T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

NOSQL VE RDBSM YAPILARIN PERFORMANS KARŞILAŞTIRMASI

Yüksek Lisans Tezi

Tezi Hazırlayan:

Nevzat Aydın BOLAT

İstanbul, 2019

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

NOSQL VE RDBSM YAPILARIN PERFORMANS KARŞILAŞTIRMASI

Yüksek Lisans Tezi

Tezi Hazırlayan:

Nevzat Aydın BOLAT

Öğrenci No:

1608200808

Danışman:

Dr. Öğr. Üyesi Turhan KARAGÜLER

İstanbul, 2019

YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum "YSDK ve RDBSM Yapıların Performans Karşılaştırması" başlıklı bu çalışmanın, bilimsel ahlak ve geleneklere uygun şekilde tarafımdan yazıldığını, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını belirtir ve bunu onurumla doğrularım. 29/01/2019

Nevzat Aydın BOLAT

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ SAVUNMA SINAVI SONUÇ TUTANAĞI

Beykent Üniversitesi
Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Aşağıda tez adı belirtilen yüksek lisans öğrencisi 160820808 no'lu **Nevzat Aydın BOLAT**'in 29/1/2019 tarihinde yapılan tez savunma sınavı[1] sonucunda..60.. dakika süreyle sunduğu ve savunduğu tezi hakkında[2] oybirliğiyle, KABUL.. kararı verilmiştir.

Bilgilerinize saygılarımızla arz ederiz.

Anabilim Dalı : .Bilgisayar Mühendisliği...
Programı : Bilgisayar Mühendisliğ....
Tez Başlığı[3] : NOSQL ve RDBMS YAPILARIN PERFORMANS KARŞILAŞTIRMASI

Tez Sınav Jürisi	**Öğretim Üyesi**	**İmza**
Danışman	: Dr. Öğr. Üye. Turhan ICARA GÜLER	
Üye	: Dr.Öğ.Üyesi Ediz ŞAYLICI	
Üye	: Prof. Dr. Gökhan Silahtaroğlu	

1 Jüri üyeleri, söz konusu tezin kendilerine teslim edildiği tarihten itibaren en geç bir ay içinde toplanarak öğrenciyi tez sınavına alır. Tez savunma sınav süresi en az 45, en çok 90 dakikadır. Jüri üyeleri, sınav öncesi yapılacak toplantıda, kendi aralarından danışman dışında bir üyeyi başkan seçer. Tez sınavı, tez çalışmasının sunulması ve bunu izleyen soru-cevap bölümünden oluşur. Tez sınavı, öğretim elemanları, lisansüstü öğrenciler ve alanın uzmanlarından oluşan dinleyicilerin katılımına açık ortamlarda gerçekleştirilir. Belirlenen günde yapılamayan jüri toplantısı, katılanların hazırladığı bir tutanakla enstitü yönetimine bildirilir. Bu durumda, jüri en geç on beş gün içinde toplanarak adayı tez savunma sınavına alır. (05 Ağustos 2017 tarihli 30145 sayılı Resmi Gazetede Yayınlanan Değişiklik-Madde 29-3)

2 Tez sınavının tamamlanmasından sonra jüri, tez hakkında salt çoğunlukla "kabul", "düzeltme" veya "ret" kararı verir. Jüri başkanı, jüri üyelerince imzalanmış karar tutanağını, tez sınavını izleyen üç gün içinde ilgili enstitü yönetimine teslim eder. Tezi hakkında düzeltme kararı verilen öğrenci en geç üç ay içinde gerekli düzeltmeleri yaparak ve birinci fıkradaki usule göre tezini aynı jüri önünde yeniden savunur. Süresi içerisinde "düzeltme" savunmasına girmeyen öğrencinin enstitü ile ilişiği kesilir. (Beykent Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliği-Madde 29-4)

3 İleride doğabilecek aksaklıkların engellenmesi için tezin başlığının yazılması gerekmektedir.

Adı ve Soyadı : Nevzat Aydın BOLAT
Danışmanı : Dr. Öğr. Üyesi Turhan KARAGÜLER
Türü ve Tarihi : Yüksek Lisans, 2019
Alanı : Bilgisayar Mühendisliği
Anahtar Sözcükler : Düzensiz Veri Tabanı, Düzenli Veri Tabanı, Yazma, Okuma, Güncelleme, Silme, Tablo, Tablo İlişkisi, Geliştirme, Kullanıcıya Sunma, Bakım

ÖZ

NOSQL VE RDBSM YAPILARIN PERFORMANS KARŞILAŞTIRMASI

Düzensiz ve düzenli veri tabanı yapıları karşılaştırmalı olarak incelenmiştir. Amaç olarak, veri tabanı uygulamaları farklı boyutlarda veri üzerinde okuma işlemi yaparken ne kadar verimli çalıştıklarının anlaşılması güdülmüştür. Aynı veriler hem düzensiz hem de düzenli uygulamalara yüklenmiş ve bir program aracılığıyla verimleri ölçülmüştür. Sonuçta çeşitli veri akış yükü ve veri boyutlarında hangi düzenin hem geliştirme hem de kullanıcıya sunmada daha verimli sonuç verdiği yorumlanmıştır. Veriler eşliğinde veri tabanlarının çok fazla verinin bulunmadığı, tablolar arasında çok fazla ilişkinin olmadığı kısımlarında düzenli yapıların, büyük tablolar ve karmaşık ilişkilerin bulunduğu kısımlarındaysa düzenli yapıların kullanılmasının geliştirme, kullanıcıya sunma ve bakımını yapma konularında daha verimli çözümler sunduğu anlaşılmıştır. Veri tabanı mühendislerinin ve veri tabanıyla uğraşan tüm yazılımcıların bir tasarıyı geliştirirken hangi yaklaşımı tercih etmeleri gerektiğine ışık tutulmaya çalışılmıştır.

Name and Surname : Nevzat Aydın BOLAT
Supervisor : Dr. Lecturer Turhan KARAGÜLER
Degree and Date : Master's Degree, 2019
Major : Computer Engineering
Key Words : Unstructured Database, Structured Database, Read, Table, Table Relation, Development, Serving, Maintaining

ABSTRACT

PERFORMANCE COMPARISON OF NOSQL AND RDBSM SYSTEMS

Relational and non-relational database structures are compared. The aim is performance comparison while reading different sizes of data. The same data was loaded to relational and non-relational systems and measured via a program. In the end it is interpreted that which system handles which data size and process load better in serving data to consumers. In the light of this information it is concluded that non-relational system performs better for databases with less count or complexity of queries while relational system performs better if the database has more count or complexity of queries. This paper aims to guide database engineers and developers that work on databases, which systems are better in specific situations.

İÇİNDEKİLER

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

KISALTMALAR ve TERİMLER

ACID	: Tekillik (atomicity), tutarlılık (consistency), yalıtılmışlık (isolation), değişmezlik (durability)
Ağ	: Web
Başvuru	: Başvuru (Reference)
Belge	: Doküman (Document)
Değişken	: Parametre (Parameter)
Dizin	: İndeks (Index)
Düzen	: Sistem (System)
Gözde	: Popüler (Popular)
İVTYD	: İlişkili Veri Tabanı Yönetim Düzeni(RDBMS)
JSON	: JavaScript Object Node
Karmaşık	: Kompleks (Complex)
MSSQL	: Microsoft Structured Query Language
NoSQL	: Not Ordered Structural Query Language
Ortam	: Platform (Platform)
Önbellek	: RAM
RAM	: Random Access Memory (Donanımsal ön bellek)
RDBMS	: Relational Data Base Management System
Tasarı	: Proje (Project)
TSQL	: Transact Structured Query Language
Uygunlaştırma	: Optimizasyon (Optimization)
Veri Tabanı Bölme	: Sharding
Verim	: Performans (Performance)
XML	: Markup Language
Yapılandırılmış	: Organize (Organized)
YSD	: Yapılandırılmış Sorgulama Dili (SQL)
YSDK	: Yapılandırılmış Sorgulama Dili Kullanmayan (SQLK)
Zaman Damgası	: Time Stamp

1. BÖLÜM

GİRİŞ

İnsanlık yazılı tarihin başlangıcından beri bilgileri yapılandırılmış bir şekilde tutmak için yeni yöntem arayış biçimleri geliştirmiştir. Anıtlar, tabletler, parşömenler, el yazması kitaplar, matbaa kitapları, not defterleri, her zaman için bilgiye birden çok kişinin kolayca ulaşmasını sağlayacak şekilde hazırlanmıştır. Günümüz elektronik ortamları da aynı amacı gözeterek gelişmektedir. Tüm bu bilgilerin saklanmasında genel olarak iki yaklaşım vardır. Tezin konusu verilerin elektronik ortamda saklanması olmasına karşın, geleneksel veri saklama yollarından biri olan kitaplar ile arasındaki benzerlik çok yakındır. YSD yöntemi, 'A. Turing'in bu konudaki düşüncelerine ilgili kitabından ulaşabilirsiniz' derken, YSDK yöntemi A. Turing'in ilgili düşüncelerini aynı kitabın içine yazmasına benzemektedir; aradığınız bilgiye hızlıca ulaşabileceksinizdir. Ama bu, sonuçta kitap başına çok daha fazla sayfanın basılması anlamına gelecektir. Bu yüzden, verileri saklamak için bu iki yöntemden birini seçmeden önce kâr - zarar durumunun incelenmesi gerekir. Kurulacak bir düzeni önceden düşünmek, düzenin belli bir zaman sonra ne boyutlara geleceğini, nasıl bir karmaşıklığa sahip olacağını, ne kadar bakılacağını/sorgulanacağını öngörmek hem geliştirme masraflarının azalmasına hem de daha sonradan oluşabilecek bir düzen değişiminin (İVTYD– YSDK geçişinin) önüne geçecektir. YSD bir düzende yönetilemez boyutlarda veriyle uğraşmak, ya da çok karmaşık ve fazla verinin bulunmadığı bir düzende YSDK çözümüyle geliştirme masraflarını yüksek boyutlara çıkarmak istenmeyen ve daha sonra değiştirilmesi zor bir durumdur.

İşte bu tezin konusu, bu iki yöntemin yetkinliklerini sınamak ve hangi sınırlar içerisinde kullanılmasının daha verimli olduğunu anlamaya çalışmaktır. Böylece bu tezi inceleyen yazılımcılar, veri tabanı geliştiriciler ve bilişim teknolojilerinde çalışan her türden insan, seçimlerini yaparken başvurabilecekleri bilgiler bulabilecektirler.

YSDK veri tabanı tipleri çok fazla ve karmaşık olmasına rağmen bu tezde piyasanın mali, kullanıcıya özel ve basit veri tutma işlemleri için elverişli olan ve piyasada bolca yer bulan ‘belge tipi YSDK’ çözümü, çeşitli değişkenler karşısında sınanacaktır [1][s8]. Aynı değişken ve karşılığı veriler, aynı şekilde YSD tasarısında (MSSQL kullanılacaktır) da sınanıp veriler karşılaştırılacak ve bir sonuca varılacaktır. İki veri kümesi ve iki ortam kullanılarak büyük ve küçük firmaların durumları taklit edilmeye çalışılacaktır. Belge tipi YSDK çözümü için kullanılacak ürün ElasticSearch adlı bir Java uygulamasıdır. Bu uygulama, tüm verileri belgelere JSON olarak kaydedip, bünyesindeki güçlü bir arama motoruyla sorgulayabilmektedir [2][s2]. Tüm çalışma yalnız sanal makine üzerinde gerçekleştirilecektir.

Tezde elektronik veri depolama yöntemlerinin kısa tarihçesi verilecektir. YSD ve YSDK yöntemleri incelenecektir. Karşılaştırma, günümüz piyasasında belirli ve sınırlı bir yere denk gelmektedir. İşte bu yer incelenecek, kurum ve kuruluşların hangi durumlarda ne kadar ve ne karmaşıklıkta veri barındırdıklarına değinilecektir. Sonraki aşamada örnek bir veri tabanı hem YSD hem de belge tipi YSDK üzerinde oluşturulacak, MSSQL ile YSD veri tabanı, .net C#’ta yazdığım uygulama ile de ElasticSearch üzerindeki eşdeğeri çeşitli karmaşıklık ve boyutlar karşısında ölçülecek, grafiğe dökülecek ve yorumlanacaktır.

Karmaşıklığın, birbirine bağlı tablo sayısının az olduğu ve hızlı artmayan/durağan veri boyutları durumlarında YSD bir çözüm geliştirmenin hem daha YSDK çözümden ucuz, hem de donanımın (sabit disk, bellek ve işlemci bakımlarından) YSDK çözüme oranla anlamlı bir farkının olmadığını teyit etmek ve bunun tersi durumlarda da YSDK’nin daha kabul edilebilir bir çözüm olduğunu ortaya çıkartmak, hipotezin bir parçasıdır.

1.1 Konu Üzerine Yapılmış Çalışmalar

Bu bölümde, bu tezin konusunu içeren, daha önce yapılmış tezler, akademik makaleler ve diğer araştırmalar incelenecektir.

1- 2013 yılında yapılan bir çalışma yaygın olarak kullanılan YSDK çözümlerinden olan MongoDB ile yine yurt dışında yaygın kullanılan MySQL arasında bir verim karşılaştırması sunmuştur. Küçük işlem hacminde MySQL daha güçlü bulunmuşken, yüksek işlem hacimlerinde MongoDB'nin daha hızlı sonuç döndüreceği öngörülmüştür. Ülkemizdeki halka açık en büyük veri tabanlarından olan Kariyer.net dâhil birçok firma MSSQL kullanmaktadır. Daha küçük firmalar yönetim kolaylığı, servis ve destek, ileri seviye uzman gerektirmemesi gibi nedenler yüzünden Microsoft ürünlerini daha da çok tercih etmektedir. Çalışmamda da MSSQL'i tercih etmemin sebebi budur.

Çalışma karmaşık sorgularda MongoDB'yi daha verimli bulmuştur. Bunun nedenini de daha fazla disk alanı kullanarak, veri tabanı şemasını daha sade tutmasına bağlamıştır. Ancak bunun verinin çoklanması ve yönetiminin zorlaşmasını da beraberinde getirdiğini vurgulamıştır [1].

2- 2014 yılında yapılan bir konferansta sunulan veriler, yüksek hacimli çalışmaların (100.000 satır ve üzeri) MongoDB'de, Oracle'a göre çok daha hızlı gerçekleştiği yönündedir. Ancak bunlar firmaların raporlama gibi işlemlerde kullandıkları ve sıkça gerçekleşmeyen veri girişi ve güncellemeye yönelik işlemlerin sonuçlarıdır. Çalışmamda ise son kullanıcıların sorgularına verilen cevapların süresi ölçülmüştür [2].

3- 2013 yılında yapılan bir çalışmada MongoDB ile MySQL karşılaştırılmıştır. MongoDB yazma, güncelleme ve basit sorgularda daha iyi bir verim sergilerken, MySQL anahtar değer içermeyen durumlarda güncellemeyi ve sorgulamayı daha verimli olarak bitirmiştir. MySQL aynı zamanda karmaşık sorgularda da daha verimli olarak gözlemlenmiştir.

MongoDB şeması sürekli değişen veri tabanları için daha iyi bir çözüm olarak görülmüştür. Ancak YSD çözümlerin standart olması, çözümlerin daha kolay bulunacağı anlamına da gelmektedir. MongoDB'nin karmaşık sorgulardaki veriminin veri tabanının yatay genişlemesi ile iyileştirilebileceği de son sözler arasındadır [3].

1.2 ElasticSearch'ün Kullanıldığı Firmalar

28.11.2018 tarihi itibariyle ElasticSearch'ün sitesinde 163 büyük firmanın [4], ElasticSearch'ü düzenlerinin bir kısmında kullandığı bildirilmiştir.

Firmaların birçoğu çalışanlar için toplanan verilerin analizi, müşteriler için veri tabanı taraması (müzik, kişi, yazılı basın, gibi, kategorilenebilen şeyleri aramak için), düzen geçmiş analizi, müşteri verilerinin analizi ile düzen iyileştirmesi, aramayı basitleştirerek maliyetleri düşürmek, iç ağ ve güvenlik takibi, alışveriş, mobil oyun veri tabanı, kullanıcı verilerini raporlama ve hata analizi, bilimsel analizler, geçmişe dönük arşivleme-arşivden çıkarma, kullanıcı davranışlarını takip etme, büyüyen veriyi işleyebilme, makine öğrenme, web sitelerine çoklu dil desteği sağlama, yazılım hata ayıklama, internet bağlantılarını denetleme, belgeleme indeksleme gibi işler için ElasticSearch'ten faydalanmaktadır.

Microsoft, Adobe, Mozilla, Cisco, Tinder, Über, Ebay, Nvidia, Slack, BBC, Walmart, Facebook, Netflix, Symantec, Verizon, Warner Brothers, Ebay, Blizzard, Dell, Volkswagen, Godaddy, SAP Concur, Airbus, ElasticSearch'ü kullanan bâzı ünlü firmalardır.

Görüldüğü üzere ElasticSearch birçok alanda, önemli firmalarca kullanılmaktadır. Ancak küçük ve orta boy firmalar da müşterilerini yönetebilmek için ElasticSearch'ten faydalanabilecekler midir? Çalışmanın devamında bu konu araştırılmıştır.

1.3 Platformların Seçilme Nedenleri

İlerleyen bölümlerde belirtileceği gibi birçok YSDK tipi, bu tiplerde üretilmiş YSDK programları ve İVTYD programları bulunmaktadır. Peki, bu çalışmada YSDK tiplerinden belge veri tabanı, belge veri tabanı programlarından ElasticSearch ve İVTYD'lerden MSSQL seçilmesinin nedenleri nelerdir?

1.3.1 Doküman Veri Tabanı

YSDK terimi bir çatı terimdir. Bu çatının altında bulunan dört başlığın birbirleriyle olan ilişkisi YSDK, yani yapılandırılmamış olmalarıdır. Bunun anlamı; verimde ve geliştirmede büyük kayıplar vermeden birbirlerinin yerine kullanılamayacak olmalarıdır, çünkü veri depolama ve bu verileri geri getirme yöntemleri birbirlerinden büyük ölçüde farklıdır [1][s9]. Bu çalışmada ölçülmek istenen küçük ve orta boy firmaların müşterilerine sunduğu satış işlemlerinde kullanılabilecek en mantıklı tip ise belge veri tabanıdır. Bunun nedeni ilerleyen bölümlerde bu veri tabanlarının özelliklerini incelerken açığa çıkacaktır.

1.3.2 ElasticSearch

ElasticSearch dünya çapında kullanılan yetkin bir üründür. Bu ürünü kullanan firmalar ürünün yetkinliği konusunda fikir sahibi olmamızı sağlamaktadır.

1.3.3 MSSQL

StackOveflow'da yapılan ankete göre MSSQL (YSD Server) %41'lik arama [5] ile ikinci sırada bulunmaktadır. Firmaların kullandıkları veri tabanlarını öğrenmek her zaman mümkün değildir, ancak veri tabanı üzerinde geliştirme yapan yazılımcıların başvurularını sayıya dökmek (StackOverflow anketi ve arama sonuçları gibi) bize genel durumu gösterebilmektedir. MSSQL'de daha deneyimli olmam ve geliştirmenin kolay olması, bu ürünün seçilmesinde rol oynamıştır.

2. BÖLÜM

YSD VE NOSQL VERİ TABANI TİPLERİ

2.1 Veri Tabanı ve Veri Depolama Türüne Göre Veri Tabanı Tipleri

Veri tabanları, içlerinde birbiriyle doğrudan ya da dolaylı olarak ilişkili olan veriler bütündür [8][s4]. Bu veriler tek bir tür bilgi sağlayabileceği gibi, örneğin; günlük satış rakamları, her bir maldan depoda ne kadar kaldığı gibi, toplu ve belli türlerde bir bilgi yığını barındıran, örneğin; videolar, Excel halinde tutulan özlük bilgileri, yemek kartı hesapları, mal türleri de olabilir.

Veri tabanlarının içindeki veriler sürekli bir devinim halindedir. Sürekli olarak yeni bilgiler eklenir, eski bilgiler çok nadiren değişebilir, veri sayısı aynı kalmasına rağmen var olan veriler belli aralıklarda yeniden düzenlenebilir. Ve tüm bunların oluş sıklığı, verinin türüne, kullanıldığı firmaya, eğitim kurumuna, bilimsel çalışma yapılan kuruluşa göre değişiklik gösterir. Bu tip özel ve devlete bağlı kuruluşların zaman içerisinde biriktirecekleri veri çok fazla olacaktır. İlerleyen zamanlarda ihtiyaçların değişimi ve artışı da göz önüne alınmalıdır. Bu yüzden, bir kuruluşa ait veri tabanı tasarlanırken yalnızca elde bulunan, aktarılacak verilere bakılmamalıdır. Kuruluşun hedeflerine bakılarak, bu hedeflerin gerçekleştirileceği planlanan zaman sürecinde biriktirecekleri veri ve bu verinin ne kadar karmaşıklaşacağı göz önüne alınmalıdır [6]. Bu araştırma ve planlama yapılmadığı durumda, ileride çok zahmetli ve pahalı olan veri tabanı iyileştirmeleri, örneğin; YSD bir düzende işlemcilerin ve belleklerin maliyet yaratacak ve bir süre sonra da teknolojik sınıra dayanarak sürdürülebilirliğini yitirmesi, ya da sürekli değişen küçük miktardaki bir verinin bakımını ve yürütmesini yapmak için çok daha fazla adam/gün maliyeti oluşturması söz konusu olacaktır.

Bir veri tabanı oluşturulurken aşağıdaki konulara dikkat edilmelidir.

Veri tabanı, kuruluşun tüm verilerini tutabilmelidir. İlk başta bu şekilde görünmese bile, ilerleyen zamanlarda tüm veriler birbirlerine dolaylı olarak bağlanacaktır.

Verilere kuruluş içerisinden herkesin erişimi sağlanmalı, kuruluş dışındansa güvenli bir şekilde yalnızca güvenliği tehlikeye düşürmeyecek kısımlarının çıkışı için hazırlık yapılmalıdır.

Veriler mümkün olduğunca tekrar etmemelidir. Bunu yapmanın yolu merkezi bir düzen kurmaktır. Dışarıya açılacak veriler, yalnızca bu verileri barındıran sunuculara kopyalanabilir, elektronik satış, özgeçmiş filtreleme gibi karmaşık ilişkili ve yüksek boyutlu verilerin işlendiği durumlarda YSDK yöntem kullanılması ve verinin tekrarlanması kaçınılmazdır.

Tablolar oluşturulurken yapısal sorunlarla karşılaşılmamalı ve ACID kuralına uyulmalıdır.

Yedekleme, yedekten döndürme, güvenlik duvarı, anti virüs yazılımları gibi yöntemlerle veri güvenliğini sağlanmalıdır.

Veri kontrol edilebilir, şeffaf olmalıdır.

Veri tabanları ilişkisel tablo düzeni ve ilişkisiz tablo düzeni olarak ikiye ayrılır. Bu bölümde bu terimlerin açıklanması, karşılaştırmanın anlaşılır olması için gereklidir.

2.2 İlişkisel Veri Tabanı (İVTYD)

İlişkisel veri tabanlarının tercih edilmesinin en büyük nedeni her bir parça verinin yalnızca bir yerde tutulmasıdır. Bu şekilde sorgunun sonunda dönecek verinin güncel ve tutarlı olduğundan emin olunabilir.

Veriler tablolarda saklanır. Bu tablolar içlerindeki verilerin türlerine göre oluşturulurlar. Bu tipteki her bir veri parçasının belirlenmesi, bulunması için bir kimlik numarası vardır. Her bir satırın kendi bir ve yalnızca bir kimlik numarası olabilir. Ancak bu satırda, ilişkileneceği diğer satırların kimlik numaraları bulunabilir.

Bu şekilde ilişkilenen satırlar - en küçük veri parçaları - kimlik numaraları sayesinde merkezi veri tabanı düzeninin her yerinden çağırılabilirler. İlgili veri başka bir noktada tutulmadığı için çekilen veri günceldir.

İlişkisel veri tabanlarının en dikkat edilmesi gereken yeri veri tabanı şemalarıdır. Her bir veri bir kez kullanılacak şekilde geliştirilen bu şema, eklenen her bir yeni veri kümesi yüzünden yeniden ayarlanmaya ihtiyaç duyar.

Ancak bu verilere ulaşabilmek için diğer tablolar üzerinden kimlik numaralarıyla gidildiği için her sorguda aynı hıza ulaşılamaz. Yapılacak sorgu birçok tablodan veri çekilmesini gerektiriyorsa, ya da bir veriye ulaşmak için birçok tablodan ilişkisel bağlantı yoluyla geçilmesi gerekiyorsa, o sorgu hem zaman hem de işlem bakımından masraflı olacaktır.

Özellikle Türkiye'de, internet kullanımının yaygınlaşmasından önce ve kullanılan internetin küçük hacimli olduğu 2000'li yılların başında veri trafiğinin az olması, birçok firmayı verim açısından zorlamıyordu. Büyük firmaların kendi iç işlerini ve müşterileri, tedarikçileriyle olan ilişkilerini elektronik ortama aktarmaktaki zorluklar, büyük firmaların elektronik ortama geçişlerini yavaşlattı. Ancak müşteri sayısı az olan firmalar dış işlerinde ilişkisel veri tabanlarını kullanabilmeye devam etmektedir.

İleri okuma için Codd kuralları incelenebilir.

2.2.1 MSSQL

Microsoft firmasına ait veri tabanıdır. Yönetimi kolay olduğu ve güçlü bir kullanıcı ara yüzü sunduğu için birçok firma tarafından tercih edilmektedir.

2.2.2 ORACLE

SUN firmasına ait veri tabanıdır. Yönetimi daha çok komut satırlarına dayandığı için uzman kullanıcılara ihtiyaç duyar. Ancak veri uygunlaştırması ön sıradadır.

2.2.3 MYSQL

Oracle firmasına ait veri tabanıdır. Açık kaynaktır, bu yüzden yapısı değiştirilebilir. Bu özelliği Facebook gibi özelleşmeye ihtiyaç duyan firmalar için gözdelik kazandırmıştır.

2.3 İlişkisiz Veri Tabanları (YSDK)

İlişkisiz veri tabanlarında veri bölümleri arasında bir ilişki bulunmaz. Bunun sonucu olarak da veriler farklı veri bölümlerinde yinelenirler. Bu durum veri tutarlılığını azaltmaktadır. Çünkü aynı verinin tüm veri tabanı üzerinde, saklandığı her yerde değiştirilmesi gerekmektedir. Ancak diğer tablolar arasında kimlik numaraları yoluyla ilişkisel bir şekilde çekilme gibi bir zorluk olmadığı için, sonuçlar daha hızlı oluşur. İlişkisiz veri tabanlarının tarihi de ilişkisel veri tabanlarınınkiler kadar eskidir (1965 PICK), ancak çok büyük veri kümeleriyle uğraşma gereği yakın zamanlara kadar oluşmadığı için çok yaygın değildi.

İlişkisiz veri tabanlarında kural olarak, ilişkisel veri tabanlarının tersine mümkün olan en büyük veri kümeleri oluşturulur [7]. Bu şekilde sonucu büyük olacak bir sorgu tüm merkezi veri tabanından veriler çekmek yerine tek bir veri kümesinden basit bir indeks aramasında bulunur.

YSDK düzenler

Basit işlemleri, verisi yatay olarak sunuculara dağıtılmış şekilde yapabilmelidir.

Veriyi birçok sunucu üzerinde çoklamalı ve dağıtmalıdır.

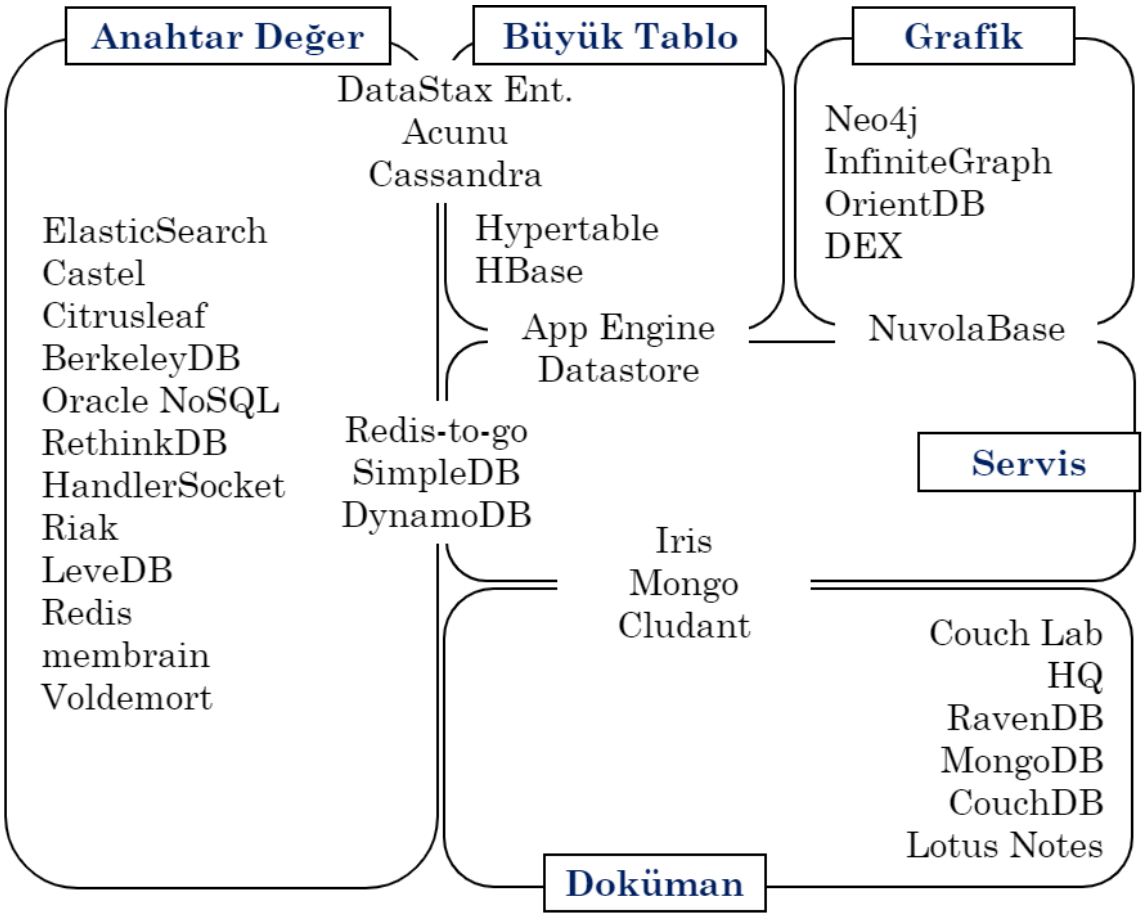
Veriyi basit bir arayüzden alabilmelidir (JSON çağırısı gibi)

ACID'e sıkı bağlı değildir

Veriyi mümkün olduğunca RAM'den okumalıdır.

Veri tabanına yeni bilgi sütun/bölümlerini bağımsız olarak atayabilmelidir.

Şekil 1. NoSQL Türleri



Zaman içerisinde ilişkisiz veri tabanlarının birçok türü yapılmıştır. Her biri farklı ihtiyaçlara cevap veren bu türleri inceleyelim.

2.3.1 Büyük Tablo (Big Table, Columnar ya da Wide Column Store)

Karmaşık sorguların gerekmediği durumlarda tercih edilebilir. Veri tabanı tek bir tablodan oluşur ve her bir satırın yine tek bir kimlik numarası bulunmaktadır. Sütun sayıları milyarları bulabilir. Bu şekilde, herhangi bir veri kümesinin atılda bekleyen birçok

boş sütunu bulunur, bu sütunu istenildiği zaman doldurularak zahmetsiz ve az uğraş gerektirecek bir şekilde veri tabanı genişletilebilir; ilişkilendirilebilir.

Çekilen herhangi bir satırın hangi veri kümesine ait olduğu, o satırın hangi sütunlarının dolu olduğuyla anlaşılır. Column: Dikeç, Super Column: tablo, Super Column Familie: ilişkilendirilmiş tablolara karşılık gelmektedir.

Yukarıda anlatılanlar bu veri tabanlarının iki boyutudur. Ancak zaman dikeci sayesinde bu veri tabanı türü üç boyutlu olmaktadır. Bir satırda yapılan her değişiklikte, satır kopyalanır ve bir zaman damgası (timestamp) alır [8][205].

En uygun olduğu yerler büyük veri kümelerinin analizidir.

Petabyte seviyelerinde çalışmak üzere tasarlanmıştır, yüzlerce ya da binlerce makine bu veriler üzerinde çalışır ve yeniden ayarlama gerekmeksizin yüzlercesi daha bu düzene katılabilir.

Veri her bir limite geldiğinde (örneğin 200mb) verimli bir şekilde sıkıştırılır ve içeriği arşivlenir. İhtiyaç duyulduğunda hızlıca açılarak içeriğine ulaşılır. Bu yüzden verileri satır satır değil, bölüm bölüm tuttukları söylenebilir.

Ürünler: Bigtable, Apache Cassandra, Apache HBase, Apache Accumulo, Hypertable, ScyllaDB, Sqrrl

2.3.2 Anahtar - Değer (Key – Value Store)

ID ve tek sütunda verisi olarak kayıt tutulmaktadır. Veri hücresi incelenmeden içinde ne tutulduğu söylenemez. Farklı dikeç ve veri türleri kullanmadığı için genellikle çok daha az disk alanı kaplar. Herhangi bir şema kullanılmaz.

Verilere ulaşmadaki tek kıstas kimlik numarası/yazısıysa ve karmaşık sorgular atılması gerekmiyorsa anahtar-değer veri tabanları kullanılabilir. [6][p4]

Ürünler: Redis, Voldemort (LinkedIn), Dynamo (Amazon)

2.3.3 Grafik

Her türlü verinin birbiriyle nesne, ilişki, özellik ve etiketler ile ilişkilendirildiği veri tabanı tipidir. Ortaya çıkan veriler YSDK yapısıyla kaydedilir. Bu yolla, her seferinde karmaşık sorgular atılmadan ulaşılamayacak sonuçlar, hızlıca elde edilir [9][s7]. Facebook gibi arkadaşının arkadaşlarının ilgilendiği videoların, grupların ve beğenilerinin gösterildiği ortamlar bu şekilde çalışmaktadır.

Kullanılan yerler:
-Öneriler: bakılan eşya, kişi, iş vb. ile ilgili olan diğer bilgileri de edinme.
-Veri yönetimi: Bölge, çalışan, yönetici, ürün, satış gibi birbiriyle alakalı veriler eşleştirilir, bir sorun ortaya çıktığında, etkilenen birimlere ve ilişkilere bakarak sorunun kaynağı ortaya çıkartılır.
-Sahtecilik: Kişilerin bilgilerini ve aralarındaki ilişkiyi araştırarak işlem anında şüpheli hareketler belirlenir.

-Arama: Herhangi bir konuda arama yaparken ilişkili ve kullanıcıya yardımcı olacak diğer bilgiler de gösterilir.
-Yetkilendirme: Çalışanlar ile donanım arasındaki ilişkiyi kolayca kırıp, yaratma. [6][p7]

Ürünler: Neo4j, InfiniteGraph, OrientDB, DEX.

2.3.4 Doküman Veri Tabanı

Yukarıdaki RDBSM ve YSDK çözümlerinin aksine, belge veri tabanları tablolar, satırlar ve dikeçler içermez. Sunucular içerisinde gruplar halinde belgeler oluşturulur. Tüm veriler bu belgeler içerisinde tutulur. Kurulacak düzene göre farklı veri depolama yöntemi seçilebileceği gibi, aynı düzen içerisinde de farklı yöntemler seçilebilir. Bu yöntemlere örnek olarak JSOM, XML, CSV ya da ofis belgeleri verilebilir.

Belge veri tabanının gücü güçlü indekslemeden gelmektedir. Bir XML'in alt birimlerindeki gibi, aynı veri kümesinin içinde aynı veriye ait birden fazla kısım bulunabilir [9][s5]; örneğin bir çalışana ait telefon numaraları aynı XML kısmı içerisinde saklanabilir.

Yöntemin dikkat edilmesi gereken yerleri ise; indeks sayısının artmasının hızı düşürebileceği ve çok fazla disk alanına ihtiyaç duymasıdır.

Ürünler: Couch Lab, HQ, RavenDB, MongoDB, CouchDB, Lotus Notes.

3. BÖLÜM

YSD VE NOSQL KOD VE PLANLAMA FARKLILIKLARI

Sistemin verimi ve donanımsal farklılıkların yanında işlemesi için yazılım da gereklidir. İVTYD ve YSDK yazılımları verileri oldukça farklı biçimde işlemektedir.

İVTYD kodları doğrudan sunucuya yöneliktir ve YSD adı verilen (MSSQL'de TSQL) benzer kod yapılarını kullanırlar. Bu tip kodlar İVTYD'ler arasında farklılaşsa da oldukça benzerdirler ve bir kısmı hemen her yazılımcı tarafından bilinmektedir.

YSDK kodlarıysa İVTYD kodlarından farklıdırlar. Web üzerinden çalıştıkları için JSON sorgu tipini benimsemişlerdir. Hataya açık ve geliştirmesi zor bir platformdur. Ancak bunun yanında ElasticSearch'ü C# üzerinde nesne tabanlı yazmayı sağlayan NEST gibi kütüphaneler de bulunmaktadır. Geliştiricilerin işini kolaylaştırsa da bu da bir öğrenim eğrisine neden olmakta ve belli bir miktar uygulama yavaşlamasını da beraberinde getirmektedir. Verim kaybı yüzünden bu çalışmada NEST kullanılmamıştır.

3.1 Veri Tabanı Çoklama ve Bölme

Veri tabanı çoklama, her sunucunun tüm verilerin bir kopyasını taşımasıdır. Uygulamalar bu sunucuların farklı bölümlerini tarayarak istenilen sonuçları daha hızlı getirebilirler. Veri tabanı bölme işleminden daha fazla sabit disk alanı istemesine karşın veri bütünlüğünü sağlamak açısından daha başarılıdır.

Veri tabanı bölme, her sunucunun tüm verilerin bir parçasını taşımasıdır [10]. Veri kayıpları, herhangi bir sunucunun durması durumunda verilerin bir kısmına erişememek

gibi sorunları olmasına karşın çok daha az sabit disk alanına ihtiyaç duyarak masrafları düşürür.

YSDK ve ilişkisel veri tabanlarında bu iki tip yöntem kullanılabilmektedir. Bu tezde veri tabanları yalnızca tek sunucu üzerinde kurulmuştur.

3.2 ACID

Hız ve verimlilik her ne kadar çok önemli olsa da eğer verilerin doğruluğundan emin olunamazsa işe yaramaz. ACID ise bu kesinliği sağlayan unsurların kısaltmasıdır. Sırasıyla; tekillik (atomicity), tutarlılık (consistency), yalıtılmışlık (isolation), değişmezlik (durability).

Tekillik: İşlem ya tamamıyla yapılmalı ya da hiç yapılmamalıdır. MSSQL gibi RDBSM yapılarda tekilliği korumak düzenin bir parçasıdır. Eğer işlem bir noktada kesintiye uğrarsa, hata alınırsa, düzen tüm işlemleri geri alarak veriyi önceki konumuna çeker. ElasticSearch gibi YSDK yapılarda eğer birden fazla tabloya / indekse veri girilmesi gerekiyorsa, bir tabloya yazdırma sırasında alınan hata işlemi durdurmamakta ve geri döndürmemektedir [11]. Bu iş yazılımcıya kalmıştır. Bu yüzden YSDK yapılarda tekilliği sağlamak daha zordur.

Tutarlılık: Program veri tabanını her zaman için doğru bir şekilde algılamalı ve işlemleri bitirdiğinde veri tabanını doğru verilere güncellemelidir.

Yalıtılmışlık: Program kullandığı tüm verileri, veri tabanında sanki başka bir işlem gerçekleşmiyormuş gibi algılamalıdır. YSDK veri tabanları da bunu sağlamaktadır ancak veri tabanındaki çoklanmış verilerin tamamını bulup düzeltmek zaman almaktadır. MongoDB gibi birçok YSDK çözümü, yeni verileri saklayıp, belli aralıklarla yazdırma yoluna gitmektedir.

Değişmezlik: Tüm işlem tamamlandıktan sonra veri tabanı üzerinde kalıcı olmalıdır. RDBSM'nin tekillik ve yedekleme teknolojileri bu maddeyi sağlar [8][s754].

Yeni bir veri tabanı kurulacakken, ya da var olan bir veri tabanı kısmen YSDK düzene geçirilecekken yukarıdaki maddeler dikkate alınmalıdır.

3.3 Veri Tabanlarını Kıyaslama

İlişkisel ve ilişkisiz veri tabanları için seçtiğimiz MSSQL ve ElasticSearch'ü Windows YSD Server işletim düzeninde VMWare sanal makinesinde çeşitli biçimlerde karşılaştıracağız.

Testler iki bölümden oluşmuştur:

1) 10.000.000 veri ve güçsüz bir düzen kullanılan bölüm, çok büyük verilere sahip olan ya da sınırlı kaynağı olan bir firmayı temsil etmektedir.
2) 100.000 veri ve güçlü bir düzen kullanan bölüm, görece az ya da geniş kaynağı olan bir firmayı temsil etmektedir.

ElasticSearch ve MSSQL'i eşit şartlarda kıyaslamanın bir parçası olarak asp.net uygulaması kullanılmıştır. ElasticSearch'ün JSON ve MSSQL'in TSQL sorguları bu uygulama üzerinden çağırılmıştır.

Gerçek hayat uygulaması olarak

- her bir sorgu teker teker çağırılmıştır (bulk yapılmamış)
- sorgular WCF ile çağırılmıştır
- her sorgunun değişkenleri rastgeledir, hiçbir sorgunun değişkenleri aynı değildir.
- sorgular 1 (tek cevabı olan sorgular), 10, 20 ve 30 (sayfalama) sonuç olarak dörder tipte sonuç döndürmüştür
- bir anda 1, 10 ve 100'erli sorgu atacak tipte denemiştir
- MSSQL'de sorgu atılacak tüm sütunlar indekslenmiştir

YSDK veri tabanı uygulamaları, aranan verileri önbelleğe alarak sonuçları mümkün olan en kısa sürede döndürürler. Bunun için çokça RAM kullanırlar. Ancak gerçek kullanım bu şekilde olmayabilir. Farklı kullanıcıların farklı istekleri yüzünden aynı sorgu art arda çağırılmayabilir. Yapılan denemede bu durum gözetilerek her sorguda farklı değerler gönderilmiş ve önbellekleme en alt seviyede tutulmuştur.

Veri tabanlarında önbelleğe alma önemli bir yer tutar. Bir sorgu sonucunda dönen veriler kaydedilir ve aynı sorgu bir daha çağırılırsa bu sonuçlar hızlıca döner. Ancak bir internet sitesinin kullanıcıları, geniş seçenek yelpazesinden çok farklı aramalar yapmaktadır. Bunu benzeştirmek için bütün sorgularda rastgele sorgu verileri kullandım. Sorgunun biçimi aynı kalmasına rağmen değişkenler her sorgu için değişti. Bu da gerçeğe daha yakın bir durum yarattı.

ElasticSearch bir veri belgesine (node) bir anda bir sorgu atabilmektedir. Buna karşın YSD veri tabanları aynı anda birden çok işlemi gerçekleştirebilecek şekilde geliştirilmektedirler. Bu yüzden aynı sunucu üzerinde aynı veri tabanını sorgulayacak YSDK veri belgelerinin çalıştırılmasıyla aynı anda birden fazla sorgu yapılabilir.

3.4 Küçük Sistemler

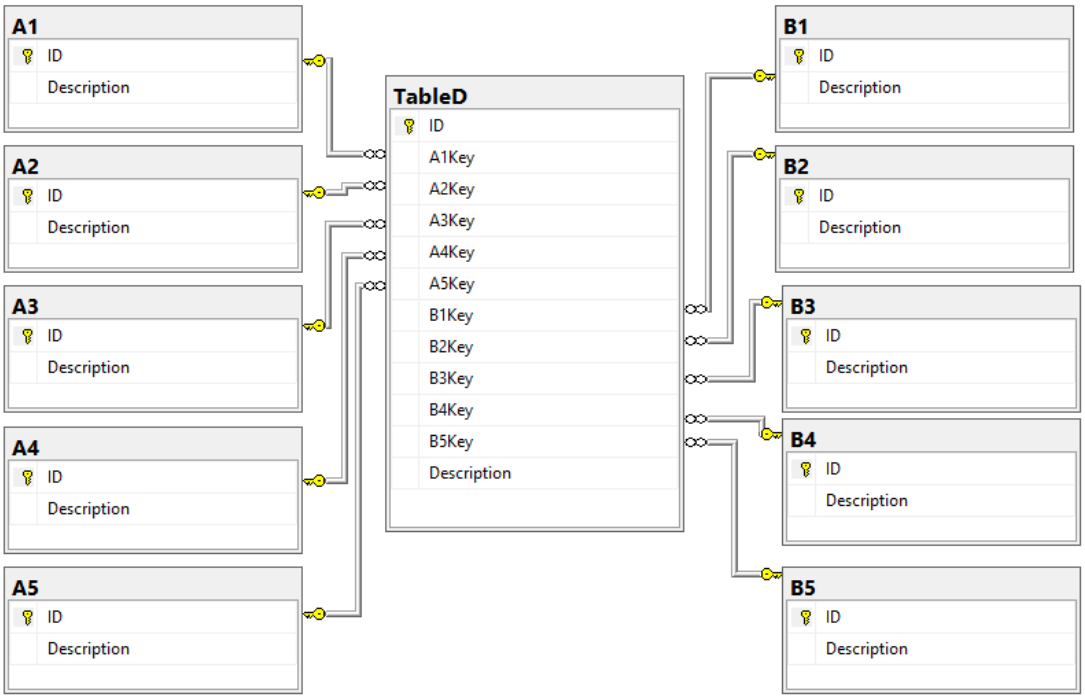
Kullandığımız makine 6 çekirdekli Intel i7 ve 6gb DDR3 RAM'e sahiptir.

Veri tabanı ElasticSearch'te 10.000.000 satıra karşılık gelmektedir

Ancak burada karşılaşılan bir sorun; ElasticSearch'ün MSSQL'e göre çok daha fazla işlemci ve bellek kullanmasıdır. Bu yüzden aynı sunucu üzerinde birden fazla veri belgesi çalıştırmak, geçici belleği hızlıca kaplamakta, işlemciyi ise sorgu anında %100'e getirmekte, bu şekilde de verimi düşürmektedir.

Bu yüzden veri tabanını sunuculara bölmek iyi bir yöntemdir. Veri tabanı her bir sunucunun bir parçasını tutarak, tuttuğu parça üzerinde işlem yapar. Bu da daha az verinin daha çok işlemci tarafından sorgulanmasını sağlar. Veri tabanını sunuculara bölmek hem YSDK hem de YSD üzerinde mümkündür. Bu çalışmada yalnızca bir sunucu ancak ElasticSearch için birden fazla veri tabanı tarayıcısı kullanılmıştır.

Şekil 2. Küçük Sistem SQL Şeması



3.4.1 Basit Sorgu

İlk sorgum yalnızca iki değerden birinin doğru olmasının sağlandığı bir sorgu. Bu basit sorgunun amacı kullanıcının

- Bir, iki müzik türü
- Sitedeki kişisel bilgiler
- Oyundaki karakter hakkındaki bilgiler
- CV bilgileri

gibi verileri getirmektir.

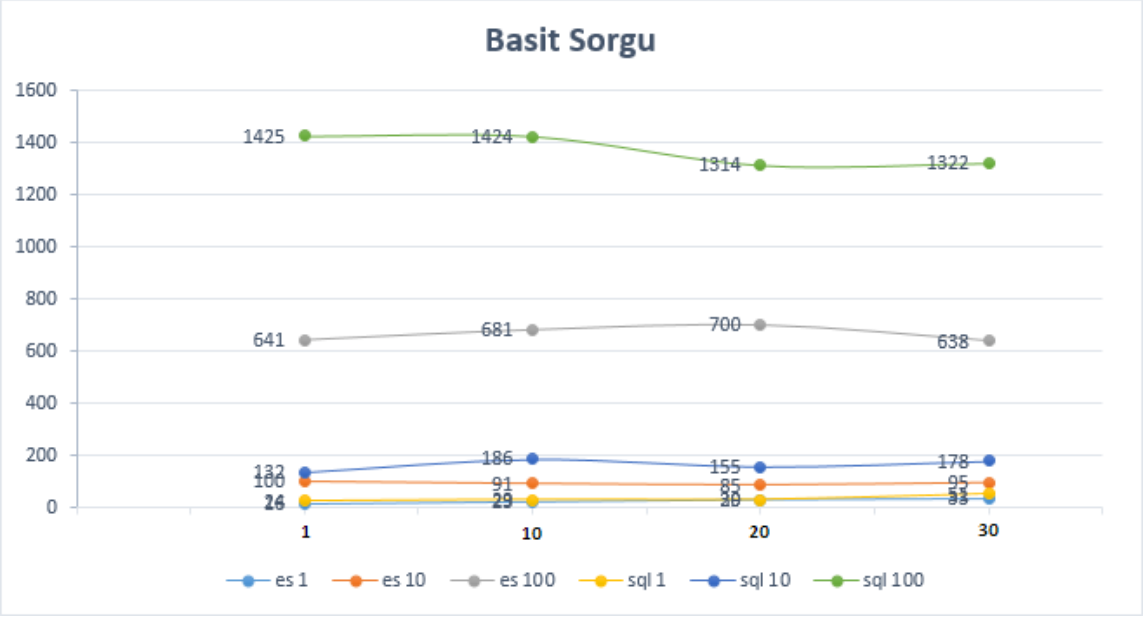
Grafikten gözükeceği gibi bu tip düşük seviye sorgularda ElasticSearch MSSQL'den daha hızlı çalışmaktadır.

Ancak sorgu gönderildiği zaman ElasticSearch daha fazla işlemci gücü harcamaktadır. Art arda 1, 10, 100'lü sorgu atıldığında işlemci daha da fazla çalışmaktadır.

Böyle bir sorgu için ElasticSearch'ün önbelleğini 2, 4, 6gb'ta tutmak, işlem hızını etkilememektedir.

Bir anda atılan sorgu sayısının artması, döndürülen sonuçların sayısının artması, MSSQL'in verimini ElasticSearch'e göre daha da olumsuz etkilemektedir. Ancak aynı anda 100 sorgu atıldığında bile verim kaybı artışı %10 civarındadır. Bu da çoğu durum için ihmal edilebilir.

Şekil 3. Küçük Sistem Basit Sorgu



Tablo 1. Küçük Sistem Basit Sorgu

	ES 1	ES 10	ES 100	SQL 1	SQL 10	SQL 100
SONUÇ 1	16	100	641	24	132	1425
SONUÇ 10	23	91	681	29	186	1424
SONUÇ 20	30	85	700	29	155	1314
SONUÇ 30	33	95	638	51	178	1322

3.4.2 Filtreleme Sorgusu

Filtreleme sorgusu da aynı veri bloğu üzerine atılmıştır. Her bir sütun beş rastgele değere göre sorgulanmıştır. Amacı:

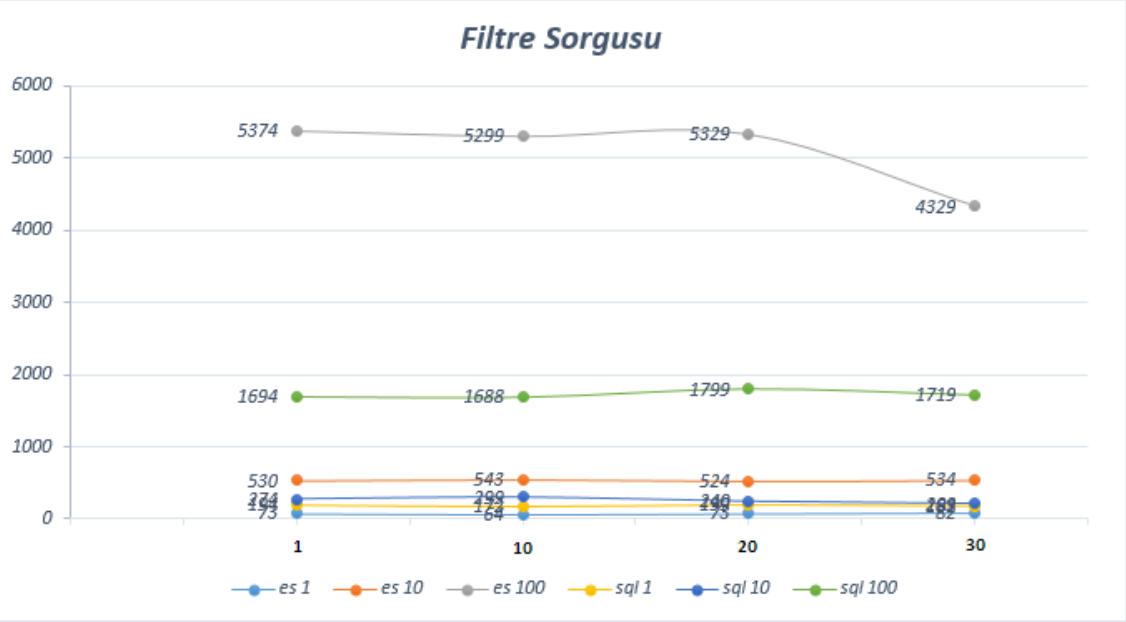
- Kriterlere uygun çalışan bulabilmek
- Satış sitelerinde ürünleri özelliklerine göre filtreleyebilmek

Grafikler incelendiğinde bir anda tek sorgu atıldığında, ElasticSearch sonuçları basit sorguda olduğu gibi daha hızlı getirmektedir. Ancak onarlı ve yüzerlik sorgularda MSSQL öne geçmektedir. Bunun nedeni olarak ElasticSearch'ün işlemciyi çok fazla kullanmasıdır. Onarlı ve yüzerlik sorgularda işlemci tüm süreç boyunca %100'e yakın çalışmaktadır. Ancak MSSQL aynı sorguyu attığında işlemci çok daha az kullanılmakta ve sonuçlar daha hızlı dönmektedir.

Veriyi birden çok sunucuya bölmek, ya da daha güçlü işlemci kullanmak, ElasticSearch'ün işlemciden kaynaklanan darboğazını ortadan kaldırıp, yine MSSQL'in önüne geçmesini sağlayacaktır.

Çoklu ElasticSearch sorgularının işlemci limitine ulaştığı için yavaşlaması, aynı makine üzerinde birden fazla node kullanılması durumunda verim artışı yaşanmaması anlamına gelmektedir. 10 ya da 100 tane karmaşık ElasticSearch sorgusunu aynı makine üzerinde çalışan bir, iki ve üç düğümü durumlarda denediğimde, tek düğümlü en hızlı çalıştığı bulgusuna ulaştım.

Şekil 4. Küçük Sistem Filtre Sorgusu



Tablo 2. Küçük Sistem Filtre Sorgusu

	ES 1	ES 10	ES 100	SQL 1	SQL 10	SQL 100
SONUÇ 1	73	530	5374	194	274	1694
SONUÇ 10	64	543	5299	172	299	1688
SONUÇ 20	73	524	5329	199	240	1799
SONUÇ 30	82	534	4329	181	209	1719

3.4.3 Aralık Sorgusu

Aralık sorgusunda aralık 50 olarak alınmıştır. Amacı:

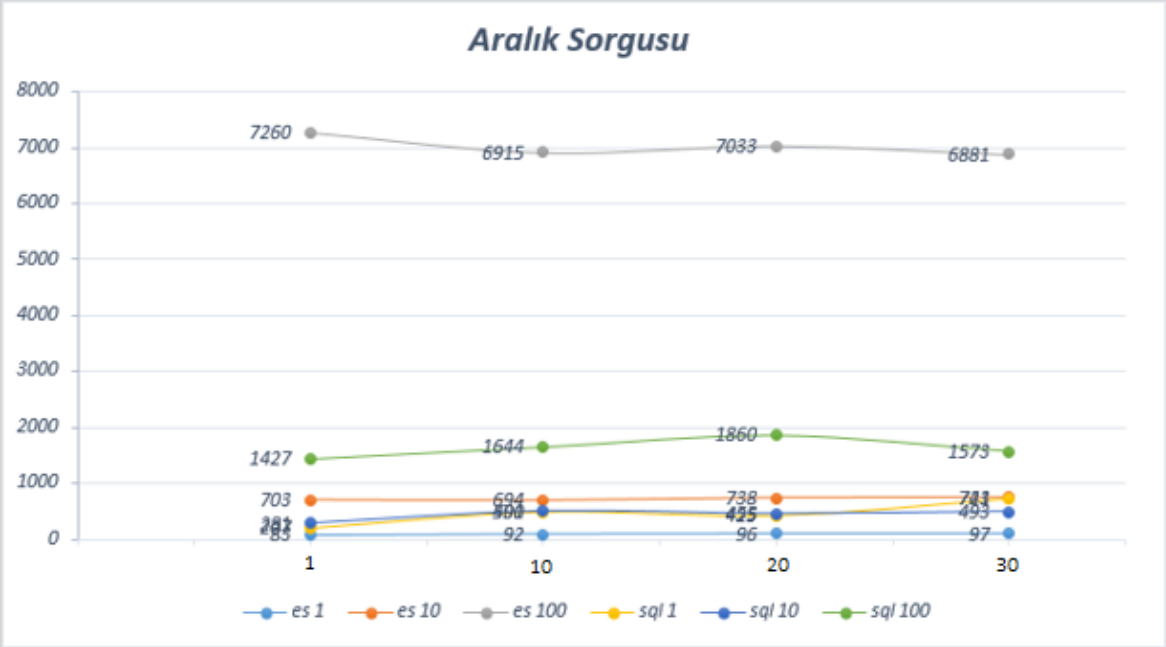
- Yaş aralığına göre seçim yapabilmek
- Ücretlendirmeye göre seçim yapabilmek
- Uzaklığa göre seçim yapabilmek

Aralık sorgusu nadiren tek başına kullanılır ve bir sorguda hemen hemen her zaman yalnızca bir tane aralık sorgusu bulunur. Bu nedenle sorgum bir tane aralık ve üç tane de çoklu sorgudan oluşmaktadır.

Tekli sorgular ElasticSearch üzerinde en hızlı şekilde bitmiştir. Diğer denemelerin aksine, MSSQL'in verimi bir anda getirilen sonuç sayısı arttıkça düşmektedir.

ElasticSearch'ün karmaşık sorgularda yaşadığı işlemci darboğazı aralık sorgusunda da ortaya çıkmıştır. Bu yüzden özellikle 100'lük sorgu setlerinde verim olarak MSSQL'in çok gerisinde kalmaktadır. Veri tabanını sunuculara bölerek bu darboğazdan kurtulunabilir.

Şekil 5. Küçük Sistem Aralık Sorgusu



Tablo 3. Küçük Sistem Aralık Sorgusu

ES	ES 1	ES 10	ES 100	SQL 1	SQL 10	SQL 100
SONUÇ 1	83	703	7260	207	281	1427
SONUÇ 10	92	694	6915	490	501	1644
SONUÇ 20	96	738	7033	425	455	1860
SONUÇ 30	97	743	6881	721	493	1573

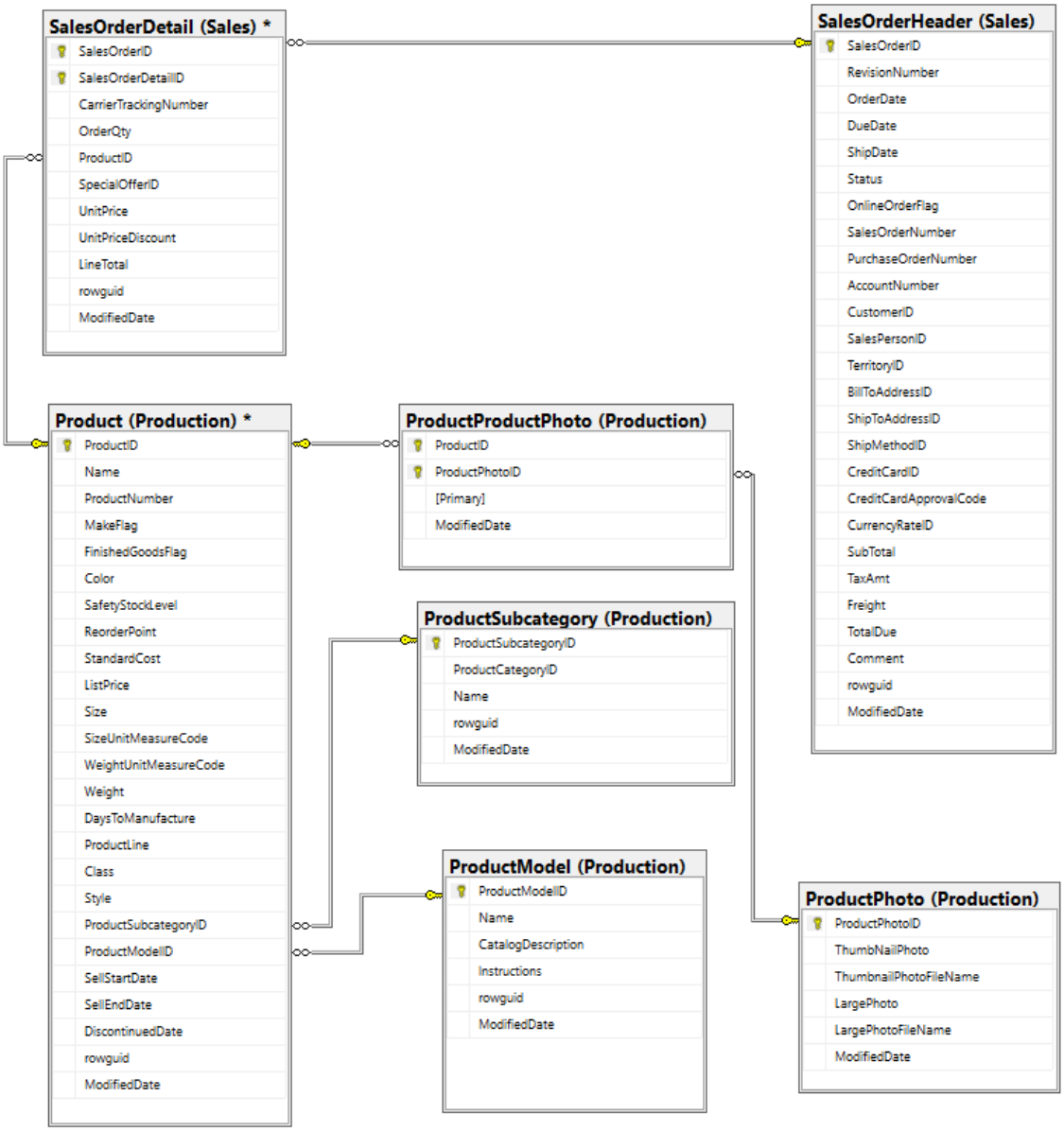
3.5 Büyük Sistemler

Kullandığımız makine 2 çekirdekli Intel Xeon ve 32gb DDR3 RAM'e sahiptir.

Veri tabanı ElasticSearch'te 120.000 satıra karşılık gelmektedir

Büyük düzende yaşanan işlemci sınırı sorunu bu denemede yaşanmamıştır.

Şekil 6. Büyük Sistem SQL Şeması



3.5.1 Basit Sorgu

İlk sorgum yalnızca üç değerden birinin doğru olmasının sağlandığı bir sorgu. Bu basit sorgunun amacı kullanıcının

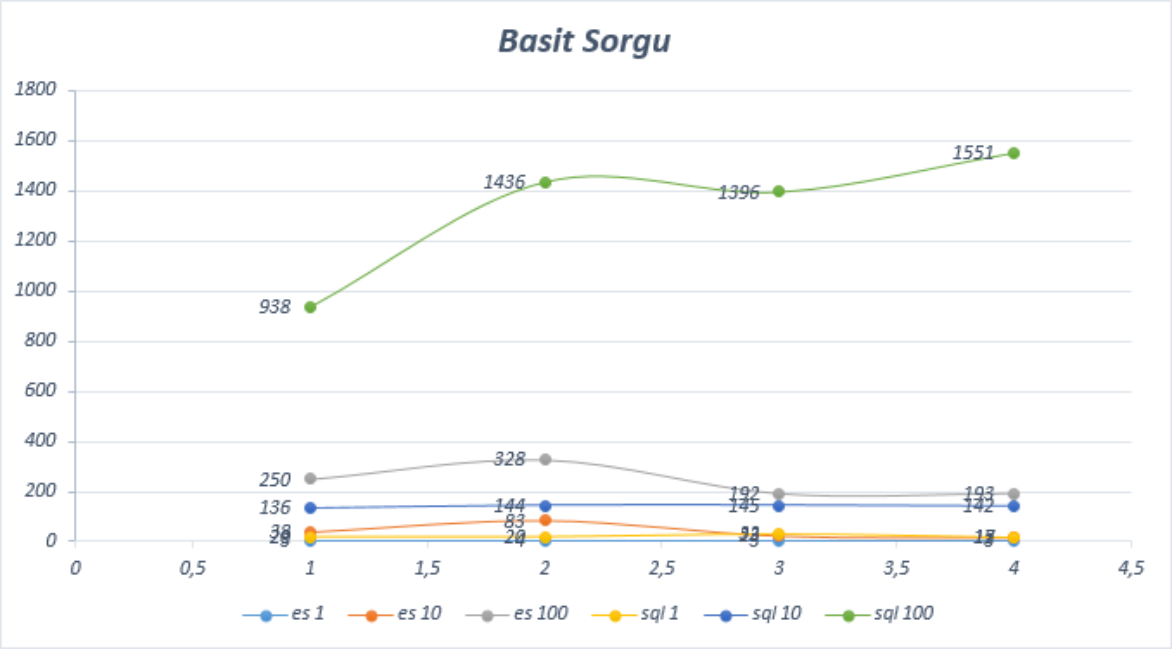
- Bir, iki müzik türü

- Sitedeki kişisel bilgiler
- Oyundaki karakter hakkındaki bilgiler
- CV bilgileri

gibi verileri getirmektir.

Grafikten gözükeceği gibi bu tip düşük seviye sorgularda ElasticSearch MSSQL'den daha hızlı çalışmaktadır.

Şekil 7. Büyük Sistem Basit Sorgu



Tablo 4. Büyük Sistem Basit Sorgu

	ES 1	ES 10	ES 100	SQL 1	SQL 10	SQL 100
SONUÇ 1	3	38	250	20	136	938
SONUÇ 10	4	83	328	20	144	1436
SONUÇ 20	3	23	192	32	145	1396
SONUÇ 30	3	15	193	17	142	1551

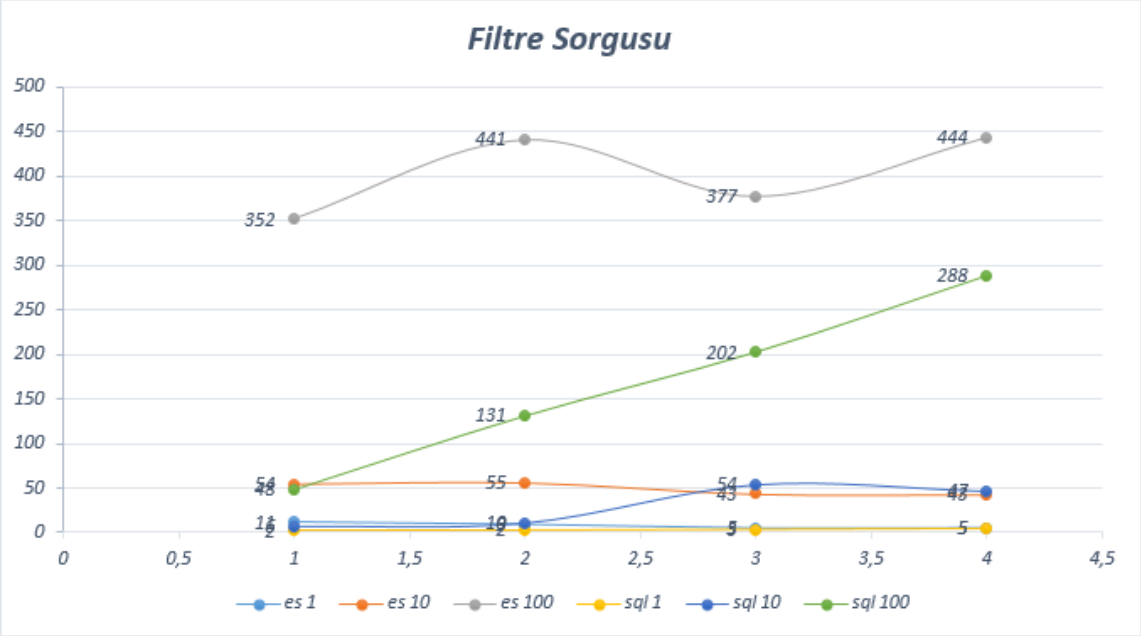
3.5.2 Filtreleme Sorgusu

Filtreleme sorgusu da aynı veri bloğu üzerine atılmıştır. Çoklu satır döndüren bir sütuna tek bir sorgu atılmıştır. Amacı:

- Hangi kullanıcıların, ne gibi şartlar altında hangi ürünleri birlikte aldığını anlamak
- Tek bir işlemde yapılan herhangi bir çoklu seçimin izlenmesi.

Diğer denemelerden öngörülenin aksine, ElasticSearch daha yavaş çalışmıştır. Aynı özellik için farklı dikeçlere sorgu atılması bu yavaşlığın nedeni olabilir.

Şekil 8. Büyük Sistem Filtre Sorgusu



Tablo 5. Büyük Sistem Filtre Sorgusu

	ES 1	ES 10	ES 100	SQL 1	SQL 10	SQL 100
SONUÇ 1	11	54	352	2	6	48
SONUÇ 10	9	55	441	2	10	131
SONUÇ 20	5	43	377	3	54	202
SONUÇ 30	5	43	444	5	47	288

SONUÇ

İki düzen üretilmiş ve bu düzenler teker teker denenerek sonuçlara ulaşılmıştır. Bu iki düzen büyük ve küçük veri tabanlarını taklit etmektedirler ve kullanıcı davranışlarının taklit edilebilmesi için de atılan her sorgunun değişkenleri farklılaştırılmıştır.

Büyük düzenler: güçlü düzen, az ve karmaşık veri; büyük firmaların çok ve karmaşık verisi olmaktadır ancak güçlü düzen yapabilecek kaynağa da sahiptirler. Bu sayede düzen herhangi bir dar boğaza neden olmaz.

Denemelerden çıkan sonuç; basit sorguların ElasticSearch üzerinde daha hızlı işlediğidir. Büyük düzenlerin karmaşık veri tabanları olur. Herhangi bir veri kümesi istendiğinde veri tabanının geneline yayılmış birçok tablodan veri çekilmesi gerekmektedir.

Basit sorguda birbirine bağlı yalnızca yedi tablo olmasına ve veriler kullanıcı gözünden, yalnızca 1, 10, 20 ve 30'arlı kümelerle çekilmesine rağmen MSSQL, ElasticSearch karşısında düşük bir verim sergilemiştir. Daha çok veri ve daha çok tablo eklenmesi durumunda verim farkının daha da artacağı söylenilebilir.

Filtre sorgusunda, her bir satırdaki alt satırlar içerisinde –her bir satın almadaki kalemler- arama yapılmıştır. Bu noktada ElasticSearch'ün MSSQL kadar hızlı çalışmadığı gözlemlenmektedir.

Kurulacak düzen, kullanılacak veri tabanı düzeni göz önüne alınarak kurgulanmalıdır. Veri tabanı düzeninin bir sorgu tipinde başarı elde etmesi, bir diğerinde de başarılı olacağı anlamına gelmemektedir.

Küçük düzenler: zayıf düzen, çok ve basit veri; her ne kadar ElasticSearch'ün veriyi bulmak için harcadığı zaman çok kısa olsa da, sorgunun JSON olarak gönderilmesi ve sonucun ElasticSearch'ten dönmesi zaman almaktadır.

ElasticSearch aynı anda birden fazla işlemi yapamamaktadır. Bu yüzden aynı anda çoklu sorgu göndermek (örnekteki aynı anda gönderilen 10 ve 100'lü sorgular), MSSQL'de olduğu gibi eşzamanlı gerçekleşmemekte, sonuçlar da uzun sürede dönmektedir.

Bu durumun üstesinden gelmek için aynı veri tabanını sorgulayan birden fazla ElasticSearch uygulaması (veri belgesi tarayıcısı) çalıştırmak gerekmektedir.

ElasticSearch, eğer toplu bir şekilde gönderilirse, sorguları eşzamanlı atabilmektedir. Ancak bunun için planlama gerekmektedir. Örneğin; sunucu tarafında sorgular hemen atılmamalı, biriktirilmeli, çoklu sorgu haline dönüştürülerek atılmalıdır. Ardından dönen sonuçlar, uygun istemcilere gönderilmelidir.

Ancak sonuçlar beklediğimiz gibi 1/2 veya 1/3 sürede gerçekleşmemiştir. Bunun nedeni karmaşık sorgunun ElasticSearch'ün işlemciyi tüm çekirdeklerde %100 kullanmasıdır. ElasticSearch uygulama sayısını arttırdıkça verim daha da düşmektedir.

Sonuç: Sorguyu mümkün olduğunca basit tutmak gerekmektedir. Basit tutulamıyorsa sunucu sayısını arttırarak sorguyu daha çok işlemciye / sunucuya yaymak gerekir. Eğer bu da yapılamıyorsa, böyle geniş aralıklı bir sorgu için ilişkisel veri tabanı yöntemi daha iyi sonuç verebilecektir. Ancak çoklu sunucu üzerine kurulu bölünmüş bir veri tabanının MSSQL'de yapılması da mümkündür. Bu nedenle bu şekilde kazanılacak bir verim artışı MSSQL için de geçerlidir.

Büyük düzenlerde ise işlemci darboğaz yapmamış, bunun sonucunda da ElasticSearch MSSQL'den daha hızlı çalışabilmiştir.

Filtreleme sorgusundaysa MSSQL daha hızlı çalışmıştır. ElasticSearch'ün sunucu tarafında kod iyileştirmesine gitmesi, ya da farklı kodlar kullanılarak verimini düzeltmesi söz konusu olabilir.

Yine de büyük düzenler için basit sorgunun ElasticSearch'te daha hızlı çalışmış olması, verim açısından karmaşık olmayan sorgular için ElasticSearch'ün kullanılabileceğinin bir göstergesidir.

ARAŞTIRMA BULGULARI VE TARTIŞMA

YSDK bir veri tabanı, istemcilere yani müşterilere, siteyi ziyaret edenlere veri sağlamak için kullanılabilir bir düzendir. Özellikle site çok yoğun bir trafiğe sahip değilse, YSD çözümlerden daha hızlı yanıt sağlayabilecektir. Bu tezin asıl sorduğu soru da budur.

Beklenti, bu kadar büyük bir veri setinde YSDK çözümün her durumda YSD çözümden daha hızlı çalışmasıydı. Ancak durum tam olarak böyle olmamıştır. Sorgu sayısının düşük olduğu durumlarda YSDK çözüm gerçekten daha hızlı çalışmış olsa da sorgu sayısı ve karmaşıklığı artarken YSD çözümün gerisinde kalmaya başlamıştır. Bunun nedenininse YSDK çözümün çok daha fazla işlemci kaynağı kullanmasıdır.

Bu düzen kaynağı verildiğindeyse YSDK çözüm veri kümesine ve kullanılan sorguya göre YSD çözümden daha iyi ya da daha kötü verim sergileyebilmektedir.

Bazı YSDK çözümlerin çalışma mantığı YSD çözümlerden çok farklıdır. İşlemci kullanımını en üst düzeyde tutarak daha hızlı sonuç dönülmesi sağlamaktadır. Aynı zamanda eş zamanlı sorgu atma da oldukça farklı çalışmaktadır; sorgular kümelenerek gönderildiği durumda eş zamanlı sorgu sağlanabilmektedir. Ancak gerçek hayatta sorguların bu şekilde gönderilmesi çok kullanışlı değildir.

Bulgular daha önce yapılan çalışmalarla kısmen uyumlu gözükmektedir. Az sorgu sayısı ve karmaşıklığı durumunda önceki çalışmalarla uyumluyken, yüksek sorgu sayısı ve karmaşıklığı durumunda uyumsuzdur. Önceki çalışmalar işlemci dar boğazından söz etmemektedir. Önceki çalışmaların bu kadar zorlayıcı sorgular içermemesi, hep aynı değişkenli sorguları atarak verileri sunucu önbelleğinden getirmesi de bulguların farklılığına bir neden olabilir.

ElasticSearch gerçekten de her site için uygun gözükmemektedir. Az istek aldığı durumlarda MSSQL'in önüne geçse bile 30-100 milisaniye ölçeğindeki kazanımlar, düzenin devam ettirilmesindeki zorluk göz önüne alındığında ihmal edilebilir. Ancak

yoğun şekilde basit sorgu istemi aldığında, bu fark bir saniyenin üzerine çıkabilmektedir. Bu durumda da ElasticSearch tercih edilebilir.

Bundan sonraki çalışmalar, sunucu sayısının arttırılması, veri tabanının sunuculara yayılması ve daha da karmaşık sorgularda ortaya çıkacak sonucun araştırılması olabilir. Ancak önceki çalışmalar da bu tip kurgular yapmış, sonucu tek makine kullanıldığı durumla hemen hemen aynı bulmuştur.

KAYNAKÇA

[1] A. Moniruzzaman ve S. Hossain, NoSQL Database: New Era of Databases for Big data Analytics – Classification, Characteristics and Comparison, 2013.

[2] O. B. R. H. M. W. G. Oleksii Kononenko, Mining Modern Repositories with Elasticsearch, Waterloo.

[3] H. Hadjigeorgiou, *RDBMS vs NoSQL: Performance and Scaling Comparison*, p. 35, 2013.

[4] B. Alexandru, F. Radulescu ve L. Agapin, MongoDB vs Oracle - database comparison, Romania, 2012.

[5] Z. Parker, S. Poe ve S. Vrbsky, Comparing NoSQL MongoDB to an SQL DB, 2013.

[6] https://www.elastic.co/use-cases, 2018.

[7] https://www.eversql.com/most-popular-databases-in-2018-according-to-stackoverflow-survey/, 2018.

[8] R. Elmasri, Fundamentals of Database Systems, Pearson Education, 2011.

[9] Beynon ve P. Davies, Database Systems, Palgrave Macmillan.

[10] R. Cattell, Scalable SQL and NoSQL Data Stores, 2010.

[11] F. Chang, J. Dean, S. Ghemawat, C. Hsieh, M. Burrows, T. Chandra, A. Fikes ve E. Gruber, Bigtable: A Distributed Storage System for Structured Data, 2016.

[12] C. Curino, E. Jones, Y. Zhang ve S. Madden, Schism: a Workload-Driven Approach to Database Replication and Partitioning, 2018, p. 1.

[13] B. M. B, M. Ouzzani ve A. Elmagarmid, Generalization of ACID Properties, Purdue University, 2009.

Ek-1: SORGULAR İÇİN AÇIKLAMA

Sorgularda sol taraf ElasticSearch, sağ taraf MSSQL'i göstermektedir

Her test ortalama için yedi kez tekrarlanmış ve ortalamadan sapan sonuçlar elenmiştir.

Sütun başlarındaki numaralar sırasıyla kaç sonuç döndüğü ve kaç sorgu atıldığını göstermektedir.

Tablolardan sonra ElasticSearch sorguları yer almaktadır.

#A# - #E# 1 ile 100 arasında bir değişkendir ve sorgu her yinelenişinde değişmektedir.

#F# - #J# 1 ile 1000 arasında bir değişkendir ve sorgu her yinelenişinde değişmektedir.

Ek-2: KÜÇÜK SİSTEMLERİN BASİT SORGUSUNA AİT VERİLER

Tablo 6. Küçük Sistem Basit Sorgu Tüm Veri

es x 1 node				**mssql**		
1-1	**1-10**	**1-100**		**1-1**	**1-10**	**1-100**
10.5611	63.9858	668.2692		22.4693	216.6733	1430.7849
9.4557	58.9333	619.0135		28.2526	134.2995	1574.153
9.9667	58.9559	597.5399		23.471	126.3191	1399.0908
10.2728	58.2287	527.1545		27.7463	144.3247	1432.1041
16.0626	57.6188	575.5649		25.4826	137.138	1390.7304
12.0325	52.1178	574.1639		25.0026	123.1571	1333.1066
9.0519	55.2351	542.3674		24.2487	127.5829	1495.0017
11.0576	**57.8679**	**588.0847**		**24.73675**	**132.1369**	**1425.837**
10-1	**10-10**	**10-100**		**10-1**	**10-10**	**10-100**
13.1833	59.2878	524.5932		35.7452	172.2378	1321.1532
11.0057	62.2851	536.6657		32.8313	205.4035	1430.7849
10.9973	63.5201	548.6427		28.5569	129.5143	1579.2192
11.0348	68.6948	538.5904		28.0907	229.5811	1332.9
14.1456	60.7591	598.732		19.2802	152.4638	1592.3066
11.2509	59.1633	649.6539		29.9739	147.8759	1359.1663
12.0898	56.2024	798.3117		30.7746	264.9753	1359.1663
11.9582	**61.7708**	**599.4096**		**29.3218**	**186.0074**	**1424.9566**

20-1	**20-10**	**20-100**		**20-1**	**20-10**	**20-100**
18.027	52.1372	603.5871		35.7452	171.4498	1540.5254
12.3294	55.2138	640.8124		32.8313	184.6019	1365.5526
13.0036	68.2148	538.3485		28.5569	121.1836	1129.3626
13.6618	65.884	583.1821		28.0907	137.5148	1217.9588
14.9375	31.1787	708.4322		19.2802	153.4502	1257.2015
13.1964	55.2206	623.7615		29.9739	147.601	1500.8698
15.034	66.0848	600.9794		30.7746	175.8048	1187.3733
14.6434	**56.2763**	**614.1576**		**29.3218**	**155.9437**	**1314.1206**
30-1	**30-10**	**30-100**		**30-1**	**30-10**	**30-100**
18.175	60.1915	610.4142		54.2694	135.4275	1447.6969
9.0902	60.1468	640.9117		45.7728	210.4973	1090.9573
17.2283	62.6413	587.1759		42.2011	150.3538	1413.3808
23.2328	70.2662	575.0176		51.1319	238.5847	1296.3108
24.0707	66.3164	656.7453		58.9915	140.3693	1380.8712
16.1085	55.8618	588.3909		57.017	203.2484	1277.7488
16.0352	72.5931	604.3772		53.9835	170.5889	1347.5965
17.7058	**64.0024**	**609.0047**		**51.9096**	**178.4386**	**1322.0803**

Tablo 5.1: Basit sorgu

```
{
   "query": {
   "bool": {
      "should": [
         { "match": { "A1ID": #A# } },
         { "match": { "A1ID": #B# } }
      ]
      }
   },
   "from": 0,
   "size": 30
}
```

Ek-3: KÜÇÜK SİSTEMLERİN FİLTRE SORGUSUNA AİT VERİLER

Tablo 7. Küçük Sistem Filtreleme Sorgusu Tüm Veri

es x 1 node				**mssql**		
1-1	**1-10**	**1-100**		**1-1**	**1-10**	**1-100**
71.8286	569.2837	5463.3516		188.953	253.6561	1931.4055
73.406	543.1179	5578.1712		308.7833	392.284	1732.8005
63.7297	570.7291	5143.0182		173.4571	239.405	1466.8766
86.9692	579.2097	5383.9173		179.2983	169.2083	1716.0955
64.3152	464.5259	4635.934		195.8311	349.5809	1689.6017
71.3787	433.8937	5121.7373		230.56	246.1087	1548.8026

80.6456	550.7747	5554.8979		198.0098	249.6266	1775.7438
73.1819	**530.2192**	**5374.1823**		**194.3516**	**274.3689**	**1694.4752**
10-1	**10-10**	**10-100**		**10-1**	**10-10**	**10-100**
62.1782	518.906	5385.6547		184.208	233.5487	1621.3945
42.8341	525.9621	4721.2238		169.7159	255.2553	1528.5738
73.5223	537.7957	5146.5375		154.2911	315.5886	1414.4977
77.927	554.7996	5593.9145		193.6747	331.7538	2008.0131
59.808	542.4797	5379.9357		183.1141	255.0605	1580.7229
62.1683	527.1851	5486.1619		153.4264	356.6774	1931.4055
76.0508	575.3125	5463.3516		172.0619	348.1678	1732.8005
64.9270	**543.9225**	**5299.4740**		**172.9274**	**299.4360**	**1688.2011**
20-1	**20-10**	**20-100**		**20-1**	**20-10**	**20-100**
78.8796	512.7448	5551.6444		168.0895	255.2479	1399.5801
87.0827	492.2302	5496.9915		206.8345	193.1104	1672.6809
75.6304	410.5296	5352.2214		208.3312	250.467	3446.6382
74.9766	561.2471	5370.3595		186.2541	226.0561	1519.3012
71.0295	553.794	5428.2767		173.9199	189.9075	1581.1754
69.8473	571.2797	5385.6547		268.2411	215.2425	1382.2384
70.7569	566.9892	4721.2238		184.8921	352.1337	1594.3999
73.5201	**524.1164**	**5329.4817**		**199.5089**	**240.3093**	**1799.4306**
30-1	**30-10**	**30-100**		**30-1**	**30-10**	**30-100**
75.9666	517.7497	5661.7665		191.4998	210.2465	1801.6812
83.2678	539.3668	5572.0481		164.931	215.9814	1961.9654
83.1456	488.049	5550.4862		178.3727	232.1428	1506.0029
78.6794	562.6494	5676.9989		183.6057	206.5199	1641.1718
86.2485	554.3652	5734.2318		195.8969	173.5952	1563.4621
85.4751	524.6707	5478.5366		175.2584	236.0156	1778.2394
82.9565	554.1785	5790.7629		178.5864	191.2748	1783.1916
82.2485	**534.4328**	**5637.8330**		**181.1644**	**209.3966**	**1719.3878**

```
{
   "query": {
      "bool": {
         "should": [
         { "match": { "A1ID": #A# } },
         { "match": { "A1ID": #B# } },
         { "match": { "A1ID": #C# } },
         { "match": { "A1ID": #D# } },
         { "match": { "A1ID": #E# } }
      ],
         "should": [
         { "match": { "A2ID": #A# } },
         { "match": { "A2ID": #B# } },
         { "match": { "A2ID": #C# } },
```

```
            { "match": { "A2ID": #D# } },
            { "match": { "A2ID": #E# } }
        ],
            "should": [
            { "match": { "A3ID": #A# } },
            { "match": { "A3ID": #B# } },
            { "match": { "A3ID": #C# } },
            { "match": { "A3ID": #D# } },
            { "match": { "A3ID": #E# } }
        ],
            "should": [
            { "match": { "A4ID": #A# } },
            { "match": { "A4ID": #B# } },
            { "match": { "A4ID": #C# } },
            { "match": { "A4ID": #D# } },
            { "match": { "A4ID": #E# } }
        ],
            "should": [
            { "match": { "B1ID": #F# } },
            { "match": { "B1ID": #G# } },
            { "match": { "B1ID": #H# } },
            { "match": { "B1ID": #I# } },
            { "match": { "B1ID": #J# } }
        ],
            "should": [
            { "match": { "B2ID": #F# } },
            { "match": { "B2ID": #G# } },
            { "match": { "B2ID": #H# } },
            { "match": { "B2ID": #I# } },
            { "match": { "B2ID": #J# } }
        ],
            "should": [
            { "match": { "B3ID": #F# } },
            { "match": { "B3ID": #G# } },
            { "match": { "B3ID": #H# } },
            { "match": { "B3ID": #I# } },
            { "match": { "B3ID": #J# } }
        ]
        }
    },
    "from": 0,
    "size": 1
}
```

Ek-4: KÜÇÜK SİSTEMLERİN ARALIK SORGUSUNA AİT VERİLER

Tablo 8. Küçük Sistem Aralık Sorgusu Tüm Veri

es x 1 node				**mssql**		
1-1	**1-10**	**1-100**		**1-1**	**1-10**	**1-100**
68.9639	733.3479	7101.0916		70.1584	191.1921	1375.2471
93.9707	576.4991	7021.3158		96.7295	169.7949	1660.3421
83.9309	741.2741	7303.1301		223.6515	260.5563	1293.2359
74.9953	741.6062	7103.7953		146.9617	607.5909	1565.6618
83.9301	617.0514	7079.7321		243.9869	216.8975	1666.6498
83.1511	733.2873	7562.9337		445.1832	183.7823	1031.5049
94.9687	778.8455	7466.4556		220.3521	246.2203	1398.3239
83.4158	**703.1302**	**7259.7870**		**206.7176**	**280.8070**	**1427.2808**
10-1	**10-10**	**10-100**		**10-1**	**10-10**	**10-100**
89.6329	742.8012	6177.5949		487.1716	217.9967	1401.3733
87.3956	638.2914	7057.1735		461.6719	691.4612	2023.828
98.3345	710.7836	6986.6794		468.7806	372.5633	1532.9303
95.1705	750.9899	7097.5789		565.3616	720.2408	1422.0155
83.6	749.7849	7344.1538		520.9836	136.2775	1584.0832
93.3274	700.8056	6856.4531		456.8697	945.1833	1374.9317
97.5665	611.9505	7312.2989		467.022	421.3242	2169.72
92.1468	**693.7677**	**6914.6298**		**489.6944**	**500.7210**	**1644.1260**
20-1	**20-10**	**20-100**		**20-1**	**20-10**	**20-100**
91.5305	797.6204	6838.785		478.6268	126.4404	1613.3136
100.1927	742.6687	7416.3526		394.0165	637.7903	1494.9518
108.5778	627.2982	7217.661		428.2638	193.8447	2293.1544
91.206	727.7161	6799.4574		413.6356	844.0004	1385.231
91.7469	723.1145	7399.6876		455.8021	483.6278	2324.2055
107.064	756.5862	7383.5885		426.9095	175.2815	2062.8466
87.6705	793.0817	6177.5949		378.9509	723.2591	1846.1986
96.2993	**738.2980**	**7033.3039**		**425.1722**	**454.8920**	**1859.9859**
30-1	**30-10**	**30-100**		**30-1**	**30-10**	**30-100**
117.7191	771.9336	6532.967		493.9288	720.6112	1507.1556
97.2572	723.3955	6905.3626		453.2658	435.6487	1544.6552
81.669	701.9184	7284.4626		448.5863	196.9617	1736.0455
104.2333	777.3112	6886.7574		539.2083	707.8729	1303.3232
106.3106	727.3741	6736.1093		459.622	471.595	1622.2541
92.8222	754.8332	6852.0031		545.2757	157.2968	1682.4636
81.1436	743.4707	6968.5749		158.406	760.3945	1613.3136
97.3079	**742.8910**	**6880.8910**		**720.6112**	**492.9115**	**1572.7444**

```
{
   "query": {
   "bool": {
      "must": { "range": { "A1ID": { "gte": #A#, "lte": #A0# } } },
         "should": [
      { "match": { "A2ID": #A# } },
      { "match": { "A2ID": #B# } },
      { "match": { "A2ID": #C# } },
      { "match": { "A2ID": #D# } },
      { "match": { "A2ID": #E# } }
   ],
         "should": [
      { "match": { "A3ID": #A# } },
      { "match": { "A3ID": #B# } },
      { "match": { "A3ID": #C# } },
      { "match": { "A3ID": #D# } },
      { "match": { "A3ID": #E# } }
   ],
         "should": [
      { "match": { "A4ID": #A# } },
      { "match": { "A4ID": #B# } },
      { "match": { "A4ID": #C# } },
      { "match": { "A4ID": #D# } },
      { "match": { "A4ID": #E# } }
   ]
      }
   },
   "from": 0,
   "size": 1
}
```

Ek-5: BÜYÜK SİSTEMLERİN BASİT SORGUSUNA AİT VERİLER

Tablo 9. Büyük Sistem Basit Sorgu Tüm Veri

es x 1 node				**mssql**		
1-1	**1-10**	**1-100**		**1-1**	**1-10**	**1-100**
2.9963	43.9999	231.0131		14.9969	116.0061	767.0402
1.9979	17.9999	314.018		15.9984	140.0053	1062.0638
2.9971	37.0009	167.0067		17.9999	119.0044	1085.0677
3.9966	18.0003	260.0132		15.9941	135.0056	1075.0628
3.9978	113.9983	329.0158		20.0018	175.0099	830.0506
3.9974	20.0002	393.0228		30.0009	109.0057	700.0415

1.996	22.0016	132.0065		24.0004	139.0049	1047.0604
3.1399	**38.0029**	**249.5134**		**19.8561**	**136.1726**	**938.0553**
10-1	**10-10**	**10-100**		**10-1**	**10-10**	**10-100**
2.9979	47.0021	602.0346		18.0007	105.0061	966.0574
8.9996	90.0043	272.0154		25.002	93.0067	1350.0865
2.9995	117.0041	150.0092		16.0011	114.9982	1591.1006
3.0006	32.0012	440.0213		14.9999	261.0141	1317.0823
2.9979	107.0054	186.0102		15.0003	60.0029	1491.0915
5.0001	114.0046	365.0214		35.0025	163.0092	1611.1025
3.9967	37.0017	139.0042		15.9964	213.0102	1723.1073
4.2846	**82.8369**	**328.0177**		**20.0004**	**144.2925**	**1435.6612**
20-1	**20-10**	**20-100**		**20-1**	**20-10**	**20-100**
1.9967	47.002	333.0191		49.0026	132.0078	1889.1175
4.9993	34.0023	105.0047		20.9995	145.0072	2079.1315
2.9959	19.999	211.0085		19.0008	177.007	1554.0994
3.999	12.0002	104.0055		16.0005	114.0066	909.0575
1.9995	15.0012	185.0106		87.0038	127.0064	1159.0737
2.9971	16.0012	122.0058		15.9994	165.0079	800.0422
1.9967	16.0012	286.0155		17.0012	153.01	1379.0839
2.6642	**22.8582**	**192.2957**		**32.1440**	**144.7218**	**1395.6580**
30-1	**30-10**	**30-100**		**30-1**	**30-10**	**30-100**
2.9971	18.977	117.0065		14.9957	122.9853	1641.8733
2.0003	15.002	123.0062		17.0245	126.9873	1459.8873
2.9986	14.0005	176.0075		17.0012	134.9877	1373.894
1.9979	15.0013	351.0194		16.0009	142.9837	1633.8751
2.9995	15.9981	175.0099		16.0001	108.987	1766.8638
1.9967	17.9991	268.0128		16.0001	126.9853	1690.8553
4.9997	11.0002	141.008		19	230.9758	1289.8921
2.8557	**15.4255**	**193.0100**		**16.5746**	**142.1274**	**1551.0201**

```
{
   "query": {
      "bool": {
         "should": [{
            "match": {
```

```
                "ID":"#A#"
            }
        },{
            "match": {
                "ID":"#B#"
            }
        },{
            "match": {
                "ID":"#C#"
            }
        }]
    }
  },
  "size": 30
}
```

Ek-6: BÜYÜK SİSTEMLERİN FİLTRE SORGUSUNA AİT VERİLER

Tablo 10. Büyük Sistem Filtre Sorgusu Tüm Veri

es x 1 node				**mssql**		
1-1	**1-10**	**1-100**		**1-1**	**1-10**	**1-100**
8.0012	31.9989	340.0083		1.0003	5.9999	36.9995
8.9998	120.995	337.0057		1.0003	4.9995	27.9991
5.9954	35.0819	301.007		0.9987	3.9996	49.0009
9.0001	67.1033	242.0032		6.0015	4.9992	46.0008
19.9985	37.0039	290.0072		2.9978	4.9988	36.0013
20.9988	45.0422	496.0105		0.9995	11.0002	98.0011
6.9969	41.01	393.0083		1.0003	8.9996	44.0002
11.4272	**54.0336**	**351.5072**		**1.9998**	**6.4995**	**48.2861**
10-1	**10-10**	**10-100**		**10-1**	**10-10**	**10-100**
6.0007	99.0039	353.014		0.9988	10.9993	148.0039
29.0012	42.0011	425.0198		2.0007	8.9978	85.0018

5.0048	53.9983	488.0218		4.9988	8.0002	208.0084
5.9992	69.0023	559.0237		1.9999	8.9994	133.0052
4.9976	67.0033	400.0166		1.9995	14.0006	82.0032
4.9973	37.003	420.0173		1.9998	12.9999	80.0025
6.0003	62.0028	403.0145		1.9994	7.9979	179.0043
8.8573	**55.1685**	**441.3519**		**2.2853**	**10.2850**	**130.7185**
20-1	**20-10**	**20-100**		**20-1**	**20-10**	**20-100**
6.0008	41.002	255.0118		1.9992	12.0005	140.006
4.9989	64.0027	311.0147		2.9993	76.0037	303.0147
4.9945	39.0006	488.0249		3.0001	34.0013	186.0063
4.9989	30.0018	428.0215		1.9973	19.9981	106.0043
3.9978	45.0022	371.0188		2.9911	95.0037	220.011
3.9978	42.002	392.0208		2.0001	130.005	270.0126
7	42.9949	393.0188		8.9999	11.9989	190.0103
5.1650	**43.4295**	**376.8759**		**3.4267**	**54.1445**	**202.1522**
30-1	**30-10**	**30-100**		**30-1**	**30-10**	**30-100**
6.0001	42.0026	439.0249		8.9979	71	235.0328
4.9986	34.0015	541.0281		7	19.0004	225.0296
4.8894	35.0019	573.0289		2.9924	49.9992	332.0479
4.9935	53.0025	366.0206		5.9981	33.9975	223.0341
5.0006	32.9996	460.0251		3.9816	61.9922	254.0119
6.0009	59.001	284.0144		2.9966	25.001	227.011
4.9994	44.0006	444.0179		1.996	66.003	517.0237
5.2689	**42.8585**	**443.8800**		**4.8518**	**46.7133**	**287.5987**


```
{
   "query": {
      "bool": {
         "should": [{
            "term": {
               "orders.order1.30": "#E#"
            }
         },{
            "term": {
               "orders.order2.30": "#E#"
```

```
            }
          },{
            "term": {
              "orders.order3.30": "#E#"
            }
          },{
            "term": {
              "orders.order4.30": "#E#"
            }
          },{
            "term": {
              "orders.order5.30": "#E#"
            }
          }]
        }
    },
    "size": 1
}
```

ÖZGEÇMİŞ

03 Şubat 1983 tarihi, İstanbul İli Üsküdar ilçesi doğumluyum. Ortaokulu Almanca olmak üzere, ortaokul ve liseyi Üsküdar Anadolu Lisesi'nde okudum. Marmara Üniversitesi'nde kimya öğretmenliğini tezsiz yüksek lisanslı olarak bitirdim. 2009 yılında mezun olduktan sonra askerliğimi Çankırı'da yaptım ve ardından yazılım sektöründe çalışmaya başladım. Orta ve büyük boy firmalarda veri tabanı, çeşitli masa üstü uygulamaları, Sharepoint, raporlama, ağ uygulamaları üzerinde çalıştım. 2013 yılında Beykent Üniversitesi bilgisayar mühendisliği bölümünde yüksek lisans eğitimine başladım. Yabancı dilim İngilizce ve Almancadır.