JSON temsilini göstermektedir. RL2NoSQL dönüşüm kurallarına göre, departman ve pozisyon tabloları çalışanda gömülü olarak oluşturulmuştur ve proje tablosu projeAtama tablosuna gömülmüştür. Ara JSON temsillerinin oluşturulması önemlidir çünkü takip eden dönüşüm adımları bu ara JSON yapısını kullanır.



Şekil 4.5 : Ara JSON temsili.

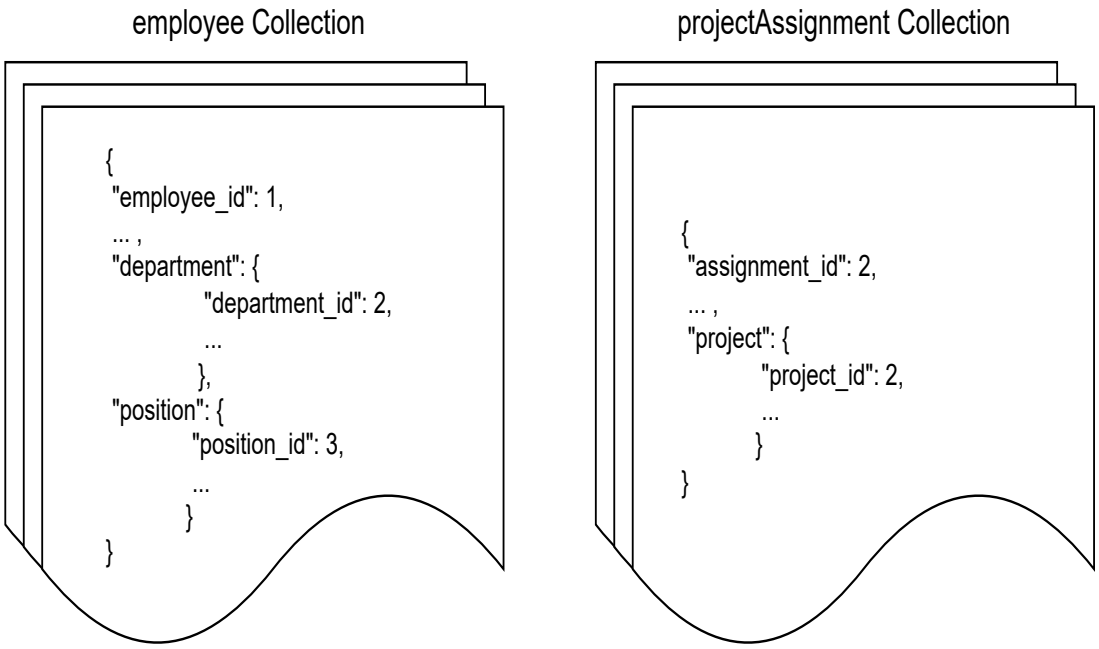
4. adımda, RL2NoSQL yöntemi kullanıcılara Redis, MongoDB, Apache Cassandra veya Neo4j gibi önerilen NoSQL depolama sistemlerinden birini seçme şansı sunulur. Ayrıca,

RL2NoSQL yöntemi, dört NoSQL depolama sistemleri kategorisinde şema geçişleri yapma kabiliyetini de içermektedir. 5. adımda, kullanıcının tercihi alınır ve 3. adımda oluşturulan ara JSON (Şekil 4.5'te gösterildiği gibi) kullanılır. Ardından, hedef şema oluşturma işlemi gerçekleştirilir ve bu işlem, veri dönüşüm ve taşınma sürecinin başlamasıyla devam eder. Şekil 4.6, bir ara JSON formunun Redis için önerilen şemasını gösterir. Şekil 4.7, MongoDB için önerilen şemayı sunar. Şekil 4.8, Cassandra için önerilen şemayı, Şekil 4.9 ise Neo4j için önerilen şemayı göstermektedir. Bu şekiller, önerilen esnek şemanın genel bir görünümünü sunar. Şema dönüşüm ve veri taşınma süreci boyunca, geçiş yapacağımız her NoSQL sistemi için uyumlu veri yapılarını ve veri tiplerini göz önünde bulundururuz.



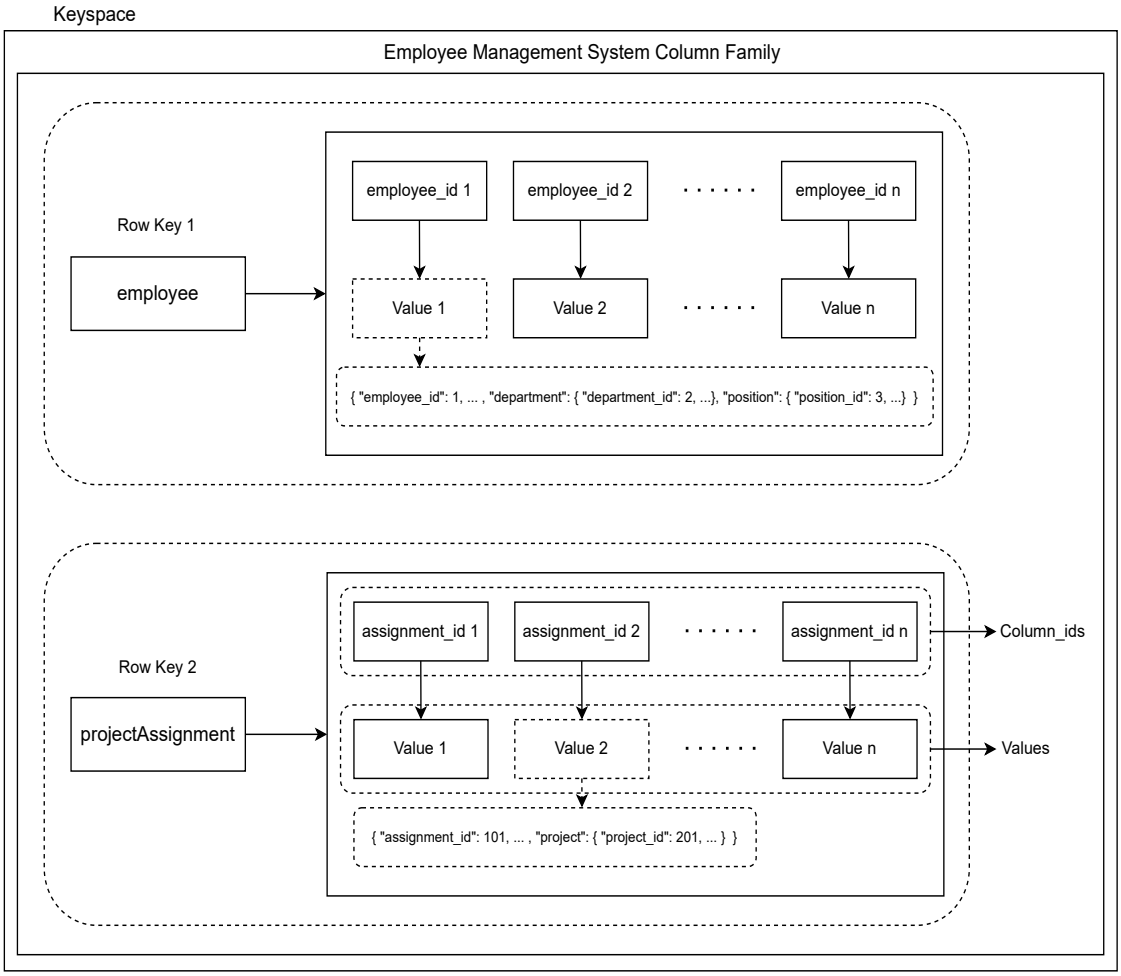
Şekil 4.6 : RL2NoSQL'in Redis için önerdiği şema.

Şekil 4.6, RL2NoSQL metodunun Redis NoSQL için önerdiği temsilini sunar. Redis, verileri anahtar-değer çiftleri şeklinde saklar. Temsilimizde, her anahtar (örneğin "çalışan" veya "projeAtama") bir hash'e karşılık gelir ve ilgili öznitelikleri içerir (örneğin, çalışanın pozisyonu, departmanı). Bu verilere erişmek ve sorgulamak için genellikle **`GET`** komutu bir anahtarın değerini almak, **`HSET`** bir hash içindeki bir alanın değerini ayarlamak ve **`HGETALL`** tüm hash'i almak gibi komutlar kullanılır.



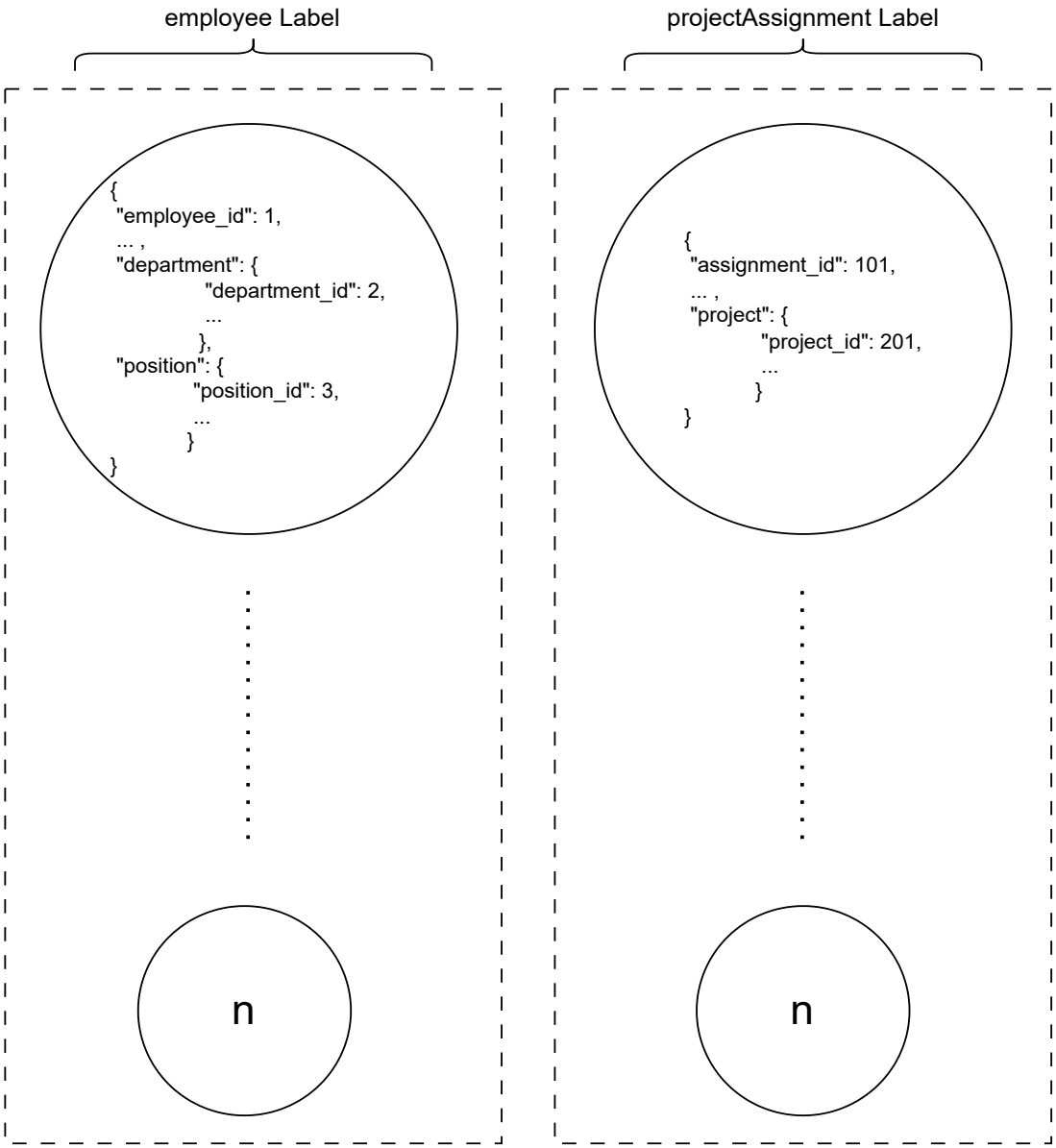
Şekil 4.7 : RL2NoSQL'in MongoDB için önerdiği şema.

Şekil 4.7, RL2NoSQL metodunun MongoDB NoSQL için önerdiği temsilini gösterir. MongoDB, verileri "çalışan" ve "projeAtama" gibi koleksiyonlarında saklar, her kayıt BSON formatında, JSON'a benzer bir şekilde, bir belge olarak saklanır. Her "çalışan" ve "projeAtama" ayrı belgelerde saklanabilir. Verilere, genellikle belgenin benzersiz tanımlayıcısı aracılığıyla `**find**` ve `**aggregate**` gibi sorgularla erişilir ve sorgulanır. Örneğin, çalışan koleksiyonundaki tüm belgeleri almak için `**db.employee.find({})**` veya belirli bir çalışanı ID'leri ile bulmak için MongoDB'nın sunmuş olduğu `**db.employee.find({"employee_id": 1})**` sorgulama yöntemini örnekteki gibi kullanılabilir.



Şekil 4.8 : RL2NoSQL'in Cassandra için önerdiği şema.

Şekil 4.8, RL2NoSQL metodunun Geniş kolon kategorisinde bulunan NoSQL veri depolama sistemleri için önerdiği şema temsilini sunar ve verilerin Apache Cassandra tarafından kolon ailelerinde nasıl saklanabileceğinin yöntemini sunar. Burada her satır bir anahtar (“çalışan”, “projeAtama”) ile tanımlanır. Her anahtar, ilgili kolonları ve karşılık gelen değerleri içeren genişleyebilir bir yapıya sahiptir. Apache Cassandra, verilere etkin bir şekilde erişmek ve sorgulamak için CQL (Cassandra Query Language) sorgulama dilini sağlar. CQL, SQL'e benzer şekilde, bir tablodan veri almak için `**SELECT**` ifadelerinin kullanılmasına olanak tanır. Veri almak için yaygın bir yöntem, belirli bir çalışanla ilişkili tüm kolonları elde etmek için `**SELECT * FROM employee WHERE employee_id = 1**` SQL sorgusunu çalıştırmak olacaktır.



Şekil 4.9 : RL2NoSQL'in Neo4j için önerdiği şema.

Şekil 4.9, RL2NoSQL metodunun Graf tabanlı NoSQL veri depolama sistemleri için önerdiği şema temsilini sunar. Neo4j temsili, verilerin bir çizge tabanlı sistemde saklandığını ve her kaydın "çalışan" veya "projeAtama" gibi etiketlerle kategorize edilen bir düğüm olarak temsil edildiğini gösterir. Neo4j'de verilere erişim ve sorgulama işlemi Cypher sorgulama dili kullanılarak gerçekleştirilir. Verileri almak için, `**employee**` etiketine sahip tüm düğümleri almak için `**MATCH (e:employee) RETURN e**` veya belirli bir çalışanı bulmak için `**MATCH (e:employee {employee_id: 1}) RETURN e**` gibi bir `**MATCH**` ifadesi kullanılabilir.

5. RL2NOSQL METODUNUN DEĞERLENDİRİLMESİ

Bu bölümde, herkese açık olan popüler bir örnek ilişkisel veri tabanı olan DVD Rental[6] veri tabanı kullanılarak RL2NoSQL yönteminin etkinliği değerlendirilmektedir. RL2NoSQL yöntemi, ilişkisel veri tabanları ve NoSQL depolama sistemleri arasında şema dönüşümünü ve veri taşınmasını kolaylaştırmayı amaçlar. RL2NoSQL yönteminin bu yeteneğini göstermek amacıyla, DVD Rental veri tabanı NoSQL depolama sistemlerinin dört kategorisine taşınmıştır. Şema dönüşüme ve veri taşınmasından sonra, kullanılan depolama sistemlerinin sorgu odaklı performans değerlendirmesi de sunulmuştur.

RL2NoSQL yönteminin girdisi, PostgreSQL için tasarlanmış kapsamlı bir test veri tabanı olan DVD Rental veri tabanıdır ve iş operasyonlarını temsil eden 15 tablodan oluşmaktadır. Bu veri tabanı aktör, kategori, film, film_aktör, film_kategori, müşteri, envanter, dil, kiralama, ödeme, personel, mağaza, adres, şehir ve ülke tablolarını içerir. Her tablo mağazanın çeşitli yönlerini detaylı olarak modeller. Bu tablolar arasındaki ilişkiler, film kiralama işlemleri, müşteri yönetimi, ödeme işlemleri ve envanter izleme dahil olmak üzere temel iş süreçlerini kolaylaştırmak için özel olarak tasarlanmıştır. Önceki bölümde sunulan geçiş adımları, DVD Rental veri tabanına uygulanmıştır. RL2NoSQL yöntemi, DVD Rental veri tabanı şemasını ve verilerini PostgreSQL'den Redis'e 1.8 saniyede, MongoDB'ye 4.57 saniyede, Apache Cassandra'ya 118.37 saniyede ve Neo4j'e 382.4 saniyede taşımıştır.

PostgreSQL'den belirtilen NoSQL depolama sistemlerine şema ve veri taşındıktan sonra, RL2NoSQL tarafından önerilen NoSQL şemasının veri sorgulama için ne kadar iyi çalıştığını görmek için Çizelge 5.1'deki sorguları dikkatlice hazırlandı. Ayrıca, Çizelge 5.1'deki beş sorgunun her biri için sonuç sayılarını sunmaktadır.

[6] https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/

Çizelge 5.1 : Gerçekleştirilen sorguların seçilmiş listesi.

Sorgu	Açıklama	Sonuç Sayısı
S1	"A" harfi ile başlayan şehirleri ülke bilgisi ile birlikte listele	43
S2	"Horror" türündeki filmlerin sayısını bul	56
S3	Rental tablosundaki tüm verileri al	16044
S4	Dili "English" olan filmleri getir	1000
S5	customer tablosundaki first_name alanının kelime haritasını oluştur	591

Çizelge 5.2, kaynak PostgreSQL ve hedef NoSQL depolama teknolojilerinde sorgulama işleminin nasıl yapıldığını göstermek için oluşturulmuştur. RDBMS'in aksine, NoSQL depolama teknolojileri için standart bir veri sorgulama yöntemi bulunmamaktadır. Her NoSQL teknolojisi, Çizelge 5.2'de gösterildiği gibi veri elde etmek için kendine özgü bir sorgulama dili kullanmaktadır.

Çizelge 5.2 : Sorgu 3 için sunulan teknolojilerdeki formları sorgulama.

Depolama Teknolojisi	Sorgulama Yöntemi	Sunulan Teknolojilerde S3'ün Gerçekleştirilmesi
PostgreSQL	SQL	SELECT * FROM rental
Redis	RQL	HGETALL rental
MongoDB	MQL	db.rental.find()
Cassandra	CQL	SELECT value FROM dvdrental. dvdrental WHERE row_key = 'rental'
Neo4j	Cypher	MATCH (n:rental) RETURN n

Sorgular, 11 çekirdekli CPU, 14 çekirdekli GPU ve 16 çekirdekli nöral motor özelliklerine sahip bir Apple M3 Pro çipi ile donatılmış güçlü bir cihaz olan 14 inç MacBook Pro üzerinde gerçekleştirildi. Ayrıca, geniş bir 18 GB birleşik bellek ve 512 GB SSD depolama alanına sahiptir. Sonuçların tutarlılığını sağlamak için her sorgu en az beş kez çalıştırıldı ve ortalama süreler kaydedilerek Çizelge 5.3'e eklendi.

Çizelge 5.3 : Veri sorgulama performansına göre depolama teknolojisi karşılaştırılması (ms).

Sorgu	PostgreSQL	Redis	MongoDB	Cassandra	Neo4j
S1	5.5	***3.99***	5.99	11.29	8.98
S2	4.99	***4.48***	***2.73***	12.45	10.60
S3	35.92	***30.64***	***28.85***	37.55	333.83
S4	11.01	***4.52***	***9.3***	***8.23***	40.70
S5	6.02	***2.69***	***4.36***	8.44	26.41

Redis'in bellek içi veri yapısı, Çizelge 5.3'te gösterildiği gibi diğer veri tabanlarına göre genellikle daha hızlı performans sergiler. Bu hız avantajı, özellikle okuma ve yazma işlemlerinde çok önemli olup, Redis'i önbellekleme, oturum yönetimi ve sıralama gibi senaryolar için tercih edilen bir seçenek haline getirir.

MongoDB, S2, S3, S4 ve S5 sorgularında PostgreSQL'e kıyasla daha hızlı yanıtlar sunarak veri modellerini ve sorgu optimizasyonunu etkin bir şekilde yönetme yeteneğini gösterir. Bu sonuçlar, MongoDB'nin büyük veri kümelerini hızlı ve etkili bir şekilde sorgulama kapasitesini vurgular ve böylece büyük ölçekli, yüksek performanslı uygulamalar için uygun bir seçenek olduğunu gösterir.

Apache Cassandra, yüksek yazma hızı ve ölçeklenebilirliği ile tanınır ve genellikle büyük veri kümeleri üzerinde hızlı sorgulama yetenekleri sunar. Ancak, tüm tabloları tek bir

tabloda depolama ve sorgulama için tüm tabloyu çağırmak için bir satır anahtarı kullanma gibi yaklaşımlar beklenen performans avantajlarını sınırlayabilir. Bu nedenle, Cassandra'nın avantajlarından tam olarak yararlanabilmek için veri tabanı modellemesi ve sorgu tasarımında dikkatli planlama gereklidir.

Neo4j'nin diğer veri tabanlarına kıyasla uzun sorgu süreleri, büyük miktarda düğümü işlemesi gerektiği için ortaya çıkar. Neo4j, düğümlere dayalı sorgulama konusunda oldukça yeteneklidir, ancak düğüm sayısındaki artış, sorguların performansını doğrudan etkileyebilir. Bu, özellikle belirli senaryolar için Neo4j'nin avantajlarını değerlendirirken kapsamlı bir performans analizi yapılmasının önemini vurgular.

Sonuç olarak, bu analiz RL2NoSQL yöntemi tarafından sunulan şema önerilerine dayanır. Sonuçların çeşitli tasarımlara ve veri tabanlarına bağlı olarak farklılık gösterebileceği akılda tutulmalıdır. Sorgu performansı değerlendirmesi, belirli sorgu türleri için özelleştirilmemiş genel bir geçiş yaklaşımı uyguladığımızı ortaya koyar. PostgreSQL, özellikle önceden oluşturulmuş indeksleri kullanarak küçük bir veri alt kümesini filtreleme durumlarında bazı durumlarda NoSQL sistemlerinden daha hızlı sorgu sonuçları elde edebilir. Bununla birlikte, NoSQL sistemlerini benimsemenin ana amacı sadece sorgu hızını artırmakla sınırlı değildir. Bu, geniş ölçeklenebilirlik, uyarlanabilir veri modelleri ve verileri dağıtık bir şekilde depolayıp işleme yeteneği gibi avantajları kullanmayı kapsar. Çizelge 5.3'te gösterilen sonuçlar, RL2NoSQL yönteminin şema önerisinin veri sorgulama için de umut verici sonuçlar sağladığını göstermektedir.

6. RL2NoSQL METODUNUN UYGULANMASI

Bu bölümde, tezin ana teması olan RL2NoSQL metodunun pratik uygulanışı ele alınmaktadır. Bu kısımda, ilişkisel veri tabanlarından NoSQL veri depolama sistemlerine geçiş sürecini kolaylaştıran RL2NoSQL metodolojisinin temel aşamaları detaylı olarak görselleştirilmiş ve açıklanmıştır. Her bir adım, RL2NoSQL metodunun kod blokları ile desteklenerek, bu karmaşık sürecin nasıl yönetildiği ve uygulandığı gösterilmektedir. Bu bölümdeki görseller, metodun işleyişini, şema dönüşümünü ve veri aktarımını adım adım izlemek için hazırlanmıştır.

```
280  def getRelationSchema():
281      data = {}
282      connection = connectPostgres()
283      cursor = connection.cursor()
284      cursor.execute("""SELECT table_nameFROM information_schema.tables
285                  WHERE table_type = 'BASE TABLE' AND table_schema
286                  NOT IN ('pg_catalog', 'information_schema');
287                  """)
288      tableList = [table[0] for table in cursor.fetchall()]
289      for table in tableList:
290          columns = getcColumnList(table)
291          primaryKeys = getPrimaryKeyList(table)
292          foreignKeys = getForeignKeyList(table)
293          relations = getRelationList(table)
294          rowCount = getRowCount(table)
295          data[table] = {"columns": columns, "primaryKeys": primaryKeys, "foreignKeys": foreignKeys,
296                         "relations": relations, "rowCount": rowCount}
297      global mainDict
298      mainDict = getRelationTypes(data)
299      closePostgres(connection)
300      return mainDict
```

Şekil 6.1 : RDBMS veri tabanı bağlantısı ve meta verilerin çekilmesi.

Şekil 6.1, RL2NoSQL metodunun başlangıç aşamasını gösterir. Burada, RDBMS veri tabanına bağlanıp gerekli bilgilerin (birincil anahtar listesi, veri tabloları ve ilişkiler gibi) alınması süreci kod blokları ile temsil edilmiştir. Bu adım, dönüşüm işlemi için temel yapı taşlarını sağlar ve sistemin veri bütünlüğünü ve doğruluğunu korumasına olanak tanır. Kod, veri tabanı yapılandırmasının incelenmesi ve gerekli meta verilerin çekilmesi için SQL sorguları içerir, böylece sonraki dönüşüm adımlarında kullanılmak üzere stratejik bilgiler elde edilir.

```
def getSharedDict():
    global mainDict
    for table in mainDict:
        mainDict[table]["isChecked"] = False
    for table in mainDict:
        if mainDict[table]["isChecked"] == False:
            if len(mainDict[table]["relations"]) > 0:
                embedded_tables = []
                for relation in mainDict[table]["relations"]:
                    if len(mainDict[relation["foreign_table_name"]]["relations"]) == 0:
                        if mainDict[relation["foreign_table_name"]]["isChecked"] == True:
                            mainDict[relation["foreign_table_name"]]["isChecked"] = False
                            del sharedDict[relation["foreign_table_name"]]
                        if relation["foreign_table_name"] not in embedded_tables:
                            embedded_tables.append(relation["foreign_table_name"])
                sharedDict[table] = {}
                sharedDict[table]["fields"] = []
                sharedDict[table]["primaryKeys"] = []
                sharedDict[table]["foreignKeys"] = []
                for column in mainDict[table]["columns"]:...
                for primaryKey in mainDict[table]["primaryKeys"]:...
                for foreignKey in mainDict[table]["foreignKeys"]:...
                if len(embedded_tables) > 0:
                    for embedded_table in embedded_tables:
                        sharedDict[table]["fields"].append({
                            "ordinalPosition": 0,
                            "fieldName": embedded_table,
                            "dataType": "embedded_table"
                        })
                        for field in sharedDict[table]["fields"]:
                            if field["fieldName"] == embedded_table and field["ordinalPosition"] == 0:
                                field["fields"] = []
                                field["primaryKeys"] = []
                                field["foreignKeys"] = []
                                for embedded_table_column in mainDict[embedded_table]["columns"]:...
                                for embedded_table_primaryKey in mainDict[embedded_table]["primaryKeys"]:...
                                for embedded_table_foreignKey in mainDict[embedded_table]["foreignKeys"]:...
                mainDict[table]["isChecked"] = True
                for embedded_table in embedded_tables:
                    mainDict[embedded_table]["isChecked"] = True
            else:
                sharedDict[table] = {}
                sharedDict[table]["fields"] = []
                sharedDict[table]["primaryKeys"] = []
                sharedDict[table]["foreignKeys"] = []
                for column in mainDict[table]["columns"]:...
                for primaryKey in mainDict[table]["primaryKeys"]:...
                for foreignKey in mainDict[table]["foreignKeys"]:...
                mainDict[table]["isChecked"] = True
    return sharedDict
```

Şekil 6.2 : Ara JSON oluşturma.

Şekil 6.2'de, RL2NoSQL metodunun ara JSON oluşturma süreci görselleştirilmiştir. Bu aşama, ilişkisel veri tabanından elde edilen verilerin, daha sonra NoSQL sistemlerine aktarımı sırasında kullanılmak üzere JSON formatına dönüştürülmesini içerir. Burada sunulan kod, çeşitli veri türleri ve ilişkileri doğru bir şekilde ele alacak şekilde tasarlanmıştır, böylece dönüştürülen verilerin NoSQL sistemlerinde uygun şekilde saklanması ve yönetilmesi sağlanır.

```
427  def getMongoDBDict():
428      global sharedDict
429      data = sharedDict
430      for table in data:
431          for field in data[table]["fields"]:
432              if field["dataType"] == "embedded_table":
433                  for embedded_table_field in field["fields"]:
434                      embedded_table_field["noSQLDataType"] = (
435                          datatypes)[embedded_table_field["dataType"]]["mongoDBDataType"]
436              else:
437                  field["noSQLDataType"] = datatypes[field["dataType"]]["mongoDBDataType"]
438      global mongoDBDict
439      mongoDBDict = data
440      return data
```

Şekil 6.3 : MongoDB için özelleştirilmiş şema oluşturma.

Şekil 6.3'de, MongoDB için özel bir şema oluşturulmasını içeren kodu içermektedir. Bu, RL2NoSQL metodunun esnekliğini gösteren kritik bir adımdır çünkü her NoSQL sistemine özgü gereksinimleri karşılayacak şekilde özelleştirilmiş şemalar oluşturulur. Kod bloğu, MongoDB'nin belge tabanlı yapısına uygun olarak veri şemalarını nasıl yapılandıracağını ve entegre edeceğini detaylandırır.

```python
def transferToMongoDB():
    mongoDBClient = connectMongoDB()
    connection = connectPostgres()
    cursor = connection.cursor()
    start_time = time.time()
    global mongoDBDict, foreignKeyIndex
    for table in mongoDBDict:
        cursor.execute(f"""SELECT *FROM {table}
            ORDER BY {mongoDBDict[table]["primaryKeys"][0]["columnName"]} ASC
        """)
        rows = cursor.fetchall()
        for row in rows:
            document = {}
            for field in mongoDBDict[table]["fields"]:
                if field["dataType"] != "embedded_table":
                    if field["dataType"] == "numeric":
                        document[field["fieldName"]] = float(row[mongoDBDict[table]["fields"].index(field)])
                    elif field["dataType"] == "date" or field["dataType"] == "bytea":
                        document[field["fieldName"]] = str(row[mongoDBDict[table]["fields"].index(field)])
                    else:
                        document[field["fieldName"]] = row[mongoDBDict[table]["fields"].index(field)]
                else:
                    for foreignKey in mongoDBDict[table]["foreignKeys"]:
                        if foreignKey["foreignTableName"] == field["fieldName"]:
                            for field_1 in mongoDBDict[table]["fields"]:
                                if field_1["fieldName"] == foreignKey["columnName"]:
                                    foreignKeyIndex = mongoDBDict[table]["fields"].index(field_1)
                                    break
                    cursor.execute(f""" SELECT *FROM {field["fieldName"]}
                        WHERE {field["primaryKeys"][0]["columnName"]} = {row[foreignKeyIndex]}
                    """)
                    embedded_table_rows = cursor.fetchall()
                    if len(embedded_table_rows) > 0:
                        document[field["fieldName"]] = {}
                        for embedded_table_field in field["fields"]:
                            if embedded_table_field["dataType"] == "numeric":
                                document[field["fieldName"]][embedded_table_field["fieldName"]] = float(
                                    embedded_table_rows[0][field["fields"].index(embedded_table_field)])
                            elif embedded_table_field["dataType"] == "date" or embedded_table_field[
                                "dataType"] == "bytea":
                                document[field["fieldName"]][embedded_table_field["fieldName"]] = str(
                                    embedded_table_rows[0][field["fields"].index(embedded_table_field)])
                            else:
                                document[field["fieldName"]][embedded_table_field["fieldName"]] = \
                                embedded_table_rows[0][
                                    field["fields"].index(embedded_table_field)]
            collection = mongoDBClient[table]
            collection.insert_one(document)
    end_time = time.time()
    print("Transfer to MongoDB is successful --- %s seconds ---" % (end_time - start_time))
    closePostgres(connection)
    return {"message": "Transfer to MongoDB is successful"}
```

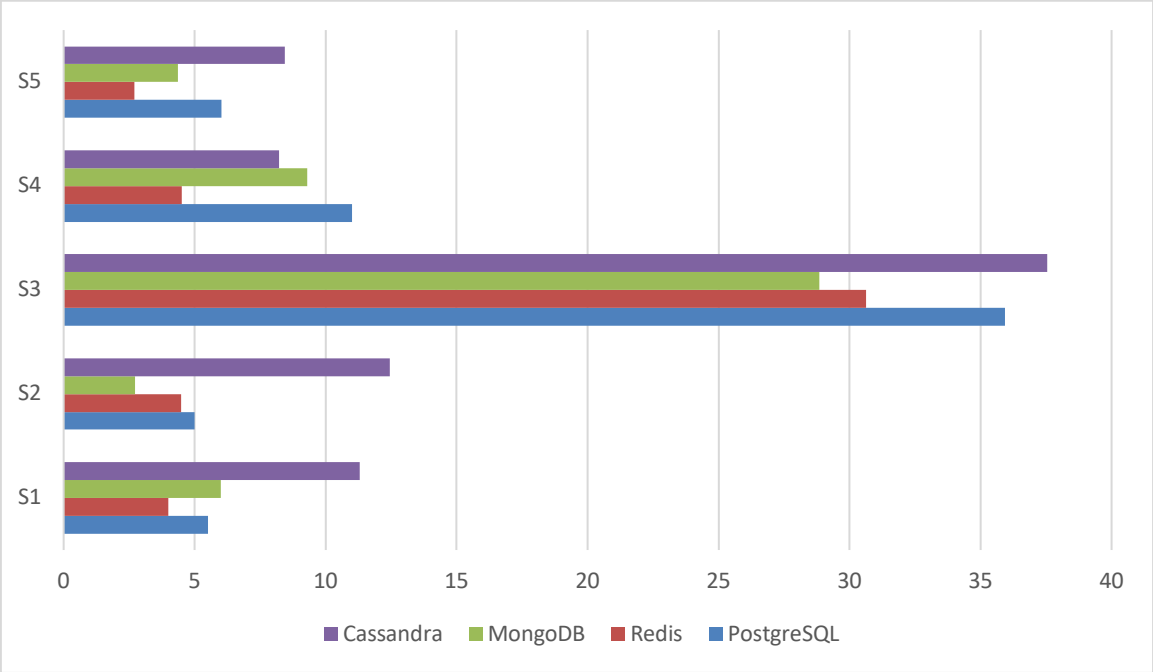
Şekil 6.4 : MongoDB'ye veri aktarımı.

Şekil 6.4’de, elde edilen şema kullanılarak MongoDB'ye veri transferini gerçekleştiren işlem kodu yer almaktadır. Bu adım, verilerin yeni NoSQL şemasına nasıl aktarıldığını ve bu süreçte karşılaşılan teknik zorlukların nasıl üstesinden gelindiğini gösterir. Kod, veri aktarım işleminin verimliliğini ve etkinliğini artırmak için optimize edilmiş metodları içerir, böylece büyük veri setlerinin hızlı ve güvenilir bir şekilde taşınması sağlanır.

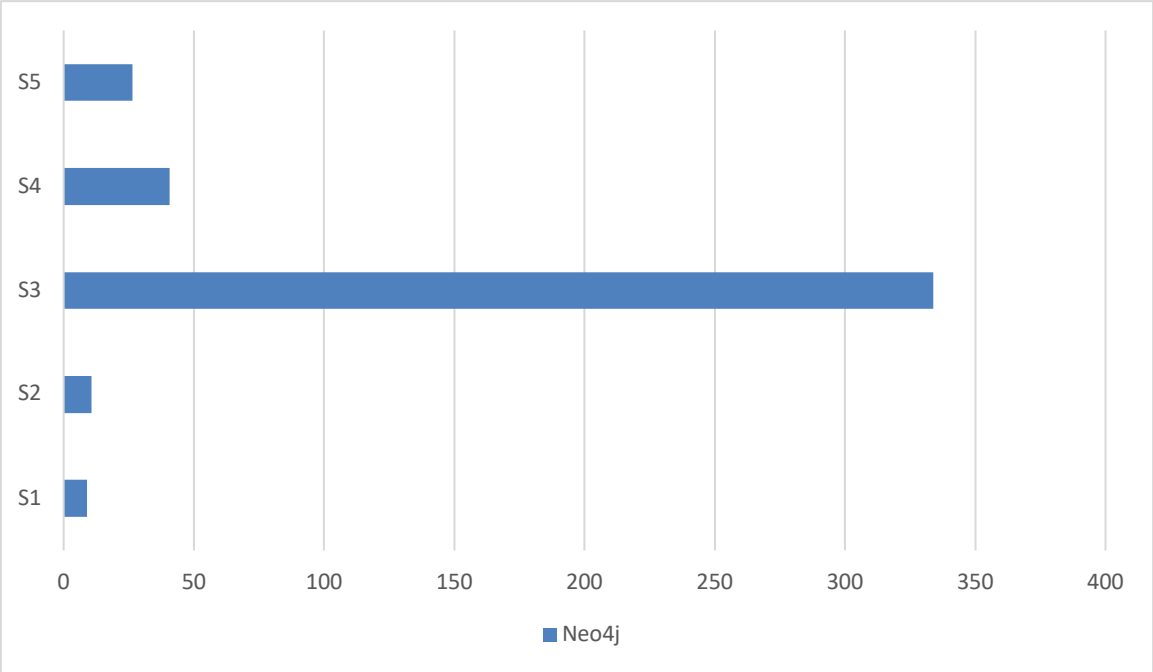
Bu şekiller, RL2NoSQL metodunun uygulanmasının her adımında teknik derinliği ve işlevselliği açısından zengin bilgiler sunarak, RL2NoSQL yönteminin pratikte nasıl işlediğini anlamak için değerli bir kaynaktır. Her biri, veri yönetimi ve dönüşüm süreçlerinin nasıl optimize edildiğini ve çeşitli NoSQL sistemlerine nasıl uyum sağlandığını detaylı bir şekilde ortaya koymaktadır.

7. BULGULAR VE ÖNERİLER

RL2NoSQL metodu, ilişkisel veri tabanı yönetim sistemlerinden çeşitli NoSQL depolama sistemlerine şema dönüşüme ve veri transferini otomatize eden yenilikçi bir çözümdür. Bu yöntem, özellikle Anahtar-Değer, Belge, Geniş Kolon ve Çizge depolama sistemleri olmak üzere farklı NoSQL veri depolama sistem kategorilerine etkili bir şekilde veri aktarabilme kapasitesine sahiptir. Kullanıcıların artan ölçeklenebilirlik, esneklik ve uyumluluk taleplerini karşılamak amacıyla geliştirilen bu metod, kuruluşların veri yönetim stratejilerini geleceğe taşımalarını sağlamaktadır. Yapılan değerlendirmeler, önerilen şema tasarımlarının veri sorgulama operasyonlarında başarılı sonuçlar verdiğini göstermiştir. Özellikle Redis, MongoDB, Apache Cassandra ve Neo4j gibi farklı NoSQL veri depolama sistemlerine uyum sağlayan şemalar, sorgulama performansını iyileştirmede büyük bir etkinlik göstermektedir. Bu sonuçlar, RL2NoSQL metodunun, veri bütünlüğü ve tutarlılığını korurken, veri ve şema dönüşüm süreçlerini basitleştirdiğini ve farklı NoSQL kategorilerine uyum sağlama yeteneğini başarıyla kanıtlamaktadır.



Şekil 7.1 : Farklı veri tabanı sistemlerinde sorgu yanıt süreleri.



Şekil 7.2 : Neo4j sorgu yanıt süreleri.

Grafiklerde belirtilen süreler, bu sistemlerin her birinde yapılan sorgu işlemlerinin ne kadar sürede tamamlandığını ve RL2NoSQL yönteminin veri yapılarını ne ölçüde optimize ettiğini açıkça ortaya koymaktadır. Bu bulgular, yöntemin hem performans hem de ölçeklenebilirlik açısından potansiyelini vurgulamakta, aynı zamanda gelecekteki iyileştirmeler için bir temel oluşturmaktadır.

Bu bulgular ışığında, RL2NoSQL metodunun gelecekteki gelişim yönleri üzerinde durulması önem arz etmektedir. Öncelikle, geniş kolon ve çizge yapıları üzerindeki dönüşümlerin iyileştirilmesine yönelik çalışmalar planlanmaktadır. Bu tür yapıların daha etkin bir şekilde ele alınması, genel dönüşüm sürecinin verimliliğini ve otomasyonunu artırarak, zaman ve kaynak tasarrufu sağlayacaktır. Ayrıca, kullanıcıların dönüşüm sürecine daha aktif katılımını sağlamak amacıyla kullanıcı dostu arayüzlerin geliştirilmesi düşünülmektedir. Bu arayüzler, dönüşüm süreçlerinin her aşamasını görsel olarak temsil ederek, kullanıcıların süreci daha iyi anlamalarını ve yönetmelerini kolaylaştırması hedeflenmektedir. Özellikle, önerilen kullanıcı arayüzünün, şema dönüşüm aşamalarını adım adım göstererek kullanıcılara işlemler üzerinde daha fazla kontrol ve görünürlük sağlaması amaçlanmaktadır.

Son olarak, RL2NoSQL metodunun çeşitli alanlardaki gerçek dünya uygulamalarında test edilmesi büyük önem taşımaktadır. Bu testlerden elde edilecek olan geri bildirimler,

metodun daha da iyileştirilmesine yardımcı olacak ve farklı sektörlerden gelen ihtiyaç ve beklentiler doğrultusunda yöntemin uyarlanmasına imkân sağlayacaktır. Geri bildirimler, aynı zamanda yöntemin geniş bir kullanıcı yelpazesine uygun hale getirilmesinde kritik bir rol oynayacaktır. RL2NoSQL metodunun sürekli olarak güncellenmesi ve iyileştirilmesi, teknolojik gelişmelere ve pazar ihtiyaçlarına hızlı bir şekilde adapte olmasını sağlayacak ve yeni NoSQL sistemlerinin ve dönüşüm tekniklerinin entegrasyonu yöntemin uygulanabilirliğini ve etkinliğini artıracaktır. Bu kapsamlı çalışmalar ve geliştirmeler, RL2NoSQL metodunun veri yönetiminde geleceği şekillendirmede önemli bir rol oynamasını sağlayacaktır.

8. SONUÇ

Bu tez çalışmasında, ilişkisel veri tabanlarından NoSQL veri depolama sistemlerine geçişi kolaylaştırmayı hedefleyen RL2NoSQL metodu geliştirilmiştir. Günümüzün veri odaklı dünyasında, büyük verinin yönetimi ve esnek veri modellerine duyulan ihtiyaç, veri tabanı yönetim sistemlerinde yenilikçi çözümler arayışını beraberinde getirmiştir. Bu araştırmada sunulan RL2NoSQL yaklaşımı, Anahtar-Değer, Belge, Geniş-Kolon ve Çizge olmak üzere dört NoSQL kategorisine otomatik veri tabanı dönüşümü sağlayarak bu gereksinime yanıt vermektedir.

RL2NoSQL metodolojisi, ilişkisel veri tabanı yapısından, esnek şemalı NoSQL veri depolama sistemlerine geçişte, veri bütünlüğünü ve tutarlılığını korurken veri kaybı ya da bozulması olmadan dönüşüm gerçekleştirebilmenin önünü açmıştır. PostgreSQL'den farklı NoSQL veri depolama sistemlerine yapılan dönüşüm işlemlerinde, elde edilen performans sonuçları, NoSQL sistemlerinin ölçeklenebilirlik, veri modellenmesi esnekliği ve yüksek hızlı işlem kabiliyetleri açısından avantajlarını ortaya koymuştur. Böylece, RL2NoSQL, veri tabanı dönüşüm sürecinin karmaşıklığını azaltarak, iş süreçlerinin verimliliğini artırmayı başarmıştır.

Bulunan sonuçlar göstermiştir ki, RL2NoSQL yöntemi, farklı NoSQL veri depolama sistemleri için şema önerileri sağlamakla kalmamış, aynı zamanda veri sorgulama operasyonlarında da potansiyel iyileştirmeler sunmuştur.

Sonuç olarak, RL2NoSQL metodolojisi, ilişkisel veri tabanlarından NoSQL veri depolama sistemlerine geçiş sürecinde bir dönüşüm yazılımı işlevi görmekte ve bu alandaki araştırma ile uygulamalar için yeni kapılar açmaktadır. İş süreçlerinde verimliliği ve iş akışlarında hızı artırma potansiyeli ile RL2NoSQL, özellikle büyük veri analitiği ve gerçek zamanlı veri işleme gibi alanlarda veri tabanı yönetimi pratiklerine yeni bir boyut getirmiştir. Bu çalışmanın sağladığı anlayış ve metodoloji, işletmelerin ve organizasyonların veri yönetimi stratejilerini yeniden şekillendirmeleri ve geleceğin veri ihtiyaçlarına cevap verebilmeleri için güçlü bir temel oluşturmaktadır.

KAYNAKLAR

Abdelhedi, F., Jemmali, R., & Zurfluh, G. (2022). *Relational Databases Ingestion into a NoSQL Data Warehouse*.

Al Mahruqi, R. S., Alalfi, M. H., & Dean, T. R. (2019). A Semi-automated Framework for Migrating Web applications from SQL to Document Oriented NoSQL Database. *ACM Computing Surveys*.

Ali, W., Shafique, M. U., Majeed, M. A., & Raza, A. (2019). Comparison between SQL and NoSQL Databases and Their Relationship with Big Data Analytics. *Asian Journal of Research in Computer Science*, 1–10. https://doi.org/10.9734/ajrcos/2019/v4i230108

Alotaibi, O., & Pardede, E. (2019). Transformation of Schema from Relational Database (RDB) to NoSQL Databases. *Data*, *4*(4), 148. https://doi.org/10.3390/data4040148

Averbuch, A., Boncz, P., Neumann, T., Association for Computing Machinery. Special Interest Group on Management of Data, ACM Digital Library., & ACM-SIGMOD International Conference on Management of Data (2013 : New York, N. Y.). (2013). *First International Workshop on Graph Data Management Experiences and Systems (GRADES2013) : co-located with SIGMOD/PODS 2013 : New York, NY, USA*.

Aydin, A. A. (2023). A Comparative Perspective on Technologies of Big Data Value Chain. *IEEE Access*, *11*, 112133–112146. https://doi.org/10.1109/ACCESS.2023.3323160

Aydin, A. A., & Anderson, K. M. (2020). Data modelling for large-scale social media analytics: design challenges and lessons learned. *International Journal of Data Mining, Modelling and Management*, *12*(4), 386. https://doi.org/10.1504/IJDMMM.2020.111409

Belkadi, F., & Esbai, R. (2021). A Model-Driven Engineering: From Relational Database to Document-oriented Database in Big Data Context. *Proceedings of the 16th International Conference on Software Technologies*, 653–659. https://doi.org/10.5220/0010604906530659

Chen, J.-K., & Lee, W.-Z. (2020). The Transformation of Relational Database to Wide Column Store Database. *2020 International Symposium on Computer, Consumer and Control (IS3C)*, 384–386. https://doi.org/10.1109/IS3C50286.2020.00105

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377–387. https://doi.org/10.1145/362384.362685

Doguc, T. B., & Aydın, A. A. (2019). CAP-based Examination of Popular NoSQL Database Technologies in Streaming Data Processing. *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, 1–6. https://doi.org/10.1109/IDAP.2019.8875874

Eldrrat, H. A., & Maatuk, A. M. (2023). A Novel Approach for Migrating Relational Databases into Graph Databases. *2023 3rd International Conference on Emerging Smart Technologies and Applications (ESmarTA)*, 1–8. https://doi.org/10.1109/eSmarTA59349.2023.10293437

Erraji, A., Maizate, A., & Ouzzif, M. (2023). New Structural and Semantic Transformation of Relational Databases to MongoDB. *International Journal of Emerging Technology and Advanced Engineering*, *13*(5), 46–72. https://doi.org/10.46338/ijetae0523_06

Feng, H., & Huang, M. (2022). An Approach to Converting Relational Database to Graph Database: from MySQL to Neo4j. *2022 IEEE 2nd International Conference on*

Power, Electronics and Computer Applications (ICPECA), 674–680. https://doi.org/10.1109/ICPECA53709.2022.9719151

Gopalan, M. G., Prasanna, C., Srihari Krishna, Y. V., Shanthini, B., & Arulkumar, A. (2017). MYSQL to cassandra conversion engine. *2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS)*, 503–508. https://doi.org/10.1109/SSPS.2017.8071648

Hamouda, S. (2023). Seamless Transition: Migrating from Relational Databases to Document-Oriented Databases. *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 883–888. https://doi.org/10.1109/IDAACS58523.2023.10348946

Hamouda, S., & Zainol, Z. (2017). Document-Oriented Data Schema for Relational Database Migration to NoSQL. *2017 International Conference on Big Data Innovations and Applications (Innovate-Data)*, *2018-January*, 43–50. https://doi.org/10.1109/Innovate-Data.2017.13

KEKEVİ, U., & AYDIN, A. A. (2022). Real-Time Big Data Processing and Analytics: Concepts, Technologies, and Domains. *Computer Science*. https://doi.org/10.53070/bbd.1204112

Mahmood, A. A. (2018). Automated Algorithm for Data Migration from Relational to NoSQL Databases. *Al-Nahrain Journal for Engineering Sciences*, *21*(1), 60. https://doi.org/10.29194/NJES21010060

Namdeo, B., & Suman, U. (2021a). A Model for Relational to NoSQL database Migration: Snapshot-Live Stream Db Migration Model. *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 199–204. https://doi.org/10.1109/ICACCS51430.2021.9441829

Namdeo, B., & Suman, U. (2021b). Schema design advisor model for RDBMS to NoSQL database migration. *International Journal of Information Technology*, *13*(1), 277–286. https://doi.org/10.1007/s41870-020-00515-8

Nan, Z., & Bai, X. (2019). The study on data migration from relational database to graph database. *Journal of Physics: Conference Series*, *1345*(2), 022061. https://doi.org/10.1088/1742-6596/1345/2/022061

Revathi, K., Tamilselvi, T., Dhanwanth, B., & Dhivya, M. (2023). Auto JSON: An Automatic Transformation Model for Converting Relational Database to Non-relational Documents. *International Journal of Advanced Computer Science and Applications*, *14*(3), 2023. https://doi.org/10.14569/IJACSA.2023.0140377

Singh, A. (2019). Data Migration from Relational Database to MongoDB using XAMPP and NoSQL. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3372802

Singh, M., & Kaur, K. (2015). SQL2Neo: Moving health-care data from relational to graph databases. *2015 IEEE International Advance Computing Conference (IACC)*, 721–725. https://doi.org/10.1109/IADCC.2015.7154801

Tandya, W., & Azizah, F. N. (2023). Migration of Relational Database to NoSQL Document-Oriented Database. *2023 IEEE International Conference on Data and Software Engineering (ICoDSE)*, 180–185. https://doi.org/10.1109/ICoDSE59534.2023.10291584

Unal, Y., & Oguztuzun, H. (2018). Migration of data from relational database to graph database. *Proceedings of the 8th International Conference on Information Systems and Technologies*, 1–5. https://doi.org/10.1145/3200842.3200852

Zaidi, N., Ishak, I., Sidi, F., & Suriani Affendey, L. (2022). An Efficient Schema Transformation Technique for Data Migration from Relational to Column-Oriented

Databases. *Computer Systems Science and Engineering*, *43*(3), 1175–1188. https://doi.org/10.32604/csse.2022.021969

ÖZGEÇMİŞ

Ad-Soyad **:** Muhammed Mehdi ELÖMER

ÖĞRENİM DURUMU:

- **Lisans** **:** 2017-2021, Fırat Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü
- **Yüksek Lisans** **:** 2021- Devam etmekte, İnönü Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Ana Bilim Dalı

MESLEKİ DENEYİM:

- 2021 - Devam ediyor – Bilgisayar Mühendisi.

YÜKSEK LİSANS TEZİNDEN TÜRETİLEN ÇALIŞMALAR

- **Deniz, A., Elömer, M. M, & Aydin, A.A. (2023).** A comparison of Apache Solr and Elasticsearch technologies in support of large-scale data analysis. Gümüşhane Üniversitesi Fen Bilimleri Dergisi, 2023.