TC. İSTANBUL KÜLTÜR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK VERİTABANLARI İÇİN TEST UYGULAMASI

YÜKSEK LİSANS TEZİ

Gözde KARATAŞ

1309261001

Anabilim Dalı : Bilgisayar Mühendisliği
Programı : Bilgisayar Mühendisliği

Tez Danışmanı : Yrd.Doç.Dr. Akhan AKBULUT

HAZİRAN 2015

TC. İSTANBUL KÜLTÜR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK VERİTABANLARI İÇİN TEST UYGULAMASI

YÜKSEK LİSANS TEZİ

Gözde KARATAŞ

1309261001

Tezin Enstitüye Verildiği Tarih : 10 Haziran 2015
Tezin Savunulduğu Tarih : 30 Haziran 2015

Tez Danışmanı : Yrd.Doç.Dr. Akhan AKBULUT
Diğer Jüri Üyeleri : Doç.Dr.Banu DİRİ (YTÜ)
Doç.Dr. Çağatay ÇATAL (İKÜ)

HAZİRAN 2015

ÖNSÖZ

Bu çalışma, İstanbul Kültür Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans Tezi olarak hazırlanan ”Dağıtık Veritabanları İçin Test Uygulaması” isimli tezi içermektedir.

Çalışmalarımın her aşamasında bilgi ve deneyimleri ile yardımcı olan ve bana büyük emekleri geçen, kendisinden çok şey öğrendiğim danışmanım Sayın Yrd. Doç. Dr. Akhan AKBULUT'a içtenlikle teşekkür ederim.

Çalışmamın şekillenmesinde emeği olan, karşılaştığım problemlerde özgün fikirlerinden faydalandığım Sayın Yrd. Doç. Dr. Levent ÇUHACI'ya ve değerli eşi Sibel ÇUHACI'ya, teşekkür ederim.

Bu günlere gelmemde en büyük rolü oynayan, bana her zaman güç ve güven veren, desteğini esirgemeyen annem Mükerrem ÇALIŞKAN'a, her ne konuda olursa olsun her kararımı sorgusuzca destekleyen ablalarım Nursevel EREN ile Gülfer HÜRDOĞAN'a ve beni evlatlarından ayırmayan eniştelerim Yusuf EREN ile Reyhan HÜRDOĞAN'a teşekkür eder, bu değerli insanların ailem olmasından duyduğum sonsuz mutluluğu belirtmek isterim. Ayrıca çeviri yapma ve kaynakları incelemede benimle birlikte uzun süre harcayan yeğenim Merve EREN'e teşekkür ederim.

Çalışma sırasında beni destekleyen, gerek kaynak sağlama gerekse düzeltmeleri yapma konusunda yardımlarını esirgemeyen Matematikçi İlayda ATEŞ'e ve Erdi AKPINAR'a, manevi olarak desteklerini her zaman yanımda hissettiğim değerli arkadaşlarım İlkay MERGER ve Begüm ÇALIŞKAN'a çok teşekkür ederim.

Haziran 2015

Gözde KARATAŞ

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

TABLO LİSTESİ

Ensititü: Fen Bilimleri Enstitüsü

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Akhan AKBULUT

Tez Türü ve Tarihi: Yüksek Lisans – Haziran 2015

DAĞITIK VERİTABANLARI İÇİN TEST UYGULAMASI

ÖZET

İnternetin yaygınlaşması ile birlikte büyük veri kullanımında İlişkisel Veritabanı Yönetim Sistemleri (İVTYS), ölçeklenebilirlik konusunda yetersiz kalmaya başlamıştır. Son 10 yılın yükselmeye başlayan yeni veri saklama teknolojisi Dağıtık Veritabanları İVTYS'lerin sunamadığı hizmetleri sağlamaktadır. Bu çalışmanın amacı, yazılım mühendisliği kapsamında önemli bir konu olan yazılım testlerinin, kullanımı artan bu sistemlere uygulanmasıdır. Bir dağıtık veritabanı uygulaması olan MongoDB üzerinde farklı sınamaların gerçeklendiği test uygulaması geliştirilmiş olup, bu uygulamaya "MongoDB Tester" ismi verilmiştir. Bu uygulama kullanılarak Dağıtık Veritabanları üzerinde yapılan iyileştirmelere ait sonuçlar paylaşılmıştır. MongoDB Tester uygulamasının önerisi ile yapılan iyileştirmelerle elde edilen sonuçlarda sistemde iyileştirmeler olduğu görülmüştür.

GÖZDE KARATAŞ

Anahtar Kelimeler : Dağıtık Sistemler, Yazılım Testi, MongoDB, NoSQL

Institute: Institute of Sciences

Department: Computer Engineering

Programme: Computer Engineering

Supervisor: Assis. Prof. Dr. Akhan AKBULUT

Degree Awarded and Date: M.Sc.– June 2015

TESTING UTILITIES FOR DISTRIBUTED DATABASE'S

ABSTRACT

As the Internet and Big Data become more widespread the use of İVTYS is starting to not be enough. In the past 10 years, the rise of Distributed Databases provides what İVTYS cannot. The purpose of this study is to apply the software testing on this increasing ly used system. The utility was designed by applying various test on MongoDB which is a distributed database application, the shared results were achieved after amendment, and the name of this application is "MongoDB Tester". The results of enhancement on distributed databases are shared by using of this application. The results which the proposal of MongoDB Tester application, have been seen that improvements in the system in the certain cases.

GÖZDE KARATAŞ

Keywords : Distributed Systems, Software Testing, MongoDB, NoSQL

1. GİRİŞ

Teknolojinin gelişmesi ile birlikte bilgisayarlar hayatımızın büyük bölümünü kaplamaya başlamıştır. Milyarlarca insan bilgisayarlara rahatlıkla ulaşabilir, gerekli işlerini en kısa sürede bilgisayarlar ile halledebilir duruma gelmiştir. İnternetin de hayatımıza girmesiyle insanlar bilgisayar başında daha çok vakit geçirmeye, yeni şeyler keşfetmeye başlamışlardır. İnsanlar aileleri ile görüşmeye, internet aracılığı ile çevrim içi oyunlar oynanmaya, bankacılık işlemlerini hızlı bir şekilde halletmeye, firmaların ellerindeki ürünler ile ilgili gerekli bilgilere ulaşmaya bilgisayar kullanımı ile başlamışlardır.

Bilgisayarların yaygınlaşması ile birlikte ona olan ihtiyaç artmakta, buna bağlı sorunlarda beraberinde gelmektedir. Özellikle gelişen teknoloji ve internet ile birlikte hızlı erişilecek büyük verilere ihtiyaç duyulmaya başlanmıştır. Bu verileri depolamak için çeşitli veritabanı sistemleri tasarlanmıştır. Bunlardan en çok bilinenleri Microsoft SQL Server, Oracle Database, MySQL, Microsoft Access gibi sistemlerdir. Bunlar genel olarak SQL dili kullanılarak yönetilirler. Bu sistemlere alternatif olarak son yıllarda ortaya NoSQL (Not Only SQL) kavramı konulmuştur. Geleneksel SQL sistemleri ile NoSQL ilişkisine daha sonra değinilecektir.

Bilgisayarların hayatın ayrılmaz bir parçası haline gelmesi ile birlikte teknolojik gelişmeler hız kazanmıştır. Bu gelişmelerde en çok öne çıkan ise internetin yaygınlaşmasıdır. İnternete, sürekli gelişen ve büyüyen uygulamalar eklenmektedir. Birbirleri ile bir şekilde iletişime geçmek isteyen insanlar internet sitelerine üye olarak resim, video veya kişisel bilgilerini girmekteler. Ürünlerini internet üzerinden tanıtmak isteyen firmalar ürün görsellerini ve bilgilerini yükleyerek müşterilerine daha kolay ulaşmaktadırlar. İnternet forumları dediğimiz siteler sayesinde insanlar akıllarındaki herhangi bir soruyu yazarak çözümünü öğrenebilmektedirler. Bunlar ve bunlar gibi birçok örnek verilebilir. Dolayısıyla internet/bilgisayarlar bir şekilde bilgilerle dolmaktadır. Bilişim dünyasında bu bilgiler "veri" ismini almaktadırlar. Yani gelişen teknoloji ile birlikte elimizdeki her bilgi bizim için bir "veri" olmaktadır.

İnternetin ve teknolojinin gelişmesi ile birlikte eldeki veri ve bu veriler arasındaki ilişki çoğalmaya başlamıştır. Bu noktada verileri doğru bir şekilde yönetebilmek ve onlara ulaşabilmek için Veritabanı Yönetim Sistemleri (Database Management System - DBMS) ve Dağıtık Veritabanları geliştirilmeye başlanmıştır.

Kaliteli sistemler/yazılımlar, belirli bir düzeyde hatasız projeler, gereksinimleri/beklentileri karşılayabilen sistemlerdir. Ancak, kalite terimi oldukça değişken bir ifade olup kullanıcının isteğine ve hedeflenen unsurlara bağlı olarak farklılıklar gösterebilmektedir. Bu bağlamda testler, beklenen ile son durumda oluşan sonuçlar arasındaki farklılıkların belirlenmesini

sağlar. Yazılım testi bir yazılımın uygun şekilde seçilmiş testler ile beklenen işlemleri sağlayıp sağlamadığını kontrol eder.

Geliştirilen yazılımlarda, hataların olması kaçınılmazdır. Çalışılan uygulama alanına göre hatalar ve sonuçlar değişmektedir. Genel olarak uygulamaya bağlı yazılım testleri,

- Ürün kalitesinden emin olmak,
- Masrafları azaltmak,
- Erken aşamalarda hataları belirleyerek ileri aşamalara yayılmasını önlemek,
- Müşteri memnuniyetini arttırmak,

amaçları doğrultusunda yapılmaktadır.

MongoDB Tester uygulamasının asıl amacı; Dağıtık Veritabanlarının yaygınlaşan kullanım alanlarını göz önünde bulundurarak, kullanıcıların daha güvenilir bir sistem kullandıklarını görmeleri için testler sunmaktır.

1.1. Problem Tanımı

Yazılım testleri, geliştirilmiş sistemlerin incelenmesinde ve değerlendirilmesinde hayati bir rol oynar. Hatalı bir sistemi müşteriye sunmak, hem müşterinin hem de geliştiricilerin zor durumda kalmasına sebep olabilir. Bu durumun önemi bilindiğinden İVTYS için hergün yazılım testleri geliştirilmekte ve çoğaltılmaktadır. Ancak Dağıtık Veritabanları için durum farklıdır. Yeni gelişen sistemler olduklarından ve varolan ile işlem yapmanın getirdiği rahatlıktan dolayı, Dağıtık Veritabanı Sistemleri ile ilgili yeterli sayıda ve yetenekte yazılım testleri bulunmamaktadır.

MongoDB Tester uygulamasının amacı, Dağıtık Veritabanları üzerinde bir test mekanizması geliştirerek kullanıcıların sistemlerinde iyileştirme yapmalarını sağlamaktır. Bu doğrultuda, C# programlama dili ve MongoDB üzerinde; NoSQLUnit isimli, Dağıtık Veritabanları için daha önceden geliştirilmiş olan test sistemi referans alınarak testler geliştirilmiştir. NoSQLUnit, içeriği bilinen bir veritabanı üzerinde test yaparken, bu tez ile geliştirilmiş sistem bilinmeyen bir veritabanı üzerinde daha kapsamlı testler yapmaktadır.

2. LİTERATÜR ARAŞTIRMASI

MongoDB Tester ile MongoDB Dağıtık Veritabanı Yönetim Sistemi üzerinde Veritabanı Testleri yaparak, sistem iyileştirmeleri hedeflenmiştir. Bu konu ile ilgili yapılmış birçok çalışma ve geliştirilmiş uygulamalar bulunmaktadır [1], [2], [3]. Geliştirilen testlerin çoğu İlişkisel Veritabanları için olmakla birlikte Dağıtık Veritabanları ile ilgili çok fazla örnek bulunmamaktadır [4], [5], [6]. Veritabanlarını test etmek için geliştirilmiş uygulamalar çoğunlukla Birim Testi kullanmaktadırlar. Günümüzde kullanılan bilindik örnekleri aşağıdaki şekildedir [7]-[12]:

- CUnit, C programlama dilinde birim testleri yazmak ve çalıştırmak için geliştirilmiş bir test sistemidir [13].
- JUnit, Java programlama birim testleri yapan test sistemidir [14].
- Cactus, Java kodları ile geliştirilmiş, sunucu tarafında birim testi yapacak test sistemidir. Sunucu tarafındaki maliyeti düşürmek için geliştirilmiştir [15].
- CppUnit, C++ programlama dili ile geliştirilmiş ve birim testleri gerçekleyen test sistemidir [16].
- Google Test, xUnit tabanlı, C++ programlama dili için geliştirilmiş ve birim testleri gerçekleyen test sistemidir [17].

Dağıtık Veritabanları ölçeklenebilme, esnekli ve daha hızlı çalışma özelliklerine sahip olduğundan bu sistemler içinde literatürde geliştirilen testler bulunmaktadır [18], [19]. Bunlar dışında, Dağıtık Veritabanları için aşağıdaki sebepler nedeniyle testler geliştirilmiştir [20], [21], [22];

- Basit yapılar için hızlı olsalar da karmaşık yapılar üzerinde doğru çalışmayabilirler.
- İlişkisel bir model olmadıklarından, karmaşık sistemlerde sorgu işlemleri oldukça zordur ve bazı durumlarda hatalı sonuçlara ulaştırabilir.
- Hala geliştirilmekte olan bir sistem olduğundan, çok fazla kaynağa sahip değildir.

Ancak Dağıtık Sistemler üzerinde yapılan çok fazla yazılım testi bulunmamaktadır. Bu çalışma için incelenmiş birkaç test aşağıda listelenmiştir [7]-[23];

- NoSQLUnit, Java programı kullanılarak testler geliştirmeyi sağlayan açık kaynak kodlu bir sistemdir. [2], [25].
- Buffalo Software, NoSQL kullanan, Java programlama dili için geliştirilmiş bir test sistemidir.

Geliştirilmiş testlerin projeye katkısı dışında, Yazılım Mühendisliği alanında kullanılan Yazılım Testleri de oldukça önemli bir rol oynamıştır. Bu testlerden aşağıda kısaca bahsedilmiştir;

- Kara-Kutu Testi (Black-Box Testing): Testler gereksinim ve fonksiyonellik üzerinedir. Bu yüzden bu tür testlerde, yazılımın program yapısı, tasarımı veya kodlama tekniği hakkında bilgi sahibi olmaya gerek yoktur. Test ekipleri tarafından en çok kullanılan tekniktir [7], [26], [27].
- Beyaz-Kutu Testi (White-Box Testing): Bu testler ile projenin hem kaynak kodu hem de yürütülebilirliği test edilir. Dolayısıyla kodun koşullarını, alanlarını ve açıklamalarını temel alır [7], [26], [27].
- Birim Testi (Unit Testing): Özel fonksiyonlar veya kod bileşenleri test edilir. Bu testin yapılabilmesi için program kodunun mimarisinin ayrıntılı bir şekilde bilinmesi gerekir [25].
- Regresyon Testi (Regression Testing): Sistemde gerekli ve son değişiklikler yapıldıktan sonra gerçekleştirilen testlerdir. Bu sayede, daha önceki testlerde ortaya çıkan sorunların giderildiğinden ve yeni hatalar yapılmadığından emin olunur [28].
- Performance Testi (Performance Testing): Bu test, "zorlanım" ve "yük" testi ile aynı anlamda kullanılmaktadır. Kodun hangi kısmının sistemin kaynaklarını yönettiğini ve performansı etkilediğini belirler.
- Kullanıcı Kabul Testi (User-Acceptance Testing): Kullanıcıların, uygulamayı kabul etmeden önce, uygulamanın gereksinimlerini ne ölçüde karşılayabildiğini belirleyip, bir sonuca ulaşan testtir.

3. KULLANILAN SİSTEMLER

MongoDB Tester uygulamasının amacı, Dağıtık Veritabanları üzerinde bir test mekanizması geliştirerek kullanıcıların sistemlerinde iyileştirme yapmalarını sağlamaktır. Bunun için Dağıtık Veritabanı Yönetim Sistemleri hakkında detaylı bilgiye sahip olunması gerekmektedir. Bu başlık altında MongoDB Tester'ın geliştirilmesinde kullanılan Dağıtık Veritabanı Sistemleri ve özel olarak seçilen MongoDB Veritabanı hakkında detaylı bilgiler verilmiştir.

3.1. Dağıtık Veritabanı Sistemleri-NoSQL

İnternetin gelişmesi ile birlikte eldeki veri ve bu veriler arasındaki ilişki çoğalmaya başlamıştır. Bu noktada verileri doğru bir şekilde yönetebilmek ve onlara ulaşabilmek için Veritabanı Yönetim Sistemleri (Database Management System) veya İlişkisel Veritabanı Yönetim Sistemleri (Relational Database Management System) geliştirilmeye başlanmıştır. Veritabanı, verilerin bir bütün olarak tutulduğu ve ilişkilerinin incelendiği, içerisinde tablolar bulunan bir alandır. İVTYS geliştiricileri; verileri saklamak, yönetmek için geliştirilmiş, daima verinin doğru ve tutarlı olarak saklanabilmesini hedeflemişlerdir. İVTYS sayesinde veriler diğer tablolardaki sütunlar ile ilişkilendirilebilir ve ilişkilendirilmiş veriler sorgulamalar sayesinde ulaşabilir olur [20], [21].

Veri kullanımını sadece internet aracılığı ile sınırlamamak gerekir. Kullanılan uygulamalar, kullanıcı arabirimleri (UI) ile alınan bilgiler, bilgisayarlara yüklenen resimler, videolar; kindle içerisinde bulunan kitaplar, dergiler, bankalardaki hesaplar, üniversitelerde bulunan öğrenci, akademisyen ve diğer personellerin bilgilerinin tutulduğu otomasyon sistemleri, hatta telefonlardaki numara rehberi dahi birer veridir. Yani veri kullanımı, stoklanması hayatın her alanında karşımıza çıkmaktadır. Dolayısıyla artan kullanım alanı ile birlikte veri sayısı çoğalmakta, karmaşık veri tipleri ortaya çıkmakta ve bu veriler arasındaki ilişkiyi kurmak güçleşmektedir.

İVTYS'ler varlığı kesin olan verileri saklamak, ihtiyaç olduğunda onlara ulaşmak ve onları yönetmek üzere tasarlanmışlardır. Bu sistemler için önemli olan verinin her zaman doğru ve tutarlı bir şekilde saklanabilmesi ve ulaşılabilir olmasıdır. İVTYS'ler işlem tabanlı çalışan ve ACID (Atomicy, Consistency, Isolation, Durability) kuralları ile çalışan sistemlerdir. Tez İVTYS ile ilgili olmadığı için ACID kuralları hakkında detaylı bilgi verilmemiştir.

Peki hızlanan internet ve artan veri sayısına İVTYS'ler ne kadar cevap verebilmektedir? Bir apartmandaki daireleri düşünün. Öyle ki bu apartman eski tip bir apartman olsun. Yani bazı daireler 3 odalı, bazı daireler 5 odalı olabilir. Her daire için odaların özellikleri bir sisteme kayıt edilecek olursa, bazı sorunlar yaşanır. Çünkü dairelerin oda sayılarında tutarsızlıklar bulunmaktadır. Bir başka örnek verilecek olursa; büyük ve aynı zamanda gelişmekte olan bir

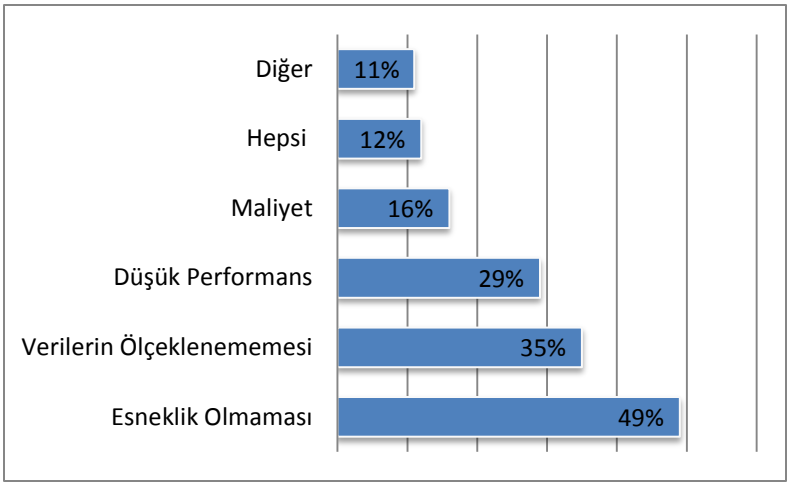
firma olduğunuzu ve her yılsonunda N adet (N her yıl sabit olmamak koşulu ile) çalışanınıza tatil ödülü verdiğinizi varsayın. Ayrıca sisteminiz İlişkisel Veritabanı olsun. Çalışanlarınızın bazılarının bilgilerinde/verilerinde tatil alıp almadıklarına dair bir bilgi bulunması gerekmektedir. Ancak ödülü alan N sayıda eleman kadar, ödülü alamayan M sayıda elemanın da, bu ödülü alamadıklarına dair verilerin sisteme işlenmesi gerekir. Diyelim ki, çalışan bilgilerinizi tuttuğunuz tablonuza her yıl "ODUL" kolonu eklensin ve ödülü alamayanlara NULL değeri girilsin. Dolayısıyla N sayıda eleman için bu tablolarınızda güncelleme yapmanız gerekmektedir. Sayısal olarak düşünürsek, şirketinizde çalışan 1000 tane personel olsun ve bunlardan sadece 10 tanesi tatil ödülü almış olsun – gelecek yıl ödül alanların sayısı 15 olabilir. Yapmanız gereken tablonuza giderek bu 10 kişinin NULL olan bilgilerini güncellemektir. Başka bir örnekle düşünelim. Milyonlarca kişinin kayıt olduğu ve ellerindeki ürünleri satmak istediği bir internet sitemiz olsun. Milyonlarca kullanıcı için binlerce çeşit ürün, bu ürünlerin özellikleri, sınıflandırılması, bu sınıfların alt sınırları şeklinde uzayan bir listeye sahip olursunuz. Ayrıca bir ürünün N tane özelliği olurken bir ürünün M tane özelliği oluyor. Sürekli yeni sınıflar ekleniyor, güncelleniyor. Hatta aynı tipte ürünlerin bile birbirlerinden farklı veya fazla özellikleri olabiliyor. Bu durumda veritabanını yönetmek oldukça zorlaşır. Yani değişen her bilgi için veritabanının sürekli güncellenmesi, yeni kolonlar veya tablolar eklenmesi gerekiyor. Akla ilk gelen, sistemimizdeki ürün sayısının milyonları bulması durumunda yaşanacak performans sıkıntısıdır. Aynı şekilde bir ürüne ulaşmak için yapılacak olan sorgu/filtrelemeler de her seferinde daha zorlu bir hal alacaktır.

Yukarıda verilmiş örneklerde ortaya çıkan sorunlar; verilerin hızlı bir şekilde ve birbirlerinden bağımsız olarak artıyor oluşu, buna bağlı olarak da performans düşüşlerinin yaşanması ve veriye erişimin zorlaşmasıdır. İşte bu noktada bilişim dünyası aşağıda verilen şartları sağlayacak bir sistem arayışına girmiş ve İVTYS'lere alternatif olarak NoSQL (Not Only SQL) kavramını ortaya atmıştır.

i. Çok sayıda veri üzerinde yapılacak okuma/yazmanın eşzamanlı olarak gerçekleştirilememesi,
ii. Artan veriyi doğru ve verimli bir şekilde depolayabilmek,
iii. Yüksek ölçeklenebilirlik ve erişim ancak bunları yapmak için düşük maliyet,
iv. İVTYS'lerde yaşanan yavaş okuma/yazma hızına çözüm,
v. İVTYS'de bulunandan sınırlı kapasitenin değiştirilmesi,
vi. İVTYS'lerin farklı şekilllerdeki verileri yazmakta ve okumakta sorun yaşanması,

olarak sıralanabilir [22], [57], [58].

Üst kısımda bahsedilmiş maddeleri, bir NoSQL sistem olan CouchBase 2011 yılında aşağıdaki istatiksel bilgiyi yayınlayarak neden İVTYS'den farklı başka bir sisteme ihtiyaç olduğunu daha anlaşılır bir şekilde göstermiştir [54].



Şekil 3.1 NoSQL Gerekliliği Nedenleri

NoSQL ilk olarak 1988 yılında, SQL arayüzü olmadan, Açık Kaynaklı İlişkisel Olmayan Dağıtık Veritabanı olarak ortaya atılmıştır. Tasarımcısı Carlo Strozzi isimli geliştiricidir. NoSQL'ın ortaya konma amacı; internet ile birlikte gün geçtikte artan veriyi depolayabilmek, çok fazla işleme sahip sistemlerin ihtiyaçlarına farklı bir ölçekleme ile cevap verebilmek ve yapısal olarak uymayan veriler üzerinde bile işlem yapabilmektir. Ancak o yıllarda İVTYS'lerden vazgeçilememiş, dolayısıyla çok fazla rağbet görmemiştir. Uzun bir süre ilgi çekmedikten sonra, 2009 yılında "Açık Kaynaklı Dağıtık Sistemler" ile ilgili yapılacak bir toplantı ile tekrar gündeme gelmiştir. Bu dönemden itibaren geniş bir kitlenin ilgisini çekmiştir [56], [57].

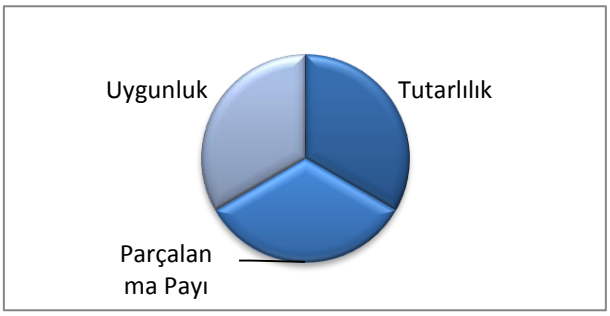
NoSQL ilk okumada SQL değil anlamını çağrıştırmaktadır ve bu durum kafa karışıklıklarına sebep olmaktadır. Kullanımı yine SQL gibidir ancak İVTYS'lerden farklı olarak üretilmiştir. Peki, neden NoSQL denilmesi gereği duyulmuştur? NoSQL'in başka isimleri de mevcuttur. NonRel, NoİVTYS bunlardan belli başlıcalarıdır. Ancak bunlar arasından NoSQL'in en çok tercih edilir olmasının sebebi ticari kaygılardır. SQL değil diyerek yeni üretilen bu teknoloji kullanıcılara etkili bir şekilde sunulmak istenmiştir ve başarılı da olmuştur.

NoSQL'in İVTYS'e göre avantajları aşağıda verilmiştir;

1. Büyük veriler üzerinde işlem yapabilmek. Bu işlemleri replikasyon ve farklı sunuculara dağıtarak daha kullanışlı hale getirmek.
2. Replikasyon sayesinde bir sunucuda aksama yaşansa dahi diğer sunucular ile işleme kullanıcıya hissettirmeden devam edebilme.
3. Yatay ölçeklenebilirliği sayesinde veri girişinde daha esnek olma.
4. Üzerinde yapılacak iyileştirmelerin maliyetinin daha düşük olması.

NoSQL bilinen model dışında – İVTYS – büyük veriyi saklama, veriyi sorgulama, farklı verileri daha kolay şekilde okuma/yazma ve veriler arası ilişkiyi/tutarlılığı daha kolay hale getirme amaçlı ortaya çıkmıştır. Dikey ölçekleme yapılan İVTYS'lerin aksine NoSQL'de Yatay Ölçekleme yapılmaktadır. Bu sayede Dikey Ölçeklemede yapılan fazla iş yükünü çalıştırmak için daha güçlü sunuculara yüksek meblağlar harcamak yerine, daha ucuz sunucularla yatay büyüme gerçeklenebilir. Ayrıca yatayda büyümenin yanında verilerin sistematik olarak dağıtılması yani "Sharding" tekniğinin kullanımı da NoSQL ile gerçekleştirilebilir.

2000 yılında Profesör Eric Brewer Dağıtık Sistemler; "Tutarlılık (Consistency), Müsaitlik (Availability) ve Parçalanma Payı (Partition Tolerance) özelliklerinin üçünü de aynı anda sağlayamazlar" şeklinde bir teori öne sürmüş ve bu teoriye "CAP Teorem" adını vermiştir. Yani bir dağıtık sistemin birimleri aynı veriye erişebilme, aynı anda tüm isteklere cevap verebilme, kaybedilen birimlere rağmen verinin bütününü kaybetmeme özelliklerine aynı anda sahip olamaz. Dolayısıyla NoSQL sistemler yeniden kullanılmaya başlandığında bu kavramlarda göz önünde bulundurularak yatay ölçeklendirme ile performansı geliştirmek hedeflenmiştir [58], [50]. Aşağıdaki şekil CAP Teorem'in dağılımını göstermektedir:



Şekil 3.2 CAP Teorem

NoSQL'in Cap Teorem'e göre işleyişi aşağıdaki gibi gruplamalar yaparak gerçeklenmiştir:

- Tutarlılık – Uygunluk (Consistency - Availability); Veritabanı parçalanma payı ile ilgilenmez, önemli olan verilere erişebilme ve tüm isteklere cevap verebilmektir. Parçalanma payı ile oluşan eksikliği kapatmak için birden fazla sunucu kullanılarak Replikasyon (Replication) yapılır. Bu grubu kullanan bazı veri tabanları; Vertica, Greenplum.
- Tutarlılık – Parçalanma Payı (Consistency – Partition Tolerance); veri bütünlüğünü bozmadan bölerek tutarlar, bu duruma rağmen veriler arası tutarlılığı kolaylıkla sağlarlar. Ancak aynı anda işlem yapılmaya çalışıldığında bazı problemlerle karşılaşırlar. Bu grubu kullanan bazı veri tabanları; MongoDB (doküman tabanlı), Berkeley DB, BigTable.

• Uygunluk – Parçalanma Payı (Availability – Partition Tolerance); bu tip sistemler müsaitliği ve parçalanma payını aynı anda sağlarken tutarlılığı sağmamakta da başarılıdırlar. Bu sistemlere; CouchDB, Voldemort, SimpleDB örnek verilebilir.

Yukarıdaki gruplamalara bir örnekle açıklanırsa; sisteminiz Müsaitlik – Parçalanma Payı grubundan olsun; elinizdeki verileri belirli parçalara bölerek bu parçaların istenilen sayıdaki kopyalarını bir sunucu yardımı ile kullanarak sisteminizin Tutarlılığını da sağlayabilirsiniz [58], [51], [52], [53].

İVTYS'lerin aksine NoSQL'de tablo, tablolar arası ilişkiler yoktur. Bunun yerine verileri JSON, XML veya BSON formatında dokümanlar halinde denormalize biçimde tutar. Dolayısıyla İVTYS'de yapılan yeni bir özellik geldiğinde yeni kolon ekleme ile vakit kaybetmek yerine, JSON (veya XML veya BSON) formatlı dokümanımızda yeni bir parametre oluşturuyoruz. Örneğin, elimizde bir Operating Systems dersinin bilgilerinin tutulduğu bir doküman olsun. Genellikle bu derste 1 vize ve 1 final yapılıyor olsun. Dolayısıyla İVTYS'de tablo buna uygun kolonlar ile hazırlanacaktır. Ancak bir yıl 2 vize ve 1 final yapılmaya karar verildiği durumda tekrar yeni bir kolon eklemek gerekecek ve eski kayıtların 2.vize bilgilerini NULL girmek gerekecektir. Bunun yerine NoSQL kullanarak veriler JSON formatında kayıt edilirse, yeni kalan kayıtlardaki ek bir bilgi yüzünden eski kayıtlarda güncelleme yapmaya gerek kalmayacaktır.

Numara	1.Vize	Final
452	30	50
453	80	75

Tablo 3.1 Örnek Tablo 1

Numara	1.Vize	2.Vize	Final
452	30	NULL	50
453	80	NULL	75
454	45	19	100

Tablo 3.2 Örnek Tablo 2

JSON yazılımı ile ise işlem çok daha anlaşılır ve kolaydır;

{"Numara":452, "1.Vize":30, "Final":50}
{"Numara":453, "1.Vize":80, "Final":50}
{"Numara":454, "1.Vize":45, "2.Vize":19, "Final":50}

Bu sayede tablolara olan mecburiyetimiz ortadan kalkmış olur. [55]

Dolayısıyla NoSQL sistemlerde sabit tablo tanımlamalarına ihtiyaç olmadığından kayıt edilen verilerin kolonları arasında bir tutarlılık olmak zorunda değildir.

NoSQL ile ilgili bir diğer unsur da "İlişkisel olmayan" özelliğidir. İVTYS'lerde performansı en çok düşüren "JOIN" komutları NoSQL içinde kullanılamaz. Çünkü dokümanlar arasında bir ilişki yoktur. Bu yüzden NoSQL bir sistemde birleştirme operasyonu yapmaya çalışmak çok büyük problemlere ve vakit kaybına sebep olur. Bunun sebebi birleştirilmek istenen verilerin farklı parçalarda olma ihtimalidir. Ama yine de İVTYS yüzünden veriler arasında ilişki kurmaya alışık olanlar NoSQL'de de bu özelliği kullanmak isterler ise, ilişkilendirmeyi elle yapmaları gerekmektedir. İlişkilendirme durumu İVTYS sistemlerde ciddi performans kaybına sebep olurken, NoSQL için bu işlemler artacak diğer işlem performanslarının yanında çok küçük bir kayıp olur. Fakat NoSQL'in geliştirilmesinde en büyük amaç "İlişkilendirmeme" olduğu için, birleştirme operasyonu yerine verileri denormalize biçimde tutmakta fayda vardır. Her ne kadar denormalize halinde veriler kendi kendilerini tekrar edecekte olsalar, NoSQL'de performans kaybına sebep olmazlar.

Verilere ulaşabilmek için verilerin nasıl saklandığını bilmekte önemlidir. NoSQL sistemler veriyi çok basit bir mantıkla saklarlar. Bu sistemlerin, veri saklama yöntemi şu şekildedir; her veri için elinde bir anahtar-değer (key-value) çifti bulunuyor ve NoSQL veri saklama/okuma için matematiksel bir algoritma olan Hash Fonksiyonları kullanıyor. Bilindiği gibi Hash Fonksiyonlarının özelliği; değişen uzunluktaki bir giriş değerini alarak, bu değerden tutarlı ve sabit uzunluklu bir çıkış değeri üretmektir. NoSQL'de de bir anahtar-değer çifti kaydedilirken bir Hash Fonksiyonundan geçirilir ve ortaya çıkan Hash/Geri Dönüş değeri verinin kaydedileceği NoSQL veritabanının sunucusunun bilgilerini içerir. Anahtar-değer çifti NoSQL veritabanı sistemi tarafından bu sunucuda saklanır. Bu çiftin okunması gerektiği durumda ise, veriye erişmek isteyen uygulama bu Hash değerini sağlamak zorundadır. NoSQL sistem bu değeri kullanarak verinin saklandığı sunucuyu belirler, sunucuya erişir ve ilgili anahtar-değer çiftini getirir. NoSQL'in çok sayıda veriyi saklayacak olan Dağıtık yapılarda veri yazma/okuma için tercih edilmesinin nedenlerinden biri de bu basit yazma/okuma mantığında saklıdır.

Herkes elindeki verileri bir şekilde saklayabilir/işleyebilir ama bunu iyi bir performans ile gerçekleştirmek, eldeki makinelere/bilgisayarlara iş yükünü doğru şekilde dağıtabilmekten geçer. Bu mantıkla hareket eden, dünyanın en büyük arama motoru Google, 2004 yılında MapReduce ismi verilen bir sistem duyurdular. Bu sistemi geliştirirken, 1960'lı yıllarda geliştirilen fonksiyonel programlamadaki Map ve Reduce fonksiyonlarından esinlendiklerini belirtmişlerdir. Veriler işlenirken bu iki fonksiyon (Map/Reduce) kullanılır. Derinlemesine MapReduce tanımına girmemekle birlikte kısaca açıklayacak olursak, MapReduce; çok büyük veri setleri ile (öyle ki büyüklüğü Terabaytları bulabilecek veriler) yapılacak işlemlerin birden fazla iş birimine dağıtılmasını sağlayan yöntemdir. Bu sayede iş yükü ardışık olarak hafifletilmiş olur. İşte NoSQL sistemlerin bazıları da MapReduce özelliğine

sahiptir. Bu sayede İVTYS'lerde SQL kullanılarak gerçekleştirilecek karmaşık sorguların analizleri, NoSQL'de kolaylıkla gerçekleştirilir.

Bu kadar çok özelliği olan bir sistemin belli şekilde kategorize edilmesi de gerekmektedir. Geliştiriciler bunu bildikleri için, NoSQL'i 4 kategoriye ayırmışlardır. Bu kategoriler aşağıda listelenmektedir [57], [50], [25];

1. Anahtar-Değer Veritabanı (Key Value Databases); Anahtarın değeri bulmada referans olduğu veritabanlarıdır. Anahtar-Değer ifadesindeki değer metin veya binary olarak saklanabilir. Bu veritabanı modeli ölçeklemeler için en uygun olanıdır. REDIS, Flare, MemcacheD gibi dağıtık veritabanları buna örnektir.
2. Doküman Veritabanı (Document Databases); saklanan veriler/dokümanlar kendilerine özgü bir isimle JSOn veya BSON formatında saklanırlar. Verilere ulaşmada en uygun veritabanıdır. MongoDB, CouchDB gibi dağıtık veritabanları buna örnektir.
3. Kolon Veritabanı (Column Databases); Anahtar-Değer veritabanına benzerler. Ancak burada anahtar, kolon, satır veya diğer bilgilerin kombinasyonu şeklinde oluşturulur. HBase, Cassandra, Hypertable buna örnektir.
4. Graf Veritabanı (Graph Databases); Anahtar değerleri belirlenecek bir graf yapısında saklanır. Bu veritabanı ilişkisel veritabanı mantığına en yakın olandır. Bu yüzden ölçeklendirme işlemi oldukça zorlayıcı olabilir. Sesame, Jena veritabanları buna örnektir.

Geleneksel veritabanları da, sistemlerine yukarıda belirtilmiş 4 kategoriden birini seçerek NoSQL özelliği eklenebilmektedir. En büyük veritabanlarından olan Oracle, Anahtar-Değer Veritabanını kullanarak NoSQL desteği vermektedir [20]-[23].

NoSQL veritabanlarının bir diğer özelliği ise Fire&Forget Prensibi ile çalışıyor olmalarıdır. Bu prensibe göre veri sisteme yollanır ancak ne olduğu konusunda bir bilgiye ulaşılamaz. Yani verinin veritabanına kayıt edilmeme ihtimali bulunur. Her ne kadar bu şekilde bahsedilince Fire&Forget kullanımının doğru olmadığı düşünülse de, performans artışına büyük ölçüde katkıları olmaktadır.

NoSQL sistemler avantajlı sistemler olsalar da büyük şirket hesaplamaları, Bankcılık işlemleri gibi işlerin yapılacağı sistemlerde kullanmak problemlere yok açabilir. Veriler arasında gerçekleşecek herhangi bir tutarsızlık şirketleri büyük ölçüde zarara sokabilir. Bunun verine istatiksel hesapların tutulduğu, gerçek zamanlı analizlerin gerçeklendiği, kontrolsüzce ve birbirinden bağımsız olarak büyüyen verilerin tutulacağı sistemlerde NoSQL'i tercih etmek çok daha faydalı olacaktır. NoSQL'in avantajları;

- NoSQL sistemler kullanıcıya esneklik ve yüksek erişim hakkı tanırlar.
- Okuma/yazma açısından çok daha hızlıdırlar.
- Yüzlerce sunucuyu farklı yerlerden birbirleriyle eşleştirerek çalıştırabilir ve bu sayede çok büyük verileri dağıtarak performans artışı sağlayabilirler.
- Esneklikleri sayesinde bakımları için gerekli maliyet oldukça düşüktür.

- Çok çeşitlidirler.
- Birçok sistemi açık kaynaklı ve günümüzün yükselen teknolojisi Bulut Bilişim için oldukça uygundurlar.

NoSQL'in dezavantajları;

- Güvenlik açısından hala yeterli değillerdir.
- Kullandıkları Fire&Forget Prensibi, çok riski olduğu için bazı sektörlere uygun değildir.
- Birçok geliştirici İVTYS'lere alışık olduğu için NoSQL'de bulunmayan ilişkilendirmeye adapte olmakta zorlanmaktadırlar. Bu yüzden bazı İVTYS'ler NoSQL içerisine uyarlanmaya çalışılmakta ancak bu durum daha büyük karışıklıklara sebep olmaktadır [25], [26], [53], [54], [55], [59], [60].

Günümüz uygulama geliştirme süreçleri çok hızlı ve büyük beklentiler içinde olduğundan, NoSQL sistemler ilgiyi yavaş yavaş üzerlerine çekmeye başlamışlardır. Şu anda birçok firma projeleri için NoSQL sistemlere adapte olmaya başlamışlardır bile. Eksikleri olsa da her zaman yeni gelişmelere şans tanımak gerekir.

3.2. MongoDB

MongoDB, 2007 yılında 10Gen tarafından C++ dili ile geliştirilerek kullanıma sunulmuş; açık kaynaklı, doküman tabanlı, Dağıtık – NoSQL bir veritabanıdır. Verileri JSON veya BSON formatında koleksiyonlarda dinamik şemalar halinde saklar. MongoDB içerisinde koleksiyonlar ve bu koleksiyonlar içerisinde dokümanlar bulunur. Burada koleksiyonlar, İVTYS'lerde tablo yapısına, dokümanlar ile her bir tablonun kayıdına karşılık gelmektedir. Ayrıca dokümanlar anahtar/değer bilgileri ile tutulur ve her değere anahtar ismi yardımı ile erişilir [30], [31], [32].

Dağıtık veritabanları arasında en çok tercih edilenlerden biri olan MongoDB, özellikle hız gerektiren; ilişkisel veritabanlarının yavaş kaldığı ve performansın arttırılması gereken zamanlarda, çok büyük uygulamaları oluşturmak ve çalıştırmak için kullanılmaktadır. Kullanım alanları arasında, yüksek hacimli problemler, analiz için kullanılacak büyük verilerin saklanması, caching işlemleri gibi alanlardır [33], [35], [36].

MongoDB; C, C#, Java, PHP gibi birçok dil ile uyumlu bir veritabanıdır. Program içinde çalışması için ihtiyaç duyulan paketlere ulaşıldığı zaman rahatlıkla kullanılabilir [34], [37], [38].

MongoDB'ye, 1.6.0 sürümü ile eklenen özelliklerden biri de Replikasyon/ReplicaSet özelliğidir. Bu sürümü kadar kullanılmakta olan "ReplicaPair/Master-Slave" yerine getirilen ReplicaSet'i, Master-Slave yapısından ayıran temel özellikler "Auto Failover" ve "Automatic Recovery" özelliklerine sahip olmasıdır. ReplicaSet kullanılarak; okuma/yazma işlemleri ayrı sunuculara yönlendirilebilir, veri aynı anda farklı sunucularda bulunacağı için

sürekli yedeklenmiş olur, herhangi bir sunucunun kapanması durumunda, işlemler yapılmaya devam eder; kapanan sunucu tekrar aktifleştiğinde Recovery işlemi başlar ve bunların hepsi otomatik olarak herhangi bir talimata ihtiyaç olmadan gerçekleşir [36], [37].

MongoDB'de Replikasyon özelliğinin yanında Sharding gibi bir özellikte bulunmaktadır. Sharding, büyük veriyi yönetebilmek için parçalara ayırarak farklı sunuculara dağıtmak ve bu parçalar üzerinden işlem yapmaktır. Kısaca veriyi bölmek olarakta isimlendirilebilir. Sharding sayesinde büyük boyutlu koleksiyonlar parçalanarak, performansta iyileşme sağlanmaktadır [39].

Ayrıca MongoDB bir koleksiyon üzerinde arama yaptığınız belirli alanlar için, İVTYS'lerdeki gibi "Indexing/Index" özelliğine sahiptir. Indeks, bir koleksiyon içerisindeki değerlere daha önceden belirlenen bilgileri kullanarak, filtreleme yolu ile erişmeyi sağlar. Bu erişim için; belirlenen bilgilere göre verileri sıralı bir hale getirir, böylece filtreleme işlemi sırasında bir arama algoritması çalıştırarak; sıralı veri içerisinden sonuca erişir.

MongoDB'nin özelliklerini kısaca özetleyecek olursak;

- Doküman temellidir ilişkisel olmayan veritabanıdır,
- Koleksiyonları oluşturmakta kullandığı şema yapısı dinamiktir,
- Replikasyon ile veri güvenliği sağlar,
- Indexing ile istenilen her koleksiyonu indeksleyebilir,
- Otomatik Sharding ile yatayda genişleme sağlar,
- Map/Reduce özelliğine sahiptir,
- JSON veya BSON tipinde veri saklama ve sorgulama ile veriye daha hızlı erişim sağlar,
- GridFS yardımı ile dokümanları dosya sistemi ile saklayabilir [23]-[30].

4. ÖNERİLEN TEST UYGULAMASI-MONGODB TESTER

Bu çalışmada NoSQL Veritabanları üzerinde Testler yapılarak, iyileştirme oranları ve kullanım hakkında bilgi verilmiştir. Testler için bir NoSQL veritabanı olan MongoDB kullanılmış ve kullanıcıların rahatça ulaşarak istedikleri zaman test yapabilmeleri için C# Programlama dili ile bir web sitesi oluşturulmuştur.

NoSQL veritabanları SQL gibi programlama dillerine uyumlu gelmedikleri için, kullanılacak olan NoSQL veritabanına uygun olan yazılımın (driver) projeye eklemesi gerekmektedir. MongoDB, C# ile uyumlu bir driver bulundurmaktadır. Projeyi oluşturduktan sonra, öncelikle MongoDB C# Driver sisteme eklenmiş, bu sayede MongoDB materyalleri kullanılabilir hale gelmiştir.

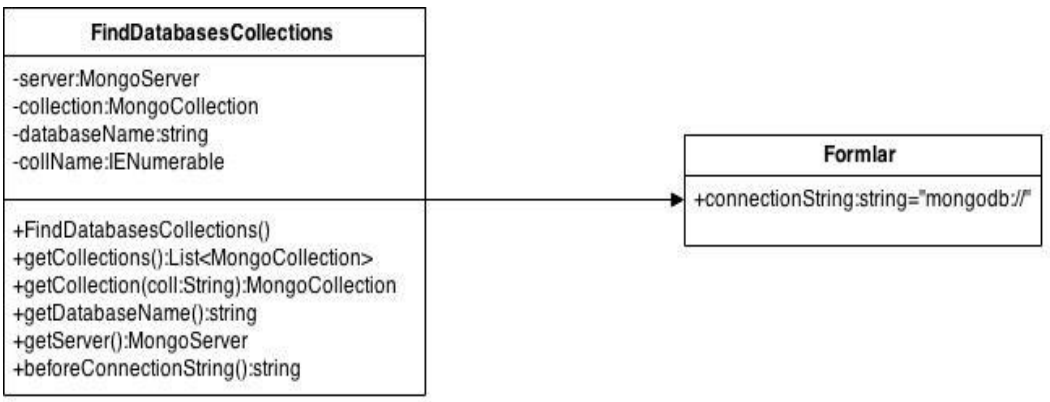
Yazının geri kalanında şu bilgiler verilmiştir; MongoDB Tester uygulamasını geliştirmek için kullanılan sınıf yapıları ve içerikleri hakkında açıklamalar yapılmış, MongoDB Tester'ın içerdiği testler hakkında kapsamlı bilgiler verilmiştir.

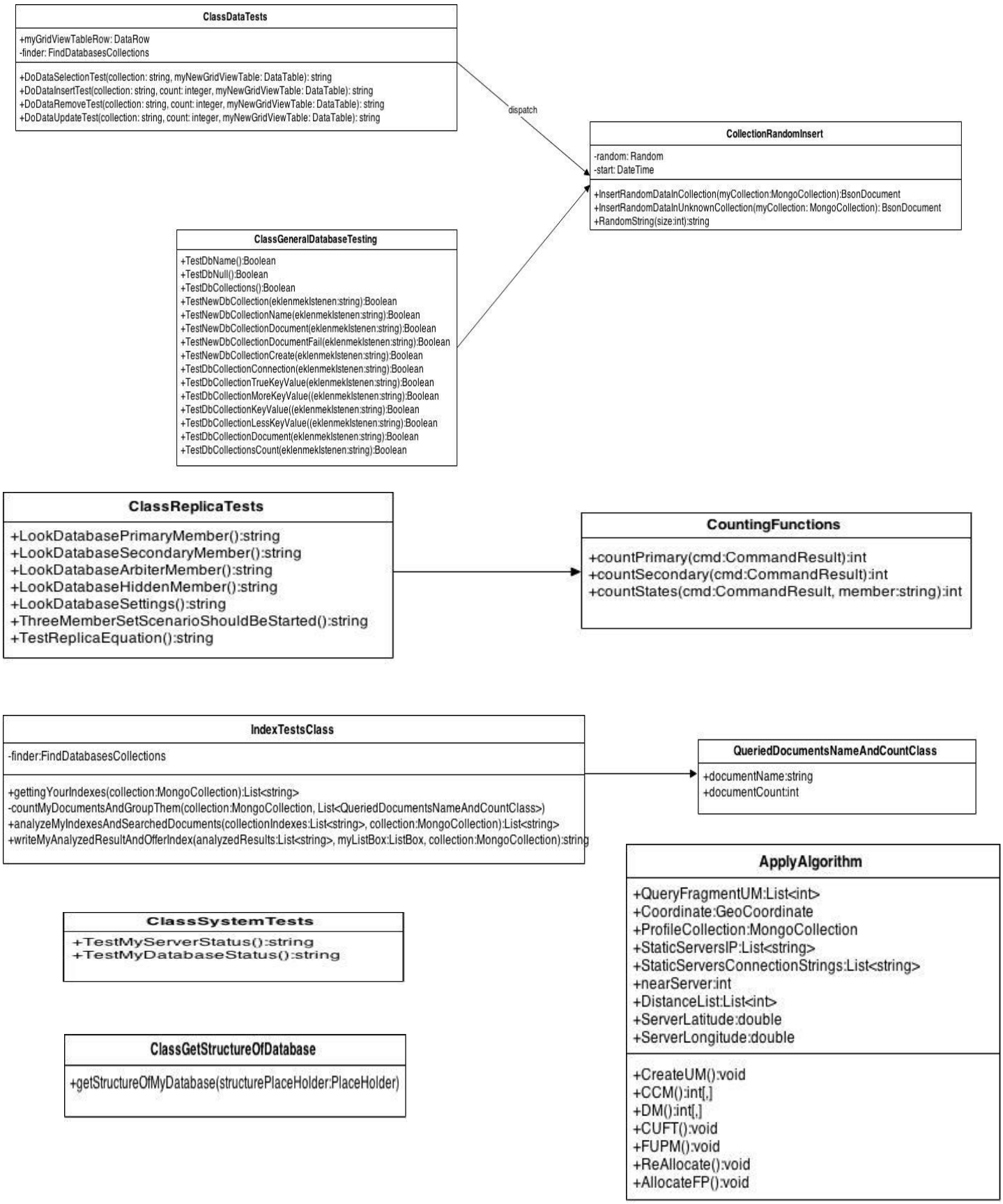
4.1. UML Diyagramlar

Bu bölümde, sistemin geliştirilmesi esnasında kullanılan sınıflar ve genel yapının işleyişini anlatan bilgiler diyagramlar ile gösterilmiştir. Gösterilen tüm diyagramlar UML Diyagramları ile tasarlanmıştır.

4.1.1. Sınıf Diyagramları

Bu bölümde bulunan Sınıf Diyagramları sistemin sınıflarını ve bu sınıflar arasındaki ilişkiyi göstermektedir. Aşağıdaki sınıflar ile ilgili detaylar 4.2. Sınıf Yapıları başlığı altında verilmiştir.



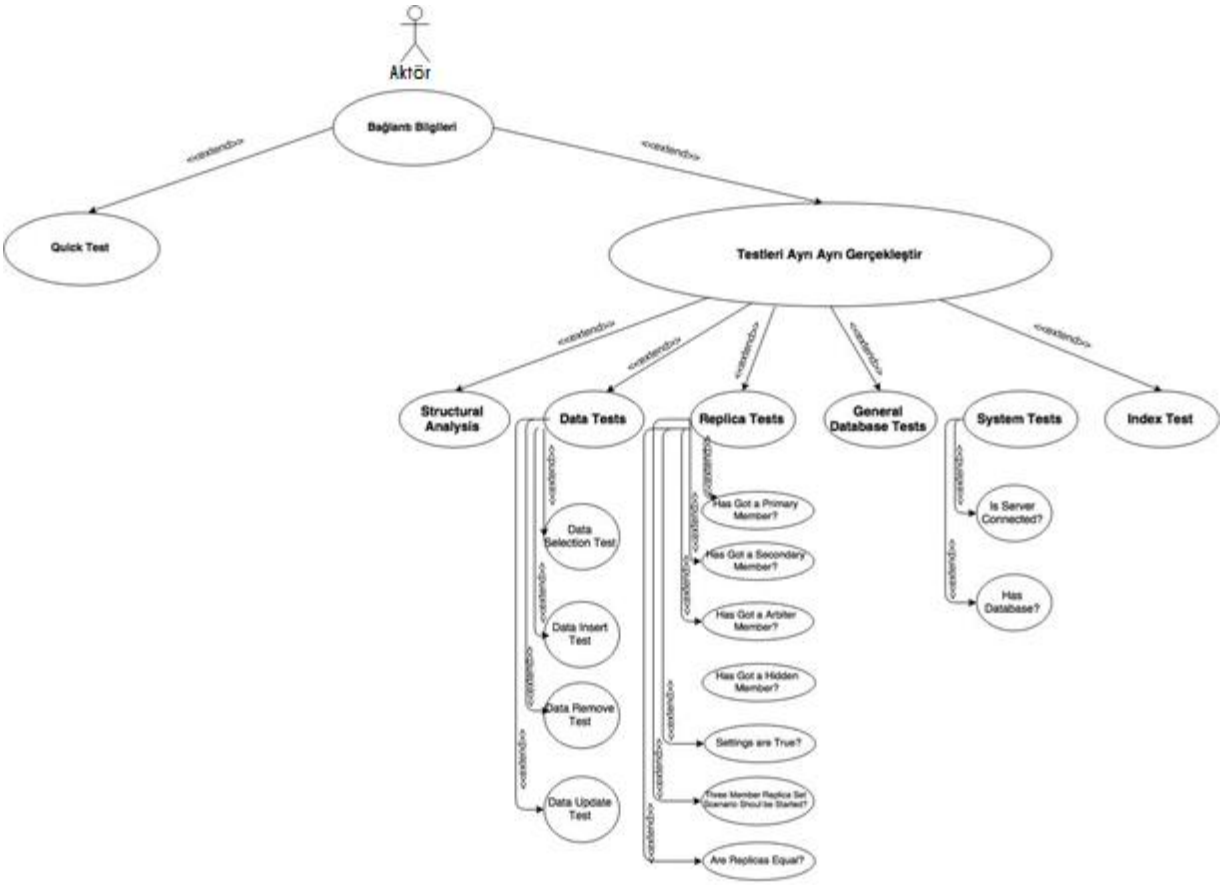


Şekil 4.1 Sınıf Diyagram

4.1.2. Kullanım Senaryosu Diyagramı

UML Kullanım Senaryosu Diyagramları sistemin işleyişinin göstermek için kullanılır. Kullanım Senaryosu diyagramlarında aktörler, sistem, durum ve bunların arasındaki ilişkiler vardır.

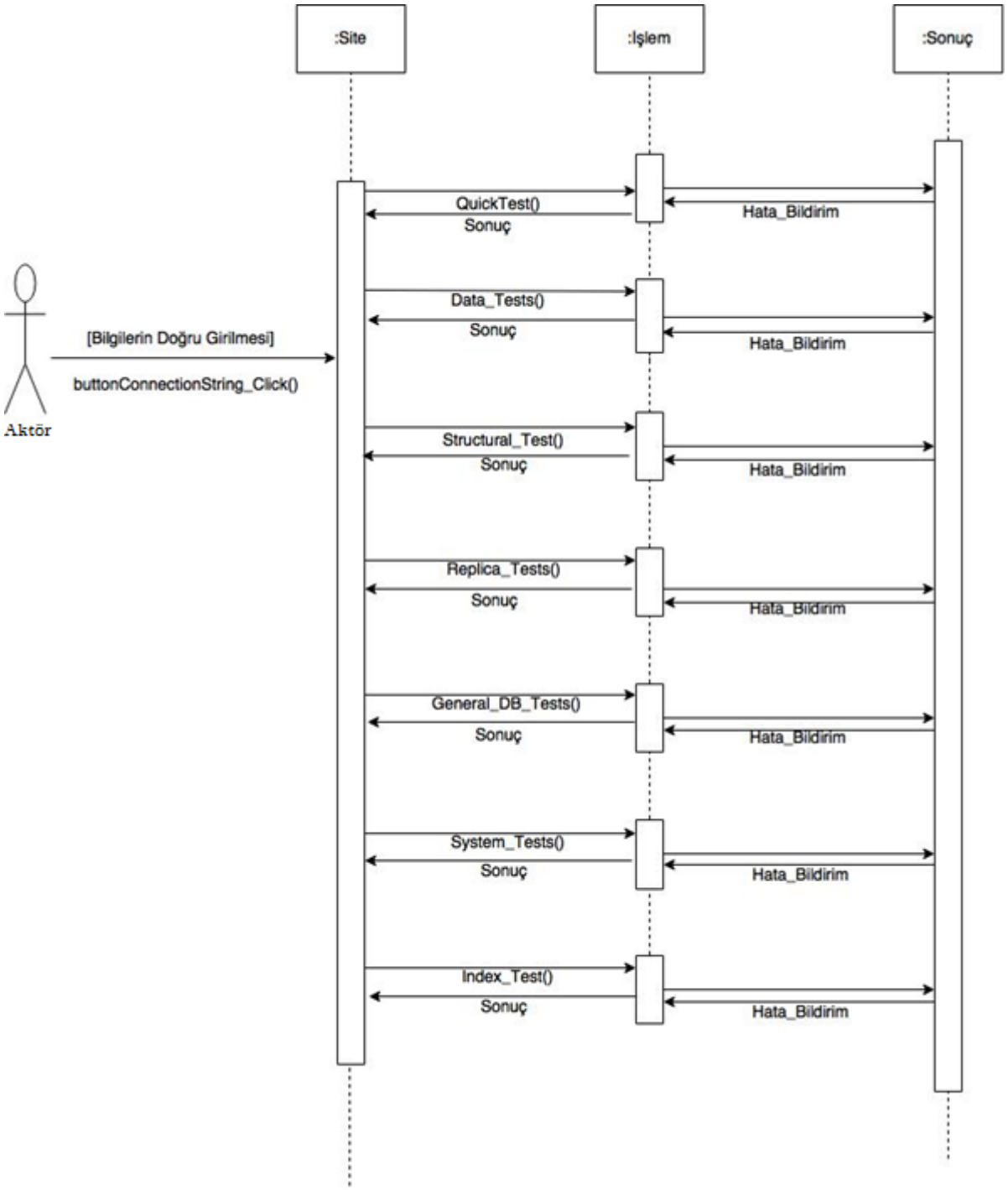
Aşağıda bulunan diyagram geliştirilen sistem ile ilgili işleyişi göstermektedir. Burada Aktör, veritabanını test etmek isteyen kullanıcıdır. Kullanıcı ancak bağlantı bilgilerini doğru bir şekilde girerse sisteme giriş yapabilir. Sisteme giriş yapılması durumunda, kullanıcıya iki seçenek sunulmaktadır. İlki; tüm testleri içeren "Quick Test", diğeri ise kullanıcının isteğine göre tek tek yapılacak olan testlerdir. Testleri ayrı ayrı gerçekleştirmesi durumunda, menüden uygun testi ve gerekli bilgileri seçerek işlemleri gerçekleştirir.



Şekil 4.2 Kullanım Senaryosu Diyagramı

4.1.3. Sıralama Diyagramı

Sıralama Diyagramları, sistemdeki nesneler ve bileşenler arasındaki ileti akışının olaylarını ardışık bir şekilde modellemede kullanılırlar. Aşağıdaki diyagram da sistemdeki akışı göstermektedir.



Şekil 4.3 Sıralama Diyagramı

4.2. Sınıf Yapıları

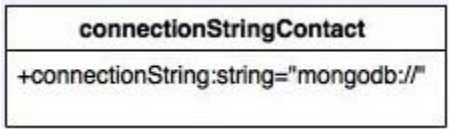
Testlerin gerçeklenebilmesi için sistemde 12 tane sınıf tasarlanmıştır. Bu sınıflardan bazıları testler ile doğrudan bağlantılı iken bazıları da sistem için gerekli diğer işleri yapmaktadırlar. Devam eden kısımda, sınıflar birbirleri ile bağlantılarına göre detaylıca anlatılmış ve içerdikleri UML sınıf diyagramı ile gösterişmiştir.

4.2.1. connectionStringContact Sınıfı

MongoDB veritabanının herhangi bir programlama dili ile çalıştırılması için, düzgünce oluşturulmuş bir bağlantı dizisine (connection string) ihtiyaç vardır. Bu connectionString genel olarak,

mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]] [/[database][?options]]

yapısında olmalıdır. Daha sonra bu bilgi yardımı ile veritabanı üzerindeki koleksiyonlara ve dokümanlara ulaşım sağlanır. Dolayısıyla öncelikle bu bağlantı dizesinin yapısının oluşturulması gerekir. Bağlantı dizesinin düzenli yapısını elde edebilmek ve sayfalar arasında geçiş yapabilmek için Şekil 4.4'de bulunan Formlar isminde bir sınıf tasarlanmıştır.

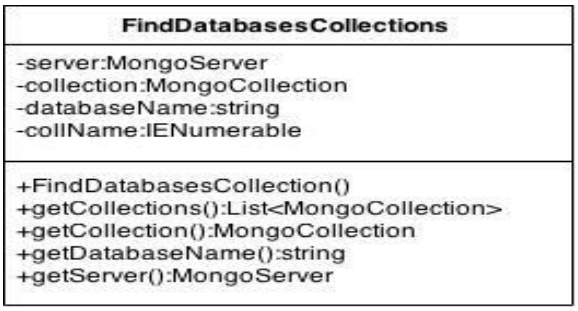


Şekil 4.4 Formlar

connectionStringContact sınıfı içerisinde herhangi bir fonksiyon bulundurmamaktadır. Sadece "connectionString" isminde, ilk değeri "mongodb://" olan bir eleman içermektedir. Ekran üzerinden kullanıcıdan alınan bilgileri bu connectionString elemanı ile aktararak veritabanına ulaşım sağlanmıştır.

4.2.2. FindDatabasesCollestions Sınıfı

FindDatabasesCollections sınıfı tüm testler tarafından kullanılan bir sınıftır. Veritabanına ait bilgilerin bağlantı dizesi yardımı ile elde edilmesini sağlar.



Şekil 4.5 FindDatabasesCollestions

Yukarıda Şekil 4.5 ile gösterilen sınıfta kullanılan fonksiyonlar ile ilgili bilgiler bulunmaktadır;

4.2.2.1. FindDatabasesCollections() Fonksiyonu

FindDatabasesCollections sınıfının yapılandırıcısıdır. Bağlantı dizesini kullanarak veritabanına bağlanır.

MongoClient tipinde *client* adı verilen bir değişken ile sistem sunucuya bağlantı için hazırlanır. (MongoClient –eğer verilmişse bağlantı dizesine uygun olarak, verilmemiş ise localhost içinde bulunan- MongoDB sunucusuna bağlantı için kullanılır). Sunucu içinde bulunan veritabanına bağlanabilmek için veritabanının ismine ihtiyaç olduğundan, *MongoURL.Create()* komutu kullanılarak veritabanı ismine erişilir ve *databaseName* değişkenine atanır. C# üzerinde MongoDB sunucusuna bağlanmak için en çok kullanılan komut *GetServer()* komutudur. Bu yüzden *client* değişkeni ve bu komut yardımı ile veritabanı sunucusuna aşağıdaki şekilde bağlanılır ve sunucu bilgileri *server* isimli değişkene atanır;

server = client.GetServer()

Son adım olarak; veritabanındaki koleksiyon isimlerini *collName* ismindeki global değişkene *GetCollectionNames()* komutu ile aşağıdaki şekilde atanmıştır;

collName = server.GetDatabase(databaseName).GetCollectionNames()

Sonuç olarak bu yapılandırıcıda, veritabanına bağlantı için sunucuyu hazırlama ve sunucu bilgilerini alma, veritabanının ismini belirleme ve veritabanının içerdiği koleksiyonlara ulaşma işlemi gerçekleştirilmiştir.

4.2.2.2. getCollections() Fonksiyonu

Bu fonksiyon List<MongoCollection> tipinde geri dönüş değerlidir. Proje içinde yapılacak bazı testlerde, kullanıcının hangi koleksiyon üzerinde işlem yapılmasını istediği bilgisini belirlemesi gerekmektedir. Bu fonksiyon da, veritabanındaki koleksiyonları bir liste halinde çekerek kullanıcı ekranına getirmemize yardımcı olmaktadır.

Listeleme işlemini gerçekleştirmek için daha önceden içerisine bilgi aktarılmış *collName* isimli global değişkenden faydalanılmıştır. Bu değişken IEnumerable şeklinde bilgileri tuttuğundan, içerdiği bilgilere bir döngü yardımı ile erişilir ve bu sayede koleksiyon isimleri bir listeye eklenebilir. MongoDB içerisinde kullanıcının yarattığı koleksiyonlar dışında Sistem koleksiyonları da bulunmaktadır. Bu koleksiyonlar veritabanında "system" ön eki ile kayıtlıdırlar. Ancak bu fonksiyon ile belirlenecek koleksiyonlar kullanıcı tanımlı oldukları için, sistem koleksiyonlarına ihtiyaç yoktur. Sorgu esnasında bunlar dışındaki diğer koleksiyonlar aşağıdaki şekilde belirlenmiştir;

if (!item.ToString().Contains("system."))

Bu sayede *collName* içerisinde bulunan koleksiyonlar ayrıştırılmış ve sadece kullanıcının tanımladığı koleksiyonlara ulaşılmıştır. Koşulun sağlanması durumunda, *collections* değişkenine koleksiyonlar aşağıdaki şekilde eklenmiştir;

collections.Add(server.GetDatabase(databaseName).GetCollection(item.ToString()))

MongoCollection olarak alınan her koleksiyon *collections* isimli List tipli değişkene eklenir. Tüm işlemler tamamlandıktan sonra geriye bilgi olarak yine *collections* değişkeni gönderilir.

Sonuç olarak bu fonksiyonda; veritabanındaki koleksiyonlar geri dönüş tipi List<MongoCollection> olan bir listeye aktarılmıştır.

4.2.2.3. getCollection() Fonksiyonu

Bazı durumlarda ismi bilinen bir koleksiyonun C# üzerinde kullanılabilmesi için, MongoCollection haline gelmesi gerekmektedir. Bunun için önce sunucuya ulaşılmalı, veritabanı ismi bulunarak, ismi bilinen koleksiyon, *GetCollection()* komutu ile aktif hale getirilmelidir. Bu yüzden, her seferinde aynı işlemleri tekrarlamamak için getCollection() fonksiyonu oluşturulmuştur. Fonksiyon daha öncesinden belirlenmiş sunucu ve veritabanı ismini kullanarak, istenen koleksiyona ulaşır;

collection = server.GetDatabase(databaseName).GetCollection(coll)

ve MongoCollection şeklinde geriye döndürülür.

4.2.2.4. getDatabaseName() Fonksiyonu

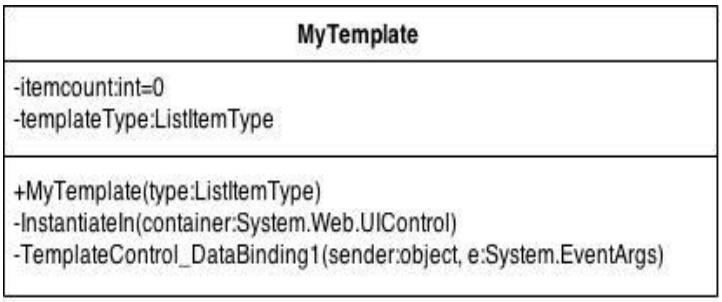
Veritabanı ismine ihtiyaç duyulacak durumlarda kullanılmak üzere tasarlanmış bir fonksiyondur. Global *databaseName* değişkenine aktarılmış veritabanı ismi kullanıcıya gönderilir.

4.2.2.5. getServer() Fonksiyonu

Veritabanı sunucusuna ihtiyaç olabilecek durumlarda kullanılmak üzere tasarlanmış bir fonksiyondur. Global *server* değişkenine atanmış veritabanı sunucusu bilgileri kullanıcıya gönderilir.

4.2.3. MyRepeaterTemplate Sınıfı

MyTemplate sınıfı, kullanıcı ekranına dinamik bir şekilde tablolar ekleyebilmek için tasarlanmış bir sınıftır. İçerisinde HTML kodları kullanılır. Şekil 4.6 ile içerdiği elemanlar gösterilmiştir.



Şekil 4.6 MyRepeaterTemplate

4.2.3.1. MyTemplate(ListItemType templateType) Fonksiyonu

MyTemplate isimli fonksiyon, sınıfın yapılandırıcısıdır. Ana programda belirlenen tablo alanına göre *templateType* elemanına ilk anda değer ataması yapmayı sağlar.

4.2.3.2. InstantiateIn(System.Web.UI.Control container) Fonksiyonu

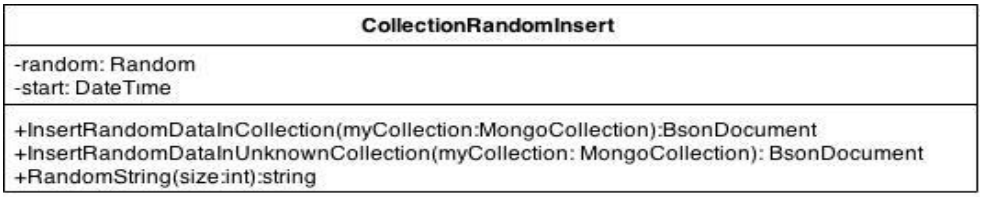
Sınıfa ait bu fonksiyon, HTML kodları ve TemplateControl_DataBinding fonksiyonu yardımı ile oluşturulacak tablonun kullanıcı ekranına yerleştirilmesini sağlar. Bunun için daha önce kullanıcı tarafından tablonun hangi alanı üzerinde işlem yapacağına karar verir ve gerekli gördüğü yerde başka bir fonksiyondan yararlanır.

4.2.3.3. TemplateControl_DataBinding(object sender, System.EventArgs e) Fonksiyonu

Bu fonksiyon, tablonun Item alanı için oluşturulmuştur. *InstantiateIn* fonksiyonu tarafından kullanılır. Fonksiyon tablo için daha önce belirlenmiş olan kolon başlıklarına göre, o kolonlara karşılık gelen satırları uygun elemanlar ile doldurur.

4.2.4. CollectionRandomInsert Sınıfı

Veritabanı üzerinde yapılacak olan bazı testlerde, veritabanı koleksiyonlarına doküman eklemek gerekmektedir. Test için gerekli verileri kullanıcıdan istemek bir seçenek olsa da verileri sistem tarafından üretmek daha yararlı bir test ortamı sağlayacaktır. Şekil 4.7 ile gösterilen CollectionRandomInsert sınıfı, kullanıcının test etmek istediği koleksiyona doküman ekleme işlemlerini yapar.



Şekil 4.7 CollectionRandomInsert

4.2.4.1. InsertRandomDataInCollection(MongoCollection mycollection) Fonksiyonu

Bir koleksiyonun dokümanlarına uygun veriler üreterek, yeni bir doküman ekleyebilmeyi sağlayan fonksiyondur. Parametre olarak *myCollection* isminde üzerinde işlem yapılacak olan koleksiyonu alır. Koleksiyon için veri üretme işlemleri ise aşağıdaki şekilde gerçekleştirilir;

İşlem yapılacak olan koleksiyonun ilk dokümanı elde edilir. Bunun için koleksiyon üzerinde IQuerable bir sorgu gerçekleştirilir. Ardından bu sorgunun ilk dokümanı barındırması durumunda; dokümanın anahtar isimleri .*Names* komutu yardımı ile belirlenir**.** Ve bu anahtar isimleri kullanılarak, dokümanın her değerinin tipi kontrol edilir; her değerin tipine göre veri üretme işlemi gerçeklenir. Üretilen veri, *newKayit* isimli BsonDocument tipindeki değişkene aktarılır. Bu işlemler dokümanın anahtar isimleri tamamlanıncaya kadar devam eder. Tüm anahtar/değer çiftleri için veri üretme işlemi gerçekleştikten sonra, koleksiyon içine newKayit değişkeni kayıt edilir.

4.2.4.2. InsertRDataInUnknownCollection(MongoCollection myCollection) Fonksiyonu

İçinde hiçbir doküman bulunmayan bir koleksiyona doküman üretmek için kullanılan fonksiyondur. Fonksiyon kayıt yapacağı koleksiyonu parametre olarak alır. 3 tane veri üretir ve koleksiyonun içine kayıt edebilmek için *newKayit* isimli BsonDocument tipindeki değişkene aşağıdaki şekilde aktarılır;

- newKayit.Add("firstMember", random.Next(1, 1028)); anahtar ismi “firstMember”, değeri tamsayı olan veri.

- newKayit.Add("secondMember", RandomString(10)); anahtar ismi “secondMember”, değeri karakter katarı olan veri.
- newKayit.Add("thirdMember", start.AddDays(random.Next((DateTime.Today - start).Days))); anahtar ismi “thirdMember”, değeri DateTime olan veri.

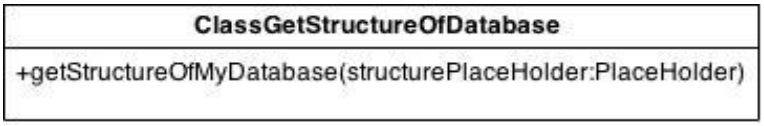
Ardından üretilen(ve anahtar/değer bilgileri geliştirici tarafından belirlenmiş olan) veri koleksiyona eklenir.

4.2.4.3. RandomString(int size) Fonksiyonu

Bir programlama dili, sayısal verileri rastgele üretebilir ancak bir karakter katarı üretemez. CollectionRandomInsert sınıfında bazı durumlarda karakter katarı da üretilmesi gerekebilir. Bu fonksiyon yardımı ile karakter katarı üretme işlemi gerçekleştirilir. Fonksiyon, karakter katarının boyunun ne kadar olacağını belirten *size* isimli parametreyi alır, bir döngü yardımı ile ASCII kodları üretir ve bunları karaktere çevirerek StringBuilder sınıfından bir nesneye atar.

4.2.5. ClassGetStructureOfDatabase Sınıfı

Structure Test için oluşturulmuş sınıftır. Testi yapan kişiye kullanıcıya veritabanının yapısını, yani, koleksiyonları, dokümanları ve bu dokümanların veritabanındaki tipleri hakkında bilgi vermeyi sağlar.



Şekil 4.8 ClassGetStructureOfDatabase

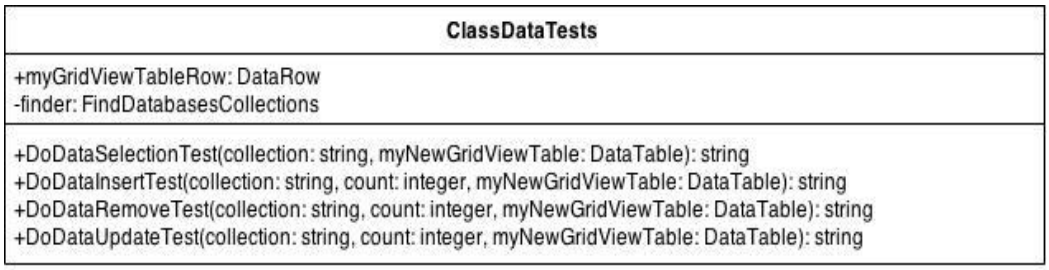
Şekil 4.8 ile gösterilen ClassGetStructureOfDatabase sadece 1 tane fonksiyon içerir. Aşağıda bu fonksiyon hakkında kısaca bilgi verilmiş, ayrıntılı açıklama Testler başlığı altında yapılmıştır.

4.2.5.1. getStructureOfMyDatabase(PlaceHolder PlaceHolder1) Fonksiyonu

Fonksiyon; üzerinde test yapılan veritabanındaki koleksiyonların dokümanlarının anahtar isimleri ve değer tiplerini, her koleksiyon için bir tablo halinde kullanıcıya gösterir.

4.2.6. ClassDataTests Sınıfı

Veri Testleri için geliştirilmiştir. Bu sınıf ile veritabanındaki koleksiyonlar için; veri ekleme, veri silme, veri güncelleme ve veri listeleme işlemleri kullanılarak testler gerçekleştirilmiştir. Sınıfın fonksiyonlarının yaptığı işler Testler başlığı anlatında detaylıca anlatılmış olup Şekil 4.9 ile genel yapısı gösterilmiştir.



Şekil 4.9 ClassDataTests

4.2.6.1. DoDataSelectionTest(String collection, DataTable myNewGridViewTable) Fonksiyonu

Seçilen koleksiyon içerisindeki verileri okuma süresini belirlemek için oluşturulmuş fonksiyondur. Fonksiyonun içine parametre olarak; üzerinde işlem yapılacak koleksiyon bilgisi ve verilerin listeleneceği bir DataTable gelir. Gelen koleksiyon içerisinden veriler okunur ve işlemler sonucunda da okunma süresi kullanıcı ekranında veriler ile birlikte gösterilir.

4.2.6.2. DoDataInsertTest(String collection, int count, DataTable myNewGridViewTable) Fonksiyonu

Seçilen koleksiyon içerisinde, belirlenen sayıda veriyi eklemek için tasarlanmıştır. Fonksiyonun içine parametre olarak; üzerinde işlem yapılacak koleksiyon bilgisi, kaç tane veri üzerinde işlem yapılacağı ve verilerin listeleneceği bir DataTable gelir. Koleksiyonun içerisine listelenecek elemanlar rastgele, CollectionRandomInsert sınıfı kullanılarak üretilir ve eğer doğru bir şekilde ekleme gerçekleştirilir ise, eklenen veriler ile birlikte ekleme süresi kullanıcı ekranında gösterir.

4.2.6.3. DoDataRemoveTest(String collection, int count, DataTable myNewGridViewTable) Fonksiyonu

Seçilen koleksiyon içerisinden, belirlenen sayıda veriyi silmek için tasarlanmıştır. Fonksiyonun içine parametre olarak; üzerinde işlem yapılacak koleksiyon bilgisi, kaç tane veri üzerinde işlem yapılacağı ve verilerin listeleneceği bir DataTable gelir. Koleksiyonun

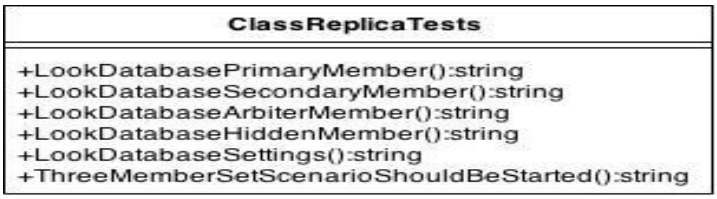
içerisinde silinecek elemanlar önce; rastgele, CollectionRandomInsert sınıfı kullanılarak üretilir daha sonra da üretilen elemanlar silinir, bu sayede veritabanının orijinal verileri üzerinde bir değişiklik yapılmamış olur. Eğer verileri silme işlemi doğru bir şekilde gerçekleştirilir ise, silinen veriler ile birlikte silme süresi kullanıcı ekranında gösterilir.

4.2.6.4. DoDataUpdateTest(String collection, int count, DataTable myNewGridViewTable) Fonksiyonu

Seçilen koleksiyon içerisinde, belirlenen sayıda veriyi güncellemek için tasarlanmıştır. Fonksiyonun içine parametre olarak; üzerinde işlem yapılacak koleksiyon bilgisi, kaç tane veri üzerinde işlem yapılacağı ve verilerin listeleneceği bir DataTable gelir. Koleksiyonun içerisinde güncellenecek elemanlar önce; rastgele, CollectionRandomInsert sınıfı kullanılarak üretilir daha sonra da üretilen elemanlar güncellenir, bu sayede veritabanının orijinal verileri üzerinde bir değişiklik yapılmamış olur. Eğer verileri güncelleme işlemi doğru bir şekilde gerçekleştirilir ise, güncellenen veriler ile birlikte güncellenme süresi kullanıcı ekranında gösterilir.

4.2.7. ClassReplicaTests Sınıfı

Dağıtık Veritabanlarının en önemli özelliklerinden bir tanesinin de replikasyondur. Bir çok dağıtık sistem kullanıcısının veritabanlarını replikasyon ile oluşturması durumunda, replikasyon ile ilgili testlerin de yapılması gerekmektedir. ClassReplicaTests, Replikasyon Testleri için tasarlanmış bir sınıftır ve eğer veritabanı replikasyon yapıyor ise, sistem üyelerinin (Birincil, İkincil, Hakem, Saklı) varlığını ve doğru şekilde oluşturulup oluşturulmadığını kontrol eder.



Şekil 4.10 ClassReplicaTests

Şekil 4.10 ile gösterilen ClassReplicaTests sınıfı için 6 tane fonksiyon oluşturulmuştur, bu fonksiyonlardan her biri sistemin üyelerinin varlığını kontrol eder. Fonksiyonlardan detaylı olarak Testler başlığı altında bahsedilmiştir;

4.2.7.1. LookDatabasePrimaryMember() Fonksiyonu

Bir replika kümesinin en önemli üyesi Birincil (Primary) elemandır. Dolayısıyla bir sistemi test ederken ilk önce, Birincil elemanın varlığının kontrol edilmesi gerekir. LookDatabasePrimaryMember() fonksiyonu ile Birincil elemanın varlığı kontrol edilmiştir.

Fonksiyon herhangi bir parametre almadan, veritabanının tamamını kontrol eder. Ve belirli işlemler sonucunda Birincil eleman bulunması durumunda Doğru/True değerini bilgi olarak döndürür.

4.2.7.2. LookDatabaseSecondaryMember() Fonksiyonu

Bir diğer önemli üye, İkincil (Secondary) elemandır. LookDatabaseSecondaryMember() fonksiyonu ile İkincil elemanın varlığı kontrol edilmiştir.

Fonksiyon herhangi bir parametre almadan, veritabanının tamamını kontrol eder. Ve belirli işlemler sonucunda İkincil eleman bulunması durumunda Doğru/True değerini bilgi olarak döndürür.

4.2.7.3. LookDatabaseArbiterMember() Fonksiyonu

Bir diğer önemli üye, Hakem (Arbiter) elemandır. LookDatabaseSecondaryMember() fonksiyonu ile Hakem elemanın varlığı kontrol edilmiştir.

Fonksiyon herhangi bir parametre almadan, veritabanının tamamını kontrol eder. Ve belirli işlemler sonucunda Arbiter eleman bulunması durumunda Doğru/True değerini bilgi olarak döndürür.

4.2.7.4. LookDatabaseHiddenMember() Fonksiyonu

Bir diğer önemli üye, Sakli (Hidden) elemandır. LookDatabaseSecondaryMember() fonksiyonu ile İkincil elemanın varlığı kontrol edilmiştir.

Fonksiyon herhangi bir parametre almadan, veritabanının tamamını kontrol eder. Ve belirli işlemler sonucunda Sakli eleman bulunması durumunda Doğru/True değerini bilgi olarak döndürür.

4.2.7.5. LookDatabaseSettings() Fonksiyonu

Replika kümesini oluşturan üyeler dışında, kullanıcı oluşturduğu sistemin bilgilerinin doğruluğunu da kontrol etmek gerekir. LookDatabaseSettings() fonksiyonunu ayarların doğruluğunun kontrolü için geliştirilmiştir.

Fonksiyon, replika kümesinin ulaşılan bilgilerini kullanıcı ekranına listeleyerek, testi yapan kişiye bilgilerin doğru olup olmadığını sorar. Yapılan işlem her ne kadar test değişmiş gibi görünse de, kullanıcı sistemini istediği gibi kurup kuramadığını kontrol etmiş olur.

4.2.7.6. ThreeMemberSetScenarioShouldBeStarted() Fonksiyonu

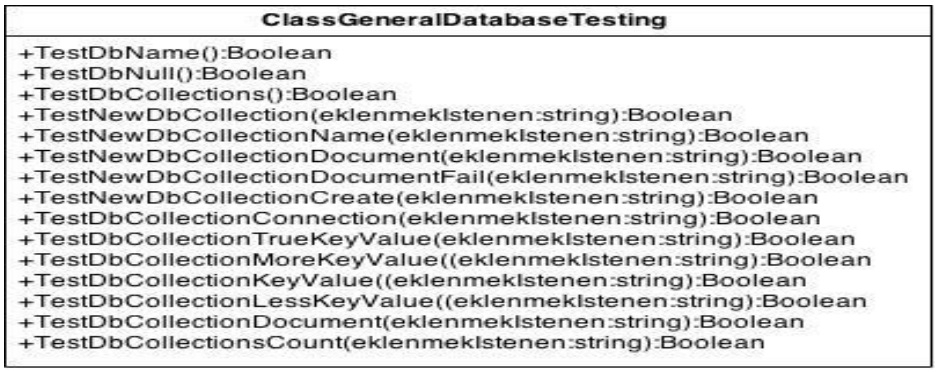
Kullanıcıların oluşturdukları koleksiyonlar dışında MongoDB içerisinde sabit olarak bulunan ve hazır gelen veritabanlarından ve koleksiyonlarında vardır. Bu fonksiyon kullanıcının oluşturduğu veritabanı dışında kalan veritabanını yani, "admin" veritabanını kontrol etmek için tasarlanmıştır. CountingFunctions sınıfını da kullanarak, "admin" veritabanında bulunan Birincil ve İkincil üye sayısını kullanıcıya bildirir.

4.2.7.7 TestReplicaEquation() Fonksiyonu

Kullanıcı tarafından tanımlanmış olan replika kümelerinin tutarlılığını kontrol etmek için tasarlanmış bir fonksiyondur. Bu sayede replika kümelerine kayıt edilen koleksiyonların/dokümanların doğru bir biçimde kayıt edilip edilmediği kontrol edilmiş olur.

4.2.8. ClassGeneralDatabaseTesting Sınıfı

Veritabanı içerisinde kullanıcı tanımlı koleksiyonları da kullanarak, aşağıda açıklaması verilen testleri yapar. Testlerin sonuçları True/False şeklinde bilgiler olarak alınır. Ayrıca Şekil 4.11 ile gösterilen CollectionRandomInsert sınıfı bu sınıfın Base Class'ıdır. Burada kullanılan fonksiyonlar aşağıda açıklanmakla birlikte, her biri veritabanını test eden bir fonksiyon olduğundan, açıklamalarında Test olarak bahsedilmişlerdir.



Şekil 4.11 ClassGeneralDatabaseTesting

4.2.8.1. TestDbName() Fonksiyonu

Veritabanının içeriğinin kontrol edildiği bir testtir. Bu test ile, veritabanının içinin boş olup olmadığı belirlenir; önemli olan sonuç, veritabanı içinin boş olmasıdır. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getDatabaseName()* fonksiyonu kullanılarak veritabanı ismine ulaşılır. İsmin null değerde olması kullanıcıya True bilgisi gönderilir.

4.2.8.2. TestDbNull() Fonksiyonu

Veritabanının içerdiği, kullanıcı tanımlı koleksiyonları kontol eder. Bu test ile, veritabanının koleksiyon içerip içermediği belirlenir; önemli olan sonuç, veritabanının koleksiyon içermemesidir. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getDatabaseName()* fonksiyonu kullanılarak veritabanı ismine ulaşılır. İsmin null değerde olması testing başarısız olduğu yani bir veritabanı içermediği anlamına gelir. Ancak null değerden farklı olması durumunda, veritabanının koleksiyon içerip içermediği kontrol edilir; bu işlem için yine FindDatabasesCollections sınıfının *getCollections()* fonksiyonu kullanılır ve gelen koleksiyonların bilgileri *collName* isimli List<MongoCollection> tipindeki değişkene atanır. Ardından bu değişkenin herhangi bir bilgi bulundurup bulundurmadığı *.Any()* fonksiyonu yardımı ile kontrol edilir. Bilgi içermemesi durumunda test başarılı olmuş ve istenen sağlanmıştır, dolayısıyla True bilgisi gönderilir; bilgi içermesi durumunda ise test başarısız olmuştur.

```
if (!collName.Any())
    return true;
else
    return false;
```

4.2.8.3. TestDbCollections() Fonksiyonu

Veritabanının içerdiği, kullanıcı tanımlı koleksiyonları kontrol eder. Bu test ile, veritabanının koleksiyon içerip içermediği belirlenir; önemli olan sonuç, veritabanının koleksiyon içermesidir. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getDatabaseName()* fonksiyonu kullanılarak veritabanı ismine ulaşılır. İsmin null değerde olması testing başarısız olduğu yani bir veritabanı içermediği anlamına gelir. Ancak null değerden farklı olması durumunda, veritabanının koleksiyon içerip içermediği kontrol edilir; bu işlem için yine FindDatabasesCollections sınıfının *getCollections()* fonksiyonu kullanılır ve gelen koleksiyonların bilgileri *collName* isimli List<MongoCollection> tipindeki değişkene atanır. Ardından bu değişkenin herhangi bir bilgi bulundurup bulundurmadığı *.Any()* fonksiyonu yardımı ile kontrol edilir. Bilgi içermesi durumunda test başarılı olmuş ve istenen sağlanmıştır, dolayısıyla True bilgisi gönderilir; bilgi içermemesi durumunda ise test başarısız olmuştur.

```
if (!collName.Any())
    sonuc = false;
else
    sonuc = true;
```

4.2.8.4. TestNewDbCollection(String eklenmekIstenen) Fonksiyonu

Veritabanının içerdiği, kullanıcı tanımlı koleksiyonları kontrol eder. Bu test ile, kullanıcının eklemek istediği yeni koleksiyonun, daha önceden tanımlı olan koleksiyonlar arasında var olup olmadığı belirlenir; önemli olan sonuç, veritabanında aranan koleksiyonun bulunmamasıdır. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getDatabaseName()* fonksiyonu kullanılarak veritabanı ismine, *getServer()* fonksiyonu kullanılarak sunucuya, *getCollections()* fonksiyonu kullanılarakta kullanıcı tanımlı koleksiyonlara ulaşılır. getCollections() fonksiyonu ile elde edilen koleksiyon isimleri *collName* isimli bir değişkene aktarılır. Ardından koleksiyon isimleri ile, eklenmekIstenen isimli ve veritabanına yeni eklenecek olan koleksiyon bilgisi kıyaslanır; herhangi bir koleksiyon isminin bu bilgiyle aynı olması durumunda test başarısız olmuş, yani sisteme yeni eklenmek istenen koleksiyon ismine sahip bir koleksiyonun daha önceden veritabanında tanımlanmış olduğu bilgisine ulaşılır ve False bilgisi gönderilir, tüm koleksiyon isimleri ile aynı olmaması durumunda ise test başarılı olur, yani yeni eklenecek koleksiyon ismi veritabanında daha önceden bulunmamaktadır ve geriye bilgi olarak True değeri gönderilir.

```
if (item.ToString().Equals(eklenmekIstenen)) return false;
```

4.2.8.5. TestNewDbCollectionName(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu kontrol eder. Bu test ile, kullanıcının eklemek istediği, daha önceden veritabanında tanımlı olmayan, yeni bir isimle eklenecek olan koleksiyonun; doğru bir şekilde veritabanında tanımlanıp tanımlanmadığı kontrol edilir; önemli olan sonuç, yeni koleksiyon isminin doğru bir şekilde tanımlanmasıdır. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen yeni koleksiyon veritabanına koleksiyon olarak eklenir ve *newCollection* isimli MongoCollection tipindeki değişkene atanır. Ardından veritabanında yeni oluşturulan koleksiyonun ismi ile eklenmek üzere gönderilen ismin birbirine eşit olup olmadığı kontrol edilir.

if (!newCollection.Name.Equals(eklenmekIstenen)) sonuc = false;

isimlerin eşit olmaması durumunda koleksiyon sisteme doğru eklenememiştir, bu durumda bilgi olarak False değeri gönderilecektir. Eşit olmaları durumunda ise işlem başarı ile gerçekleşmiştir ve True değeri gönderilir. Tüm işlemler tamamlandıktan sonra, veritabanına sonradan eklenen koleksiyon, asıl düzeni bozmamak için veritabanından silinir.

newCollection.Drop();

4.2.8.6. TestNewDbCollectionDocument(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu kontrol eder. Bu test ile kullanıcının eklemek istediği, yeni koleksiyona kayıt edilmek istenen dokümanların doğru bir şekilde kayıt edilip edilmediği kontrol edilir; önemli olan sonuç, yeni koleksiyon içerisine dokümanların doğru bir şekilde eklenmiş olmasıdır. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen yeni koleksiyon veritabanına koleksiyon olarak eklenir ve *newCollection* isimli MongoCollection tipindeki değişkene atanır. Ardından, CollectionRandomInsert() sınıfında bulunan ve içerisinde hiçbir doküman olmayan bir koleksiyona kayıt yapabilmek için hazırlanmış olan *InsertRandomDataInUnknownCollection()* fonksiyon kullanılarak koleksiyona bir tane doküman eklenir ve bu dokümanın bilgileri *lastData* ismi verilen BsonDocument tipteki değişkende tutulur. Koleksiyona eklenen dokümanın anahtar/değer isimlerinin doğru bir biçimde eklenebildiğini kontrol etmek için, koleksiyona son eklenen(koleksiyonda tek bulunan) doküman koleksiyondan çekilir ve newCollectionDoc isimli değişkene aktarılır. Bu dokümanın anahtar isimlerine ulaşmak için *.Names* isimli komuttan faydalanılır. Bu komut ile dokümanın anahtar isimlerine ulaşılır ve newCollectionDocKeys isimli değişkene atanır.

IEnumerable newCollectionDocKeyNames = newCollectionDoc.Names;

Son adım olarak, eklenen dokümanın anahtar isimleri, koleksiyona eklenmiş veriyi tutan lastData bilgisinin anahtar isimleri ile kıyaslanır. İsimlerden birbirine uymayan çiftler olması

durumunda False bilgisi geri gönderilir, bu koleksiyona eklenen verinin anahtar isimlerinin yanlış bir şekilde eklendiğini gösterir. Aksi durumda ise True bilgisi gönderilir. Tüm işlemler tamamlandıktan sonra, veritabanına sonradan eklenen koleksiyon, asıl düzeni bozmamak için veritabanından silinir.

newCollection.Drop();

4.2.8.7. TestNewDbCollectionDocumentFail(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu kontrol eder. 1.1.8.6. başlığındaki teste çok benzer. Bu test ile kullanıcının eklemek istediği, yeni koleksiyona kayıt edilmek istenen dokümanın doğru bir şekilde kayıt edilip edilmediği kontrol edilir. Eğer kayıt doğru bir şekilde gerçekleşmemiş ise, sistem *fail* olmalıdır.

4.2.8.8. TestNewDbCollectionCreate(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu kontrol eder. Bu test ile, kullanıcının eklemek istediği, daha önceden veritabanında tanımlı olmayan, yeni bir isimle eklenecek olan koleksiyonun; doğru bir şekilde veritabanında eklenip eklenilmediği kontrol edilir; önemli olan sonuç, yeni koleksiyonun veritabanında doğru bir şekilde eklenmiş olmasıdır. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen yeni koleksiyon veritabanına koleksiyon olarak eklenir ve *newCollection* isimli MongoCollection tipindeki değişkene atanır. Ardından veritabanında yeni oluşturulacak koleksiyonun gerçekten oluşturup oluşturulamadığı kontrol edilir. Eğer newCollection bilgisi null ise koleksiyon oluşturulamamıştır, bu durumda False değeri gönderilir; newCollection bilgisi null değil ise koleksiyon oluşturulmuş yani işlem başarı ile gerçekleşmiştir ve geriye bilgi olarak True gönderilir.

if (newCollection == null) sonuc = false;

Tüm işlemler tamamlandıktan sonra, veritabanına sonradan eklenen koleksiyon, asıl düzeni bozmamak için veritabanından silinir.

newCollection.Drop();

4.2.8.9. TestDbCollectionConnection(String eklenmekIstenen) Fonksiyonu

Veritabanında var olan koleksiyonu kontrol eder. Bu test ile kullanıcının daha önceden veritabanında tanımladığı koleksiyon üzerinde işlem yapılıp yapılamayacağı belirlenir. Bunun için koleksiyonun veritabanındaki varlığı ve isminin doğruluğu kontrol edilir. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen koleksiyon veritabanında

aktifleştirilir ve *colEklenmekIstenen* isimli MongoCollection tipindeki değişkene atanır. Ardından veritabanından aktifleştirilen koleksiyonun gerçekten var olup olmadığı veya var ise isminin doğru şekilde alınma durumları kontrol edilir. Eğer colEklenmekIstenen bilgisi null ise koleksiyon oluşturulamamıştır veya colEklenmekIstenen koleksiyonunun ismi ile eklenmekIstenen parametresiyle gönderilen koleksiyon ismi uyumlu değil ise koleksiyon doğru aktifleştirilememiştir, bu durumda False değeri gönderilir; aksi durumda geriye bilgi olarak True gönderilir.

if (newCollection == null) return false;

4.2.8.10. TestDbCollectionTrueKeyValue(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu kontrol eder. Bu test ile kullanıcının üzerinde işlem yapmak istediği koleksiyona tek bir tane dokümanın kayıt edilip edilmediği kontrol edilir. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen koleksiyon veritabanında aktifleştirilir *collection* isimli MongoCollection tipindeki değişkene atanır. Ardından, koleksiyondaki tüm dokümanların sayısı bulunur. Bunun için önce koleksiyonun tüm kayıtları sorgulanır ve query isimli değişkene atanır, bu sorgudaki kayıt sayısı .*Count* fonksiyonu kullanılarak bulunur ve o da *firstLength* isimli değişikene atanır.

int firstLength = query.Count();

CollectionRandomInsert() sınıfında bulunan ve içerisine gönderilen koleksiyonun dokümanlarına uygun olarak yeni veriler üretmek için olan *InsertRandomDataInCollection()* fonksiyonu kullanılarak koleksiyona bir tane doküman eklenir. Koleksiyona tek bir doküman eklendiğini kontrol etmek, koleksiyonda tüm dokümanların sayısı bulunur tekrar bulunur ve bu sayı lastLength isimli değişkene aktarılır.

int lastLength = lastQuery.Count();

lastLength isimli değişkende, koleksiyonda son durumda bulunan doküman sayısını tutar. Bu sayı koleksiyona ekleme yapmadan önceki doküman sayısını tutan firstLength sayısı ile kıyaslanır. Aralarındaki farkında 1' den farklı olması durumunda False değeri gönderilir, bu durumda koleksiyona birden fazla doküman eklenmiş yani test başarısız olmuştur.

4.2.8.11. TestDbCollectionMoreKeyValue(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu test eder. 1.1.8.10. başlığındaki teste çok benzer. Veritabanına ait bir koleksiyona eklenen bir verinin, anahtar/değer sayısının koleksiyondakinden fazla olması durumunu test eder. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen koleksiyon veritabanında aktifleştirilir ve *collection* isimli

MongoCollection tipindeki değişkene atanır. Ardından, içerisinde doküman bulunan bir koleksiyon için veri üretme işlemini gerçekleyecek *InsertRandomDataInCollection()* fonksiyonu kullanılarak koleksiyona bir tane doküman eklenir ve bu dokümanın bilgileri *lastData* ismi verilen BsonDocument tipteki değişkende tutulur. Koleksiyona eklenen dokümanın anahtar/değer isimlerinin doğru sayıda eklenebildiğini kontrol etmek için, koleksiyonda bulunan ilk doküman ve son eklenen doküman koleksiyondan çekilir ve *firstDataInCollection* ve *lastDataInCollection* isimli değişkenlere aktarılır.

```
BsonDocument lastDataInCollection = query.Last<BsonDocument>();
BsonDocument firstDataInCollection = query.First<BsonDocument>();
```

Bu dokümanların anahtar isimlerine ve sayısına ulaşmak için *.Names* ve *.Count*i komutlarından faydalanılır. Ve ulaşılan sayı değerleri ilk doküman için *fnameLength*, son doküman için *lnameLength* isimli değişkenlere aktarılır.

```
int lnameLength = lastDataInCollection.Names.Count();
int fnameLength = firstDataInCollection.Names.Count();
```

Son adım olarak son eklenen dokümanın anahtar/değer sayısı ile ilk dokümanın anahtar/değer sayısı kıyaslanır. Ve son dokümandaki sayının daha az olması durumunda False bilgisi gönderilir. Bu durumda koleksiyona eksik bilgileri bulunan bir doküman eklenmiş olur.

4.2.8.12. TestDbCollectionKeyValue(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu test eder. 1.1.8.11. başlığındaki teste çok benzer. Veritabanına ait bir koleksiyona eklenen bir verinin istenen biçimde kayıt edilmesi durumunu test eder. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen koleksiyon veritabanında aktifleştirilir ve *collection* isimli MongoCollection tipindeki değişkene atanır. Ardından, CollectionRandomInsert() sınıfında bulunan ve içerisinde doküman bulunan bir koleksiyon için veri üretme işlemini gerçekleyecek *InsertRandomDataInCollection()* fonksiyonu kullanılarak koleksiyona bir tane doküman eklenir ve bu dokümanın bilgileri *lastData* ismi verilen BsonDocument tipteki değişkende tutulur. Koleksiyona eklenen dokümanın anahtar/değer isimlerinin doğru bir biçimde eklenebildiğini kontrol etmek için, koleksiyona son eklenen doküman koleksiyondan çekilir ve collectionDoc isimli değişkene aktarılır. Bu dokümanın anahtar isimlerine ulaşmak için *.Names* isimli komuttan faydalanılır. Bu komut ile dokümanın anahtar isimlerine ulaşılır ve collectionDocKeys isimli değişkene atanır.

```
IEnumerable collectionDocKeyNames = collectionDoc.Names;
```

Son adım olarak, eklenen dokümanın anahtar isimleri, koleksiyona eklenmiş veriyi tutan lastData bilgisinin anahtar isimleri ile kıyaslanır. İsimlerden birbirine uymayan çiftler olması durumunda False bilgisi geri gönderilir, bu koleksiyona eklenen verinin anahtar isimlerinin yanlış bir şekilde eklendiğini gösterir. Aksi durumda ise True bilgisi gönderilir.

4.2.8.13. TestDbCollectionLessKeyValue(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu test eder. 1.1.8.12. başlığındaki teste çok benzer. Veritabanına ait bir koleksiyona eklenen bir verinin, anahtar/değer sayısının koleksiyondakinden az olması durumunu test eder. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen koleksiyon veritabanında aktifleştirilir ve *collection* isimli MongoCollection tipindeki değişkene atanır. Ardından, içerisinde doküman bulunan bir koleksiyon için veri üretme işlemini gerçekleyecek *InsertRandomDataInCollection()* fonksiyonu kullanılarak koleksiyona bir tane doküman eklenir ve bu dokümanın bilgileri *lastData* ismi verilen BsonDocument tipteki değişkende tutulur. Koleksiyona eklenen dokümanın anahtar/değer isimlerinin doğru sayıda eklenebildiğini kontrol etmek için, koleksiyonda bulunan ilk doküman ve son eklenen doküman koleksiyondan çekilir ve *firstDataInCollection* ve *lastDataInCollection* isimli değişkenlere aktarılır.

```
BsonDocument lastDataInCollection = query.Last<BsonDocument>();
BsonDocument firstDataInCollection = query.First<BsonDocument>();
```

Bu dokümanların anahtar isimlerine ve sayısına ulaşmak için *.Names* ve *.Count*i komutlarından faydalanılır. Ve ulaşılan sayı değerleri ilk doküman için *fnameLength*, son doküman için *lnameLength* isimli değişkenlere aktarılır.

```
int lnameLength = lastDataInCollection.Names.Count();
int fnameLength = firstDataInCollection.Names.Count();
```

Son adım olarak son eklenen dokümanın anahtar/değer sayısı ile ilk dokümanın anahtar/değer sayısı kıyaslanır. Ve son dokümandaki sayının daha fazla olması durumunda False bilgisi gönderilir. Bu durumda koleksiyona fazla bilgileri bulunan bir doküman eklenmiş olur.

4.2.8.14. TestDbCollectionDocument(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu test eder. 1.1.8.12. başlığındaki teste çok benzer. Bu test ile, kullanıcının üzerinde işlem yapmak istediği koleksiyona kayıt edilmek istenen dokümanın gerçekten eklenip eklenmediği kontrol edilir. Bizim için önemli olan dokümanın koleksiyon içinde var olmasıdır. Test işlemini gerçeklemek için, FindDatabasesCollections sınıfının *getCollection()* fonksiyonu kullanılarak eklenmekIstenen isimli parametre ile gelen

koleksiyon veritabanında aktifleştirilir ve *collection* isimli MongoCollection tipindeki değişkene atanır. Ardından, CollectionRandomInsert() sınıfında bulunan ve içerisinde doküman bulunan bir koleksiyon için veri üretme işlemini gerçekleyecek *InsertRandomDataInCollection()* fonksiyonu kullanılarak koleksiyona bir tane doküman eklenir ve bu dokümanın bilgileri *lastData* ismi verilen BsonDocument tipteki değişkende tutulur. Koleksiyona eklenen dokümanın doğru bir biçimde eklenebildiğini kontrol etmek için, koleksiyona son eklenen doküman koleksiyondan çekilir ve collectionDoc isimli değişkene aktarılır.

BsonDocument collectionDoc = collection.AsQueryable<BsonDocument>().Last()

Ulaşılan bu doküman ile koleksiyona kayıt edilmek için üretilmiş veri, yani lastData değerinin eşit olup olmadığı kontrol edilir. Eşit olmaları durumunda True değeri gönderilir, bu durum da koleksiyona veri doğru bir şekilde eklenmiş herhangi bir değer kaybı yaşanmamıştır.

4.2.8.15. TestDbCollectionsCount(String eklenmekIstenen) Fonksiyonu

Veritabanına eklenmek istenen koleksiyonu test eder. Bu test ile, kullanıcının veritabanına eklediği koleksiyonun birden fazla kez (ve farklı şekillerde) eklenip eklenmediği belirlenir. Test için FindDatabasesCollections sınıfının *getCollections()* fonksiyonu kullanılarak veritabanındaki kullanıcı tanımlı olan tüm koleksiyonlara ve sayısına erişir. Bu sayı firstLength isimli değişken ile tutulur.

int firstLength = finder.getCollections().Count;

Ardından, eklenmekIstenen parametresi ile gönderilmiş ve yeni eklenecek koleksiyonun bilgisini içeren koleksiyon *getCollection()* fonksiyonu kullanılarak oluşturulur. Ve tekrar veritabanındaki kullanıcı tanımlı koleksiyonların sayısına ulaşılır. Bu sayı da lastLength isimli değişkende tutulur.

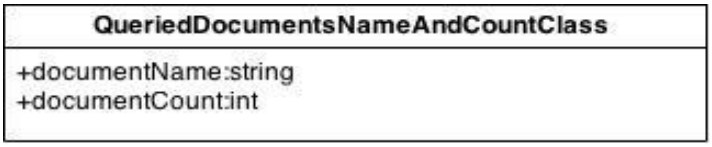
int lastLength = finder.getCollections().Count;

Son adım olarak bu iki sayı arasındaki farkın 1 olup olmadığı kontrol edilir. 1 olmaması durumunda False bilgisi gönderilir. Bu durumda veritabanına eklenen koleksiyon bir kereden fazla sayıda sisteme eklenmiştir. Aksi durumda ise True bilgisi gönderilir.

if (lastLength - firstLength != 1) return false;

4.2.9. QueriedDocumentsNameAndCountClass Sınıfı

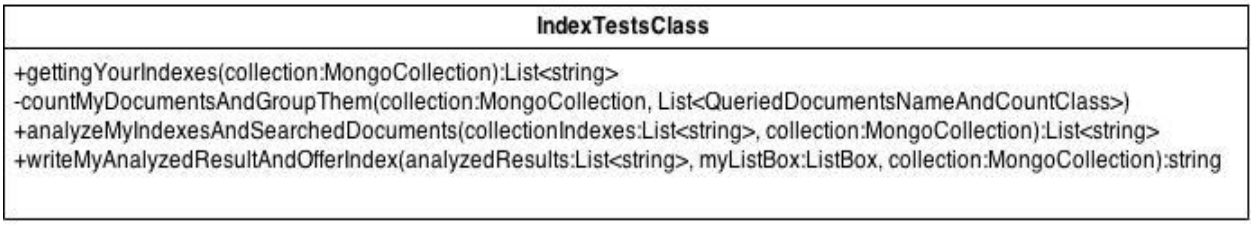
İndeksler üzerinde yapılacak test için yardımcı sınıf olarak tasarlanmıştır. Şekil 4.12 ile gösterilen sınıfn tasarlanma amacı; veritabanı içerisinde, bir koleksiyona ait anahtar/değer bilgisinin ismini ve kaç kez sorgulandığı bilgisini bellekte tutmaktır. 1.1.10. IndexTestsClass başlığında indeksler ile ilgili testler detaylıca anlatılmıştır.



Şekil 4.12 QueriedDocumentsNameAndCountClass

4.2.10. IndexTestsClass Sınıfı

Dağıtık Veritabanlarının İVTYS'den bir diğer farklarının, tablolar arası ilişki kurma özelliğine sahip olmamalarıdır. Buna karşılık bazı dağıtık sistemler opsiyonel çözümler



Şekil 4.13 IndexTestsClass

sunmuşlardır.
Üzerinde test yapılan, MongoDB veritabanı, indeksleme yöntemi ile veritabanı işlemlerini hızlandırmayı amaçlamıştır.

Testin yapılabilmesi için, "system.profile" koleksiyonunun aktifleştirilmiş olması gerekmektedir. Bu sınıf "system.index" ve "system.profile" koleksiyonlarını kullanarak, test edilen veritabanının indekslerini belirler ve eğer gerekli görülür ise yeni indeks önerilerinde bulunur.

Şekil 4.13 ile gösterilen sınıf, 4 tane fonksiyon içermektedir. Bu fonksiyonlar kullanılarak, Index Testing gerçekleştirilmiştir. Fonksiyonlar aşağıda detaylıca anlatılmıştır.

4.2.10.1. gettingYourIndexes(MongoCollection collection) Fonksiyonu

MongoDB içerisinde gömülü olarak gelen bazı koleksiyonlar vardır. Bunlara genel olarak "system" koleksiyonlarıdır.

IndexTestsClass ile veritabanının indeksleri kullanılarak test yapılmıştır. Veritabanından seçilen koleksiyonu, *collection* isminde bir değişkeni parametre olarak alan bu fonksiyon, koleksiyonun var olan indekslerini belirleyebilmek için tasarlanmıştır. Fonksiyon genel olarak; "system.index" koleksiyonunda veritabanına ait koleksiyonların indekslerini okur ve bir liste halinde bildirir.

MongoDB'de bulunan "system.index" koleksiyonu hakkındaki bilgiler şu şekildedir; koleksiyonun içerisinde bulunan her dokümanda 4 tane anahtar/değer çifti bulunmaktadır. Bu anahtar/değer çiftinden, "ns" sistemdeki koleksiyon ismini, "name" koleksiyon içerisindeki indeks isimlerini, "key" doküman şeklinde olup; içerisinde anahtar olarak indeks isimlerini ve değer olarak da bu indekslerin özelliklerini içermektedir. Dolayısıyla anahtarlar kullanılarak istenilen koleksiyonun indekslerine ulaşılabilir.

"system.index" MongoDB'nin bir koleksiyonu olduğu ve C# içerisinde kullanılabilmesi için aktifleştirilmesi gerektiğinden, bu koleksiyonu sisteme uygun şekilde kullanmak gerekir. Bunun için FindDatabasesCollections sınıfının getCollection() fonksiyonu kullanılarak, *indexCollection* isminli MongoCollection tipinde bir değişkende "system.index" koleksiyon bilgisi tutulmuştur;

```
FindDatabasesCollections finder=new
FindDatabasesCollections(Formlar.connectionString);
 MongoCollection indexCollection = finder.getCollection("system.indexes");
```

Bu sayede koleksiyon içerisinde bir sorguyu gerçekleştirilebilir. Ancak sorgu yapmadan önce dikkat edilmesi gereken bir nokta vardır; inceleme yapılacak olan koleksiyonun, yani collection ile gelen koleksiyon bilgisinin, indeks sayısı. Bilindiği gibi, "_id" her MongoDB koleksiyonunda otomatik olarak tanımlı gelen bir indekstir, yani, koleksiyonun indeks sayısının 1'den farklı olması durumunda sorgu ve diğer işlemlerin gerçekleştirilmesi gerekir. Bunu yapabilmek için, *GetStats()* komutu kullanılır. Bu komut ile ulaşılan bilgiler kullanıcı tarafından seçilmiş olan koleksiyonun, indeks sayısı bilgisini içerir. Tamsayı olarak indeks sayısının elde edilmesi için *IndexCount* komutu kullanılır. Bu iki komutunda kullanımı aşağıdaki şekildedir;

```
var status = collection.GetStats();
                    int indexCount = status.IndexCount;
```

indexCount değerinin 1'den farklı olması durumunda aşağıdaki işlemlerin gerçekleştirilir;

“system.index” koleksiyonunda yapılacak olan sorguda, kontrol edilmesi gereken iki kriter vardır; ilki hangi koleksiyonun indekslerine ihtiyaç olduğu, bu sorunun cevabı için “system.index” içerisindeki her dokümanın “ns” anahtar/değer bilgisi kontrol edilmelidir. Bir diğeri ise, “name” bilgisinin yani koleksiyonun indeks isminin “_id_” veya “_id” değerini içermemesidir. “_id” her MongoDB koleksiyonunda otomatik olarak tanımlı gelen bir indekstir, bu yüzden kullanıcının ihtiyacına göre tanımlanmış olmaz. Sistemin çalışabilmesi için “_id” dışındaki indekslere ihtiyaç vardır. Dolayısıyla aşağıda verilen sorgu bu kriterlere uygun şekilde gerçekleştirilmiştir;

```
var query =
indexCollection.AsQueryable<BsonDocument>().Where(p=>p["ns"].Equals(collection.ToString()) & !p["name"].Equals("_id_"));
```

Sorgu IQueryable<BsonDocument> şeklindedir. Yapılan sorgu ile gelecek sonuçlar BsonDocument şeklinde olduğundan, anahtar isimlerine ulaşmak oldukça kolaydır; buradan sonra sorgu içerisindeki her bir dokümanın, “key” anahtar bilgilerinde bulunan indeks isimlerine ulaşılır.

Sorgu içindeki her bir dokümana ulaşmak için *foreach* döngüsünü kullanılır.

```
foreach (var item in query) {}
```

query değişkenindeki her bir doküman BsonDocument şeklinde olduğundan “key” anahtarı kullanılarak, bulundurduğu değere ulaşılır. Burada hatırlanması gereken nokta, “key” bilgisinin de koleksiyonda bir doküman şeklinde kayıtlı olduğudur. Sorgunun içerisinden gelen her dokümana ait, “key” anahtar/değer çiftinin bilgileri BsonDocument şeklindeki *keyAsBson* değişkenine aktarılır;

```
BsonDocument keyAsBson = new BsonDocument();
keyAsBson = item["key"].ToBsonDocument();
```

Ve keyAsBson, koleksiyonun(collection) indeks bilgilerini; anahtar bilgisini indeks ismi, değeri ise indekslerin tanımlanma bilgisi şeklinde içerir. Bir BsonDocument’in anahtar bilgilesini öğrenebilmek için kullanılması gereken komut *.Names* komutudur(*keyAsBson.Names*). Dolayısıyla bu komut yardımı ile keyAsBson değişkeninde olan anahtar bilgilerine yani indeks isimlerine ulaşılır. Koleksiyonun(collection) indekslerinin birden fazla olması durumunda, keyAsBson değişkeninin anahtar bilgileri de birden fazla olacağı için, bu anahtar bilgileri bir döngü yardımı ile çekilir. Ve her anahtar bilgisi yani indeks ismi daha önceden tanımlanmış olan *collectionIndexes* değişkenine atanır.

```
foreach (var keyAsBsonName in keyAsBson.Names) {
    collectionIndexes.Add(keyAsBsonName.ToString());}
```

Bu işlemler query bilgisi sonlanana kadar gerçekleştikten sonra, koleksiyonun(collection) indekslerinin isimleri bir liste halinde elde edilmiş olur.

4.2.10.2. countMyDocumentsAndGroupThem(MongoCollection collection) Fonksiyonu

"system.profile" Koleksiyonu yardımı ile veritabanından seçilen koleksiyonu, *collection* isminde bir değişkeni parametre olarak alan bu fonksiyon; veritabanı içerisinde yapılan sorgularda kullanılan dokümanların anahtar/değer bilgilerini ve bunların kaç kez kullanıldığını yani sayısını belirler. Bu fonksiyonu kullanabilmek için yeni bir sınıf olan *QueriedDocumentsNameAndCountClass* sınıfı da tanımlanmıştır.

Fonksiyonun içinde yapılan işlemler "system.profile" koleksiyonunda gerçeklenir. bu yüzden koleksiyon hakkında kısaca bilgi verilmiştir koleksiyonun içerisinde birden fazla anahtar/değer çifti bulunmaktadır. Bu anahtar/değer çiftinden kullanılacak olanlar, "ns" sistemdeki koleksiyon ismini, "op" koleksiyonun içerisinde yapılmış olan ekleme(insert), silme(remove), güncelleme(update), arama sorguları(query) gibi operasyonların bilgilerini, "query" sorgulanmış anahtar/değer çifti bilgilerini içermektedir. Şu da unutulmamalıdır ki, "op" anahtarına karşılık gelen değeri query olan bir doküman, "query" isminde başka bir doküman içerir ve bu "query" dokümanı, $query isminde bir bilgi ile koleksiyon(collection) içerisinde sorgulanmış olan anahtar/değer çiftine ulaştırır. (system.profile koleksiyonu hakkında detaylı bilgi, MongoDB başlığı altında verilmiştir.) Dolayısıyla anahtarlar kullanılarak istenilen koleksiyonun indekslerine ulaşılabilir.

"system.profile" MongoDB'nin bir koleksiyonu olduğu ve C# içerisinde kullanılabilmesi için aktifleştirilmesi gerektiğinden, bu koleksiyonu sisteme uygun şekilde kullanmak gerekir. Bunun için FindDatabasesCollections sınıfının getCollection() fonksiyonu kullanılarak, *profileCollection* isminli MongoCollection tipinde bir değişkende "system.profile" koleksiyon bilgisi tutulmuştur;

```
FindDatabasesCollections finder=new
FindDatabasesCollections(Formlar.connectionString);
MongoCollection profileCollection = finder.getCollection("system.profile");
```

Bu sayede koleksiyon içerisinde bir sorgu gerçekleştirilir.

"system.profile" koleksiyonunda yapılacak olan sorguda, kontrol edilmesi gereken iki kriter vardır; ilki hangi koleksiyonun sorgulanmış anahtar/değer çiftine ihtiyaç olduğu, bu sorunun cevabı için "system.profile" içerisindeki her dokümanın "ns" anahtar/değer bilgisi kontrol edilmelidir. Bir diğeri ise, "op" bilgisinin yani koleksiyon içerisinde yapılmış olan operasyonun "query" şeklinde olmasıdır. Bu kriter kullanılarak, "system.profile" koleksiyonu içerisinden sadece "query" bilgisi çekilir. Dolayısıyla aşağıda verilen sorgu bu kriterlere uygun şekilde gerçekleştirilmiştir;

var query = profileCollection.AsQueryable<BsonDocument>().Where(p =>
p["op"].Equals("query") & p["ns"].Equals(collection.ToString())).Select(q => q["query"]);

sorgu IQueryable<BsonDocument> şeklindedir. Yapılan sorgu ile gelecek sonuçlar BsonDocument şeklinde olduğundan, anahtar isimlerine ulaşmak oldukça kolaydır; buradan sonra sorgu içerisindeki her bir dokümanın, "$query" anahtar bilgileri içerisinde bulunan indeks isimlerine ulaşılır.

Sorgu içindeki her bir dokümana ulaşmak için *foreach* döngüsünü kullanılır.

foreach (var item in query) {}

"system.profile" koleksiyonu incelendiğinde bazı dokümanlarının "op" ile gelen "query" bilgisi $query değerine sahip olmamasıdır, bunun sebebi; MongoDB içerisinde yapılan her işlemin, adım adım "system.profile" içine yazılmasıdır. Sorgunun içerisindeki dokümanların, $query bilgisini içermesi aşağıdaki kontrol işlemi ile gerçekleştirilir;

if (item.ToString().Contains("$query"))

bu şartın sağlanmasıyla, keyAsBson ismili BsonDocument tipindeki değişkene, sorgunun içerisinden gelen her dokümana ait, "$query" anahtar/değer çiftinin bilgileri aktarılır.

BsonDocument keyAsBson = new BsonDocument();
keyAsBson = item["$query"].ToBsonDocument();

Burada keyAsBson, koleksiyonun(collection) içerisinde yapılan sorgularda kullanılan anahtar/değer çiftlerini içeren bir bilgidir. Yani anahtar sorgulanan dokümandaki bilginin ismi, değer de o anahtarın değeridir. Bir BsonDocument'in anahtar bilgilerini öğrenebilmek için kullanılması gereken komut *Names* komuttur(*keyAsBson.Names*). Dolayısıyla bu komut yardımı ile, keyAsBson değişkenindeki anahtar isimlerine ulaşılır. Koleksiyonun(collection) içerisinde yapılan sorgularda, kullanılan anahtar/değer çiftlerinin birden fazla olması durumunda, keyAsBson değişkeninin anahtar bilgileri de birden fazla olacağı için, bu anahtar bilgileri bir döngü yardımı ile okunur.

foreach (var keyAsBsonQueries in keyAsBson.Names) {}

Sorgularda kullanılan bilgileri ve kaç kez kullanıldıklarını belirlemek için; gelen anahtar/değer çiftini aramak, eğer bu anahtar bilgisi değişken içerisinde var ise, kullanım sayısını arttırmak; yok ise de yeni bir bilgi olarak değişkene eklemektir. Bunun için kullanılan değişken *queriesDocuments* isimli değişkendir.
Bu işlemleri yapabilmek için iki tane değişken daha kullanılır; *addDoc*, yeni anahtar/değer çifti eklenmesi durumunda kullanılacak olan değişken; *hasThisDoc* isimli queriesDocuments içinde aranan bilginin var olduğu/olmadığı sonucunu bir sorgu yardımı ile aktaracak olan değişkendir. Buna göre gelen yeni bilgi, queriesDocuments içerisinde aranarak eklememi düzeltmemi yapılacağını belirlemek için aşağıdaki sorgu gerçekleştirilir;
var hasThisDoc =
queriesDocuments.Find(f=>f.documentName.Equals(keyAsBsonQueries.ToString()));

(queriesDocuments'in QueriedDocumentsNameAndCountClass tipinde olduğu ve documentName ile documentCount şeklinde iki bilgi içerdiğini hatırlayınız!) Bu noktadan itibaren yapılacak tek şey, hasThisDoc bilgisine göre, documentCount değerini arttırarak güncelleme yapmak ya da yeni bilgiyi queriesDocuments'e eklemektir.
Tüm işlemler gerçekleştirildikten sonra, ihtiyaç olunan yerde çağrıldığında, daha kolay ve düzenli bir şekilde kullanılabilmesi için queriesDocuments küçükten büyüğe documentCount bilgisine göre sıralanmıştır.

4.2.10.3. analyzeMyIndexesAndSearchedDocuments(List<string> collectionIndexes, MongoCollection collection) Fonksiyonu

Eğer bir doküman, indekslenmemesine rağmen bir koleksiyon içerisinde fazlaca sorgulanıyor ise performansı arttırmak için indekslenmesi gerekir. Bu fonksiyon; bir koleksiyon içerisinde fazla sorgulanan anahtar/değer çiftinin, koleksiyonun indeksleri içerisinde olup olmadığını kontrol etmek için tasarlanmıştır.

Fonksiyon içerisinde inceleme yapacağı koleksiyonu *collection* ve bu koleksiyonun indekslerinin bir listesini *collectionIndexes* isminde iki parametre olarak alır.

Fonksiyondaki işlemlerin gerçekleştirilmesi için, koleksiyonun(collection) indeksleri dışında, dokümanları sorgularken kullanılan anahtar/değer çiftlerine ve sayılarına da ihtiyaç vardır. Bu bilgiyi elde edebilmek için, daha önceden tanımlanmış, *countMyDocumentsAndGroupThem(collection)* fonksiyonu kullanılır. Yardımcı fonksiyondan elde edilen bilgi, *queriesDocuments* değişkene atanır.

queriesDocuments ile collectionIndexes bilgilerini karşılaştırmak için, iç içe iki döngü kullanılır. Bu işlem ile hangi anahtar bilgisinin/bilgilerinin indekslenmek üzere önerileceğine karar verilir.

```
foreach (var queryDoc in queriesDocuments){}
foreach (var collIndex in collectionIndexes){}
```

Bu döngülerde dikkat etmesi gereken önemli nokta ise, queriesDocuments içerisinde gelen anahtar bilgisinin sorgulama için 1'den fazla kez kullanılmış olmasıdır.

```
if (queryDoc.documentCount > 1)
```

queriesDocuments içindeki her bilgiyi tek tek collectionIndexes içerisindeki bilgi ile karşılaştırılır. İndekslenmesi uygun görülen bir bilginin, addDoc içerisine işlenmesi için ise *result* isimli mantıksal değişken kullanılır. Bu değişkene başlangıç olarak *false* ilk değeri atanır, ve queryDoc ile collIndex bilgilerinin eşit olması durumunda, yani koleksiyonda

bulunan indeksler ile dokümanları sorgularken kullanılan anahtar bilgilerinin herhangi birinin aynı olmasında, değeri *true* olarak değiştirilir.

if (queryDoc.documentName.Equals(collIndex)) result = true;

Bunun yapılma sebebi ise, result bilgisinin false değerine sahip olması durumunda, yani indeksler ile gelen anahtar bilgileri arasında bir eşitlik bulunmaması durumunda, indekslenmek üzere önerilecek bilgiyi addDoc değişkenine ekleyebilmektir.

if (result == false) analyzedResults.Add(queryDoc.documentName);

Tüm işlemler bittiğinde, dokümanlar arasında çokça kullanılan anahtar/değer bilgileri, artık birer indeks adayıdır.

4.2.10.4. writeMyAnalyzedResultAndOfferIndex(List<string> analyzedResults, MongoCollection collection) Fonksiyonu

Bir öneride bulunduğunuzda, elimizde onu destekleyecek bilgiler olursa, insanların o öneriyi uygulama ihtimali daha da yükselir. Dolayısıyla kullanıcıya indekslemesi gereken dokümanlar gösterildiğinde, belirgin bir sebep sunulması gerekmektedir. Bu fonksiyon ile, indekslenmemiş bir dokümanın indekslenmesi ile, veritabanı içerisinde yapılacak basit bir okumanın performans hızını nasıl etkileyeceği bulunur.

Fonksiyon, içerisinde inceleme yapacağı koleksiyonu *collection*, dokümanlarının içerdiği ve indekslenmek üzere önerilecek olan anahtar/değer bilgilerini içeren *analyzedResults* ve isminde iki parametre olarak alır.

Fonksiyon içinde yapılan işlemler ise şu şekildedir; koleksiyonun(collection) içinde analyzedResults değişkeninin ilk elemanı bulunarak, bu işlemin ne kadar sürede gerçekleştiği bulunur ve sonucu *totalTimeTakenBefore* isimli değişken ile belirlenir.

collection.AsQueryable<BsonDocument>().Select(p => p[analyzedResults[0]]);

Ardından, eğer önerilecek olan anahtar bilgisi indekslenirse performansın nasıl etkileneceğinin görülmesi için, analyzedResults değişkeninin ilk elemanı koleksiyona(collection) indeks olarak atanır. İlk eleman örnek amaçlı seçilmiştir, isteğe bağlı olarak diğer elemanlar da indekslenebilir. İndeksleme için *IndexKeysBuilder* yapısı kullanılır ve tanımlanan yeni indeksin küçükten-büyüğe doğru sıralanması için *Ascending* olarak ayarlanır. *EnsureIndex()* komutu ile de koleksiyonun içine eklenir.

```
var keys = new IndexKeysBuilder();
keys.Ascending(analyzedResults[0]);
collection.EnsureIndex(keys);
```

Yeniden indeksleme işleminden sonra, performanstaki değişimi görmek için koleksiyonun(collection) içinde analyzedResults değişkeninin ilk elemanı bulunarak, bu işlemin ne kadar sürede gerçekleştiğini bulunur ve sonucu *totalTimeTakenAfter* isimli değişken ile belirlenir.

collection.AsQueryable<BsonDocument>().Select(p => p[analyzedResults[0]]);

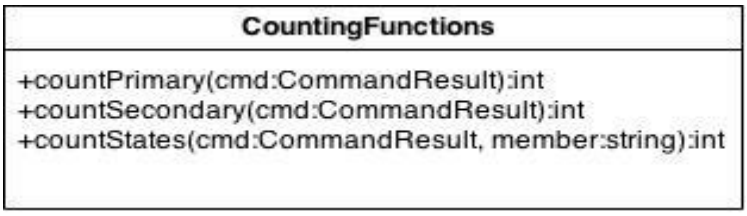
Bu işlemden sonra, kullanıcının veritabanı ayarlarının değişmiş olmasını engellemek için, sonradan atanmış indeks sistemden *DropIndex()* komutu ile silinir.

collection.DropIndex(keys);

Son olarak, tüm yapılan bu işlemler, kullanıcı ekranında gösterilmek üzere daha önceden tanımlanmış karakter tipindeki *writeResult* değişkeni ile düzenlenerek, geri bildirim yapılır.

4.2.11. CountingFunctions Sınıfı

Dağıtık Veritabanlarının en büyük özelliklerinden bir tanesinin replikasyon (kopyalama) olduğundan ve yapılan testler replikasyon yapılmış sistemleri de kapsar. Bu sınıf, replikasyon yapılmış veritabanlarının en önemli iki elemanının (Birincil-PRIMARY, İkincil-SECONDARY) varlığını ve sayısını kontrol etmek için yapılan işlemleri gerçekleştirmeye yardımcı olur.



Şekil 4.14 CountingFunctions

Şekil 4.14 ile gösterilen CountingFunctions, Birincil (Primary) ve İkincil (Secondary) elemanların sayısını belirlemek için "admin" koleksiyonunu kullanan sadece 3 tane fonksiyon içeren bir sınıftır.

4.2.11.1. countPrimary(CommandResult cmd) Fonksiyonu

countPrimary fonksiyonu 1 parametre alan ve sonuç olarak veritabanındaki Birincil eleman sayısını döndüren bir fonksiyondur. Bulundurduğu CommandResult tipindeki *cmd* parametresi, "admin" veritabanındaki replika kümelerinin durumunu (replicaset status) içeren bir komut taşır. CommandResult, MongoDB komutlarını C# üzerinde kullanabilmemizi sağlayan bir komut/aracı olarak düşünülebilir ve bu komutun/aracının kullanılabilmesi için "MongoDB.Driver" isim uzayının programa eklenmesi gerekmektedir. Gelen komut içerisindeki "PRIMARY" eleman sayısı bulunmak üzere *countStates()*

fonksiyonu çağırılır ve sonuç kullanıcıya gösterilmek üzere geri döndürülür (countStates() fonksiyonu 1.1.11.3 kısmında anlatılmıştır).

4.2.11.2. countSecondary(CommandResult cmd) Fonksiyonu

countSecondary fonksiyonu 1 parametre alan ve sonuç olarak veritabanındaki İkincil eleman sayısını döndüren bir fonksiyondur. Yani countPrimary() fonksiyonu ile aynı işi yapar. Bulundurduğu CommandResult tipindeki *cmd* parametresi, "admin" veritabanındaki replika kümelerinin durumunu (replicaset status) içeren bir komut taşır. , MongoDB komutlarını C# üzerinde kullanabilmemizi sağlayan bir komut/aracı olarak düşünülebilir ve bu komutun/aracının kullanılabilmesi için "MongoDB.Driver" isim uzayının programa eklenmesi gerekmektedir. Gelen komut içerisindeki "SECONDARY" eleman sayısı bulunmak üzere *countStates()* fonksiyonu çağırılır ve sonuç kullanıcıya gösterilmek üzere geri döndürülür (countStates() fonksiyonu 1.1.11.3 kısmında anlatılmıştır.).

4.2.11.3. countStates(CommandResult cmd, string member) Fonksiyonu

countStates fonksiyonu 2 parametreli bir fonksiyondur. Daha önce bahsettiğimiz iki fonksiyondan gelen istekler üzerine veritabanındaki Birincil ve İkincil elemanların sayısını belirler.

Aldığı parametrelerden bir tanesi "admin" veritabanı üzerinde gerçekleştirilmiş replika kümelerinin durumu hakkında bilgi veren bir komut, diğeri de bu komut kullanılarak aranılacak "PRIMARY" veya "SECONDARY" şeklinde gelecek olan replika kümesi elemanıdır.

Fonksiyon 3 tane lokal değişken bulundurmaktadır. Bunlar *number, members* ve *count* değişkenleridir;

- *number;* tamsayı (int) olarak tanımlanmış değişkendir. Aranan elemanın (member) sayısı bu değişiken ile bellekte saklanır.
- *members;* BSonArray şeklinde tanımlanmış bir değişkendir. Burada BSonArray, MongoDB veritabanına ait bilgileri aktarır. BsonArray komutunu kullanabilmek için, programa "MongoDB.BSon" isim uzayının eklenmesi gerekmektedir. members lokal değişkeni, fonksiyon içine gelen komut değişkeninin-cmd içerisinden çekilecek olan gerekli bilgileri bellekte saklamak için tanımlanmıştır.
- *count;* tamsayı (int) olarak tanımlanmış değişkendir. *members* değişkeni içerisinde bulunan eleman sayısını tutmak için kullanılacaktır.

İşlemleri gerçekleyebilmek için, fonksiyonun bir parametresi olan ve veritabanındaki replika kümesinin tüm bilgilerini içeren komut içerisinden üyelerini (members) belirlemek

gerekmektedir. Bunun için daha önceden tanımlanmış olan *members* değişkenine aşağıda sırasıyla açıklanan kodlar kullanarak replika üyeleri atanmıştır;
Burada *cmd* fonksiyona gelen komutları tutan bir parametredir.

cmd.Response;

sayesinde komutu çalıştırarak içerisindeki bilgilere erişilir. Daha sonra *cmd* bilgileri içerisinden members bilgisi çekilir, bunun içinde aşağıdaki kod kullanılır;

cmd.Response.GetElement(“members”);

ihtiyaç olan elemana/elemanlara ulaştıktan sonra, kolay kullanılmaları için üyeler BSonArray halinde tutuldu. Bir ifadeyi BSonArray şeklinde çevirebilmek için “.AsBsonArray” komutunun kullanılması gerekir;

members = cmd.Response.GetElement("members").Value.AsBsonArray;

ve sonuç olarak test edilecek veritabanı içerisindeki (varsa) replikasyon kümelerinin üyelerinin bilgilerine ulaşmış ve BSon dizisi içerisine atanmış olur. Dizi içerisine atama nedeni, PRIMARY veya SECONDARY elemanlara kolaylıkla, ulaşarak sayılarını belirleyebilmektir.

Replika kümesinin üyelerine ulaştıktan sonra üye sayısı bulundu, bunun sebebi üyelerin bir dizi ile bellekte tutuyor oluşu ve üye sayısını bulduktan sonra her üyenin istenilen şarta uygun olup olmadığını kontrol etmektir. Üyelerin sayısını bellekte saklamak için *count* değişkeni kullanılmış ve aşağıdaki işlemi gerçeklenmiştir;

count = members.Count

“.Count” komutu, bir değişkenin içerisindeki (eğer uygun şekilde tanımlanmış ise) eleman sayısının bulmasını sağlar.

Üyelerin sayısı bulunduktan sonra her üyenin içeriğini tek tek kontrol ederek *member* değişkeninin istenen şartı sağlayıp sağlamadığı kontrol edilmelidir. Bunun için döngü yardımı ile *members* dizisinin her elemanı üzerinde bazı işlemler gerçekleştirilmiştir. Dikkat edilmesi gereken önemli nokta, *members* dizisinin içerisindeki her elemanının birden fazla özelliğe sahip olmasıdır. Yani *members* içerisinde de birden fazla anahtar/değer ilişkisinin olması. (Bu konu ile ilgili detaylı bilgi ClassReplicaTest isimli, replika kümesi için altı tane test barındıran sınıfın altıncı testinde veirlmiştir.). Burada önemli olan değer “stateStr” anahtar ismindedir. Dolayısıyla her elemanın içerisindeki “stateStr” değerinin, aranan ölçüt yani *member* değişkeni ile uyumlu olması gerekir. Diğer işlemler BsonDocument yardımı ile gerçekleşir. *members* dizisinin elemanlarını her seferinde BsonDocument tipine dönüştürmek için “.AsBsonDocument” komutu kullanılır ve *myState* isimli eğişken ile bellekte tutulur;

BsonDocument myState = members[i].AsBsonDocument;

Ardından “stateStr” anahtarı ile ulaşılan değer belirlenir,

string state = myState["stateStr"].ToString();

Elde edilen değerin “string” tipine çevrilmesindeki amaç, bu değişkenin *member* ile kıyaslanacak oluşudur. Bu doğrultuda *state* değişkeninin değerinin, alınan parametreye

uygun olması durumunda eleman sayısında arttırma yapılmış ve eleman sayısını tutmak için kullanacak olan *number* değişkeni ile aşağıdaki şekilde gerçeklenmiştir;

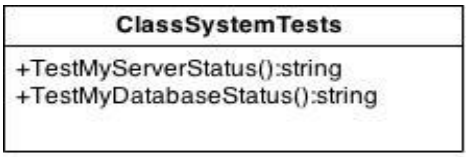
if (state.Equals(member.ToUpper())) number++;

Yukarıda belirtilmiş tüm işlemler döngü döndüğü sürece gerçeklenmiş ve işlem sonucunda kullanıcıya istediği Birincil veya İkincil elemanların sayısını döndürülmüştür.

Sonuç olarak bu sınıf içerisinde; replika kümesinin elemanlarının dağılımını kontrol etmek için, "admin" koleksiyonu üzerinde bir sorgu gerçeklenmiş ve bu sorgu *countPrimary* ve *countSecondary* fonksiyonlarına işlem yapılmak üzere gönderilmiştir. Daha sonra her iki fonksiyonda Birincil ve İkincil eleman sayılarını belirlemek üzere *countStates* fonksiyonunu çağırılmıştır. İlk iki fonksiyon tek kod satırı barındıran fonksiyonlar iken asıl işlemler *countStates* fonksiyonu içerisinde gerçeklenmiştir. Buna göre; *countStates* içerisine gelen sorgu içinden, replika kümesi üyelerine ait bilgileri ve bu bilgiler içerisinden de tanımlanma durumlarını içeren anahtar/değer ikilisi bulunarak, bu değerin, yine fonksiyona parametre olarak verilen ve aranan elemanın PRIMARY veya SECONDARY olması bilgisini içeren değişken ile uyumluluğu kontrol edilmiştir. Ve sonuç olarak Birincil veya İkincil elemanlarının sayısı bulunmuştur.

4.2.12. ClassSystemTests Sınıfı

Bir veritabanı var ise, bu veritabanına ve veritabanı sunucusuna bağlantının sağlanıp sağlanmadığının da kontrol edilmesi gerekir. İşte bu işlemleri gerçekleştirebilmek için Şekil 4.15 ile gösterilen ClassSystemTests sınıfı oluşturulmuştur.



Şekil 4.15 ClassSystemTests

Sınıf iki tane, geri dönüş tipi karakter(string) olan fonksiyon içermektedir.

4.2.12.1. TestMyServerStatus() Fonksiyonu

Fonksiyon adından da anlaşılacağı gibi, sunucunun bağlı olup olmadığını kontrol etmek üzere tasarlanmıştır.

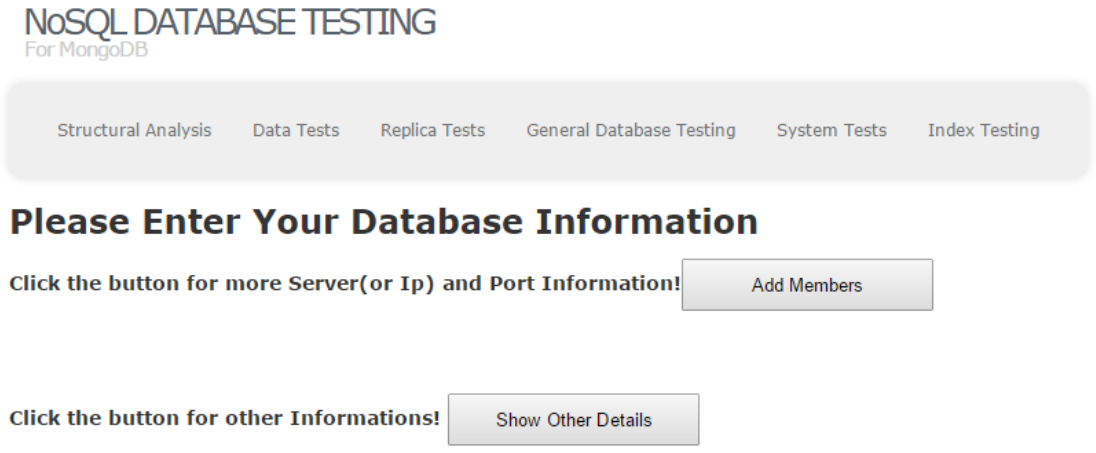
4.2.12.2. TestMyDatabaseStatus() Fonksiyonu

Fonksiyon adından da anlaşılacağı gibi, veritabanının bağlı olup olmadığını kontrol etmek üzere tasarlanmıştır.

5. TESTLER

Proje, Dağıtık Veritabanları üzerinde Yazılım testi yapmak üzere geliştirilmiştir. Bu doğrultuda, üzerinde test yapmak için, dağıtık bir veritabanı olan MongoDB seçilmiştir. Ayrıca testleri geliştirmek için C# programlama dili kullanılmıştır. Sonuç olarak, C# programlama dili kullanılarak, web tabanlı MongoDB Veritabanı Yazılım Testini gerçekleyecek bir sistem tasarlanmıştır.

MongoDB'ye C# (veya başka bir dil) yardımı ile bağlanabilmek için bir bağlantı dizesine ihtiyaç vardır. Bu dizenin özelliklerinden ve nasıl oluşturulduğundan MongoDB ve Formlar başlığında bahsedilmiştir. Geliştirilen testlerin uygulanabilmesi için, veritabanının kayıtlı olduğu sisteme bağlanılması, dolayısıyla, kullanıcıdan veritabanı ile ilgili bazı bilgilerin alınması gerekmektedir. Bu bilgiler; veritabanının kayıtlı olduğu sunucunun IP ve PORT bilgileri, sistemde gerekli ise kullanıcı ismi ve şifresi, veritabanını geliştirirken replikasyon yapılıp yapılmadığı bilgisi ve veritabanı ismi şeklindedir. Bilgilerin kullanıcı tarafından girilmesi için, sistem ilk açılışta "HomeScreen.aspx" sayfasına yönlendirildi. Bu sayfada "AddMembers", "buttonVisibleOthers" ve "buttonConnectionString" isimli üç Button, "tblName" isimli bir Table , "hdnCount" isimli bir HiddenField, "Panel1" isminde bir panel, bilgi girişini kolaylaştırmak için beş adet Label, kullanıcı ismi-şifresini alabilmek için "txtUserName" ve "txtPassword", veritabanı isminin bilgisini alabilmek için "txtDatabaseName" isminde üç adet TextBox ve replikasyon yapılıp yapılmadığının belirlenebilmesi için "checkboxReplicaYes" ve "checkboxReplicaNo" isminde iki adet CheckBox nesneleri statik olarak bulunmaktadır. Bu nesnelerden AddMembers ve buttonVisibleOthers nesneleri hariç, diğer tüm nesnelerin görülebilirliği yani Visibile özelliği *false* olarak belirlenmiştir. Aşağıda sayfanın ilk açılıştaki hali gösterilmiştir;



Şekil 5.1 NoSQL Database Testing

“Add Members” isimli Button, sistemin IP ve PORT bilgilerinin girilmesi için ekrana dinamik bir şekilde TextBox ve Label yerleştirmektedir. Nesneye tıklandığında gerçekleşen işlemler ise şu şekildedir; *count* isimli değişkene hdnCount nesnesinin bulundurduğu değer atanır. Bunun sebebi, IP ve PORT için eklenecek olan dinamik TextBox sayısını tutabilmektir;

```
int count = int.Parse(hdnCount.Value);
count++;
```

Daha sonra *count* değişkeninin son değeri, hdnCount değişkenine aktarılır ki ekrana eklenen nesne sayısı kaybolmasın.

```
hdnCount.Value = count.ToString();
```

Add Members nesnesine her tıklandığında, yeni gelecek IP ve –varsa- PORT bilgileri için birer TextBox oluşturulmaktadır. Bu işlem için;

```
TextBox textBoxHost = new TextBox();
TextBox textBoxPort = new TextBox();
```

olmak üzere iki tane TextBox nesnesi tanımlanır. Ve bunların ayırıcılar olan ID bilgileri aşağıdaki şekilde belirlenir;

```
textBoxHost.ID = "TextBoxHostID" + count.ToString();
textBoxPort.ID = "TextBoxPortID" + count.ToString();
```

bu şekilde tanımlanma nedenleri, ID bilgilerinde bir çakışma yaşanmasını önlemektir. Ayrıca her bir IP ve PORT için Label tanımlanarak, aşağıdaki şekilde ID ataması gerçekleştirilir;

```
labelHost.ID = "labelHostID" + count.ToString();
labelPort.ID = "labelPortID" + count.ToString();
```

bu işlemler tamamlandıktan sonra, tblMain isimli Table içine, TextBox ve Label nesneleri eklenir. AddMembers nesnesi kullanılarak yapılan işlemde, her seferinde iki nesne oluşturularak Table içine eklenir ancak daha önceden oluşturulmuş Label ve TextBox nesnelerinin gösterimine dair bir işlem gerçekleştirilmez. Bu işlemi gerçekleştirebilmek için, sayfanın ilk yüklenme kısmına yani Page_Load içerisine *CreateTextBoxes()* isminde, geri dönüş tipi olmayan bir fonksiyon eklenir. CreateTextBoxes() içerisinde gerçekleştirilen işlemler ise şu şekildedir; *count* isimli değişkene ekrana dinamik olarak daha önceden eklenmiş olan nesnelerin sayısı atanır;

```
int count = int.Parse(hdnCount.Value);
```

Daha sonra, önceden oluşturulmuş/tanımlanmış TextBox ve Label nesnelerini ekrana ekleyebilmek için, bir döngü içerisinde aşağıdaki işlemler tekrarlanır,

```
TextBox textBoxHost = new TextBox();
textBoxHost.ID = "TextBoxHostID" + i.ToString();
textBoxHost.Text = Request.Form["TextBoxHostID" + i.ToString()];
```

İşlemleri hem textBoxHost hem de textBoxPort için gerçekleştirilir. Ve ardından bu nesneler AddMembers_Click() içinde olduğu gibi tblMain içine yerleştirilir. Bunun sebebi, içerisine bilgi girişi yapılmış olan TextBox nesnelerindeki bilgileri kaybetmemek ve bu bilgilere ulaşabilmektir. Kullanıcı sunucu bilgilerinin girişini yaptıktan sonra “Show Other Details”

yani buttonVisibleOthers nesnesine tıkladığında, ihtiyaç olan diğer bilgilerin girilebilmesi için, daha önceden Visible özelliği *false* yapılmış tüm nesnelerin bu özelliği değiştirilerek *true* yapılır. Kullanıcı gerekli bilgileri girdikten sonra "Get All Information" nesnesine tıkladığı zaman aşağıdaki fonksiyon aktifleşir,

buttonConnectionString_Click()

ve aşağıdaki işlemler ile bağlantı dizesi oluşturulur. İşlemlere geçmeden önce, MongoDB için genel bağlantı dizesini hatırlanacak olursa,

mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]] [/[database][?options]]

Şeklindedir. Dolayısıyla ekrandan alınan bilgilerin doğru bir sıra ile yerleştirilmesi ile bağlantı dizesi elde edilebilir. Bunun için Formlar sınıfından yararlanılacaktır.

Öncelikle bağlantı dizesinin ilk bölümünü –varsa- kullanıcı adı ve şifre bilgilerinin yerleştirilmesi gerekir. Bunun için daha önceden tanımlanmış olan txtUserName ve txtPassword nesnelerinin içerdiği bilgiler alınarak, bağlantı dizesi olarak kullanılacak olan Formlar.connectionString değişkenine uygun şekilde atanır;

```
Formlar.connectionString = Formlar.connectionString + txtUserName.Text + ":" + txtPassword.Text + "@";
```

İhtiyaç olan IP ve PORT bilgileridir. Bu bilgileri öğrenebilmek için öncelikle, kullanıcının kaç tane IP-PORT nesnelerini Table içine eklettiği belirlenmelidir. Bunu belirleyebilmek için, hdnCount değişkeninin içindeki bilgiler kullanılır. Nesnelerin sayısı belirlendikten sonra, her biri eşsiz bir ID'ye sahip olduğu için, içerdikleri bilgilere erişmek çok kolaydır. Birden fazla IP-PORT TextBox nesnesi bulunabileceği için bir döngü kullanılır, bu döngünün içerisinde,

```
TextBox tb = (TextBox)tblMain.Rows[rowNumber].Cells[0].FindControl("TextBoxHostID" + rowNumber.ToString());
TextBox tp = (TextBox)tblMain.Rows[rowNumber].Cells[0].FindControl("TextBoxPortID" + rowNumber.ToString());
```

şeklinde bilgiler çekilir. Burada dikkat edilmesi gereken üç durum vardır; ilki, kullanıcının yanlışlıkla AddMembers nesnesine tıklaması ve fazladan nesne eklemesi. Bu durumu anlayabilmek için, TextBox nesnemizin içinin *null* değere sahip olup olmadığı kontrol edilir. Bu sayede, boş bir nesneden bilgi çekmeye çalışarak programın hata vermesi önlenir,

```
if (tb != null)
```

İstenilen şartın sağlanması durumunda,

```
string tableHost = tb.Text;
string tablePort = tp.Text;
```

şeklinde IP ve PORT için olan TextBox nesnelerinin içerdikleri bilgilere ulaşılır. İkincisi, eklenecek olan, iki nokta üst üste(:), virgül (,) ve slash (/) sembollerini düzgün bir sırada eklenmesidir. Genel bağlantı dizesinde görüleceği üzre, slash(/) sembolü sadece tüm IP-

PORT bilgileri girildikten sonra konulmalıdır. Üçüncüsü ise kullanıcının sistemi tasarlarken PORT bilgisini belirlememiş olası durumudur, işte tüm bunları göz önünde bulundurarak bir birleştirme işlemi gerçekleştirilir,

```
if (rowNumber != rowCount)
                {
                   if (tablePort != "")
                      Formlar.connectionString = Formlar.connectionString + tableHost +
":" + tablePort + ",";
                   else
                      Formlar.connectionString = Formlar.connectionString + tableHost +
",";
                }
                else
                {
                   if (tablePort != "")
                      Formlar.connectionString = Formlar.connectionString + tableHost +
":" + tablePort + "/";
                   else
                      Formlar.connectionString = Formlar.connectionString + tableHost +
"/";
                }
```

Bir diğer ihtiyaç duyulan bilgi ise veritabanı ismidir. Bu bilgiyi ekrandan almak için de txtDatabaseName nesnesi kullanılır,

```
if (txtDatabaseName.Text!="")
        Formlar.connectionString = Formlar.connectionString + txtDatabaseName.Text +
"/";
```

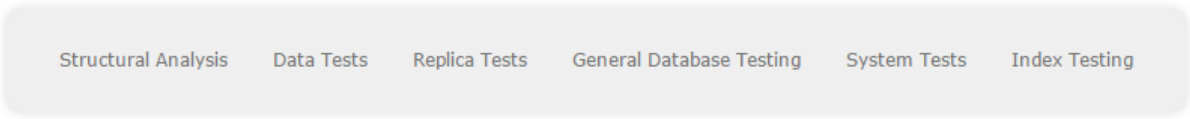
Son olarak, sistemi oluştururken replikasyon yapılıp yapılmadığı bilgisi kontrol edilmelidir. Bu durumda bağlantı dizesi özel bir bilgi içermek zorundadır ve bunu gerçeklemek için de checkboxReplicaYes nesnesinin tıklanıp tıklanmadığı kontrol edilerek aşağıdaki işlem gerçekleştirilir,

```
 if (checkboxReplicaYes.Checked)
   Formlar.connectionString = Formlar.connectionString +
"?safe=true&connect=replicaset";
```

Tüm işlemler bittikten sonra artık testlere başlanabileceği için, kullanıcı "TestResults.aspx" isimli sayfaya yönlendirilir.

```
Response.Redirect("~/TestResults.aspx",true);
```

Her bir test sayfaya menü şeklinde eklenmiştir ve menüde istenen teste tıklandığında işlem gerçekleştirilir. Testler ile ilgili bilgiler için başlıkları inceleyebilirsiniz;



Şekil 5.2 Menü

5.1. Yapısal Analiz

Yapısal Analiz Testi ile, veritabanının sahip olduğu koleksiyon isimleri ve bu koleksiyonların içerdikleri dokümanların anahtar/değer ikilisindeki anahtar isimleri ve değer tipleri belirlenerek kullanıcıya gösterilir. Veritabanının yapısının belirlendiği bu test için, ClassGetStructureOfDatabase sınıfı kullanılır. Burada sınıf içerisinde yapılanlar detaylıca anlatılmıştır.

Menüden bu testin yapılması için gerekli kısım seçildiği zaman,

getStructuralAnalysis_click()

aktifleşir ve "getStructure.aspx" sayfasına yönlendirir. Bu sayfanın içerisinde *structureLabel* ismi verilen bir Label ve *structurePlaceHolder* ismi verilen bir PlaceHolder nesnesi bulunur. getStructure.aspx içerisinde belirli işlemler yapılarak, test gerçekleştirilir ve ekrana sonuçlar listelenir. getStructure.aspx sayfasının kod kısmında, ClassGetStructureOfDatabase tipinde bir değişken tanımlanarak, sınıfın *getStructureOfMyDatabase()* fonksiyonu kullanılır ve *structurePlaceHolder* nesnesini bu fonksiyonun içinde yollanır. Bu sayede test için kullanılacak sınıf işlem yapmaya başlar;

```
ClassGetStructureOfDatabase getStructureClass = new ClassGetStructureOfDatabase();
getStructureClass.getStructureOfMyDatabase(structurePlaceHolder);
```

ClassGetStructureOfDatabase sınıfında yapılanlar şu şekildedir; test için veritabanının kullanıcı tanımlı koleksiyonlarına ihtiyaç duyulduğundan, ilk önce koleksiyonlara ulaşılır. Bunun için, *FindDatabasesCollections* sınıfı ve bu sınıfın *getCollections()* isimli fonksiyonu kullanılır.

```
FindDatabasesCollections finder = new FindDatabasesCollections();
List<MongoCollection> collections = finder.getCollections();
```

Yukarıdaki kodda görüldüğü gibi, veritabanına ait olan tüm koleksiyonlar collections isimli değişkenin içine alınır. İkinci adım olarak, koleksiyonlara ait dokümanların anahtar/değer ikililerinden anahtar ismi ve değer tipi bilgilerine ulaşılır ve kullanıcı ekranına düzenlenmiş bir tablo ile listelenir. Listeleme işlemi için Repeater ve DataTable nesneleri kullanılır.

Tanımlamadan anlaşılacağı üzere, collections değişkeni List tipinde tanımlanmıştır. Bu listenin içindeki her elemana tek tek ulaşabilmek için foreach döngüsü kullanılır. Koleksiyonun içinde bulunan dokümanın anahtar bilgisi ve değer tipi bilgilerini ekranda listelemek için DataTable nesnesi aşağıdaki şekilde kolon isimleri alır;

```
dt.Columns.Add("Koleksiyon:");
dt.Columns.Add("Kolon ismi:");
dt.Columns.Add("Kolon tipi:");
```

DataTable nesnesinin kolonlarını koleksiyonun anahtar ismi ve değer tipi bilgileri ile doldurabilmek için koleksiyon içerisinde bir sorgu yapılır ve koleksiyonun ilk elemanına ulaşılır. Ve bu sorgu BsonDocument şeklinde gerçeklenir.

```
var query = item.AsQueryable<BsonDocument>();
BsonDocument documents = query.First<BsonDocument>();
```

documents isimli ve koleksiyonun ilk dokümanının bilgileri tutan değişken, BsonDocument şeklinde olduğundan, anahtar isimlerine ulaşmak oldukça kolaydır, bunun için bir BsonDocument'in anahtar bilgilerine ulaşmayı sağlayan *.Names* komutunu kullanıldı,

```
IEnumerable documentsColumnName = documents.Names;
```

Diğer adımda bu isimler yardımı ile, *dt* nesnesine anahtar ismi ve değer tipi eklemendi. Bunun için *documentsColumnName* değişkenindeki bilgiler kullanılarak tek tek isimlere ve tiplere ulaşıldı. Değer tiplerine ulaşmak için, *documents* değerinin içeriğine indis kullanılarak erişilebildi.

```
foreach (String itm in documentsColumnName)
{
   dt.Rows.Add(item.ToString(), itm.ToString(), documents[i].GetType().ToString());
    i++;
}
```

Son adım olarak da *dt* tablosu Repeater içerisine ekledi.

Yapısal Analiz testinin sonucu olarak, kullanıcı Şekil 5.3'deki gibi, veritabanı koleksiyonları ve dokümanları hakkında detaylı bilgiye sahip olmuş olur.

Dağıtık Veritabanın içerisinde bulunan; koleksiyonları, koleksiyonların kolon bilgileri ve tipleri aşağıda listelenmiştir.

Koleksiyon	Kolon ismi	Kolon tipi
NorthWind_db.Categories_Collection	_id	MongoDB.Bson.BsonObjectId
NorthWind_db.Categories_Collection	CategoryName	MongoDB.Bson.BsonString
NorthWind_db.Categories_Collection	Description	MongoDB.Bson.BsonString

Şekil 5.3 Yapı Testi Genel Görünüm

5.2. Veri Testleri

Bir veritabanının gerçeklemesi gereken en önemli işlevler; veriye ulaşabilme, istenilen sayıda veri ekleme, silme ve güncellemedir. Dolayısıyla veritabanının operasyonlarını test etmek bir diğer önemli noktadır. Veritabanındaki bir koleksiyondan verileri okuyabilme hızı, bir seferde birden fazla veriyi ekleyebilme, silme veya güncelleme işlemlerini tutarlı ve sorunsuz bir şekilde gerçekleştirmesi, veritabanının ne derece güvenilir olduğunu gösterir. Veri Testleri bir veritabanı için bu özellikleri test eder. Kullanıcı menüde Veri Testleri başlığına tıklandığında, *dataTestsInformation_click()* yardımı ile sistem "DataTestInformationMenuPage.aspx" sayfasına yönlendirir. Bu sayfa içerisinde, başlığın altındaki testler ile ilgili genel bir bilgi verilmektedir.

Veri Testleri, test yapmak isteyen kullanıcıya 4 tane seçenek sunar. Bunlar; seçilen koleksiyondan dokümanların okunma hızını test eden "Data Selection Test", seçilen koleksiyona istenilen sayıda veriyi ekleyen "Data Insert Test", seçilen koleksiyondan istenilen sayıda veriyi silen "Data Removal Test" ve seçilen koleksiyondan istenilen sayıda veriyi silen "Data Update Test". Bahsedilen 4 test "DataTests.aspx" sayfasına yönlendirilerek gerçekleştirilir. Ayrıca bu testler ClassDataTests sınıfı kullanılarak yapılır. DataTest.aspx dataLabel isminde bir Label, dataListBox isminde koleksiyonların listeleneceği bir ListBox, dataTestGridView isminde test sonuçlarında elde edilen verilerin listeleneceği bir GridView, textBoxData isminde kaç tane veri üzerinde işlem yapılması gerektiği bilgisini içeren bir TextBox ve buttonProcess isminde testi başlatacak olan bir Button nesnesini içermektedir. Yapılan testlerin sonuçları bu dataLabel ve dataTestGridView kullanılarak ekranda gösterilmektedir. DataTest.aspx'in kod kısmında; sayfa açılışında, hangi testin yapılacağına dair karar verilerek ona uygun işlemler gerçekleştirilmektedir. Bunun için, her test için ayrıca tanımlanmış ve bir değer atanmış olan *Session["sonuc"]* nesnesi kullanılır. *Session["sonuc"]*, tüm testler DataTest.aspx sayfasında gerçeklendiği için, kullanıcının hangi testi seçtiğine karar vermek için kullanılır. *Session["sonuc"]* ile atanan değerler testleri açıklarken belirtilmiştir.

Sayfada bulunan textBoxData nesnesi her zaman görünür değildir. Bu nesne, kullanıcıdan veri sayı bilgisi isteneceği zaman aktifleştirilir. Ayrıca Page_Load kısmında, testlere uygun bilgileri ekrana çıkarmak dışında, dataListBox nesnesine, kullanıcının seçim yapmasını sağlamak için, veritabanında bulunan koleksiyonlar listelenmektedir. Kullanıcı üzerinde işlem yapılmasını istediği veritabanını seçip gerekli veri sayısını girdikten sonra "Do The Transaction" isimli butona tıkladığında, *buttonProcess_Click()* yordamı işlemeye başlar. buttonProcess_Click() içerisinde gerçekleşenler ise şu şekildedir; fonksiyonun içerisinde, Page_Load'dakine benzer şekilde, hangi testin yapılacağına dolayısıyla ClassDataTests sınıfının hangi fonksiyonu kullanılacağına karar verilir. Bu karar için yine switch-case yapısı kullanılır. Aynı yapının içinde, test sonuçları ile elde edilen veriler dataTestGridView içerisine eklenerek, kullanıcı ekranında gösterilir.

Yazının devamında yapılan testlerin içeriklerinden bahsedilmiştir. Aşağıda açıklanan tüm işlemler, ClassDataTests sınıfı içerisinde gerçekleşmektedir.

5.2.1. Veri Okuma Testi

Veri Testleri menü başlığı altındaki testlerden ilkidir. Veritabanının seçilen koleksiyonu içerisinde bulunan verilerin okunarak ekrana listelenmesini sağlar. Testin amacı, verilerin koleksiyonun içinden okunma hızını hesaplamaktır.

Test içerisinde yapılan işlemler şu şekildedir; Kullanıcı menü başlığı altında ilgili bölümü seçtiğinde, *dataSelectionTest_click()* fonksiyonu ile, "DataTest.aspx" sayfasına yönlendirilir, veri testleri için gerçekleşen tüm işlemler ve sonuçları bu sayfa ile bağlantılıdır. dataSelectionTest_click() fonksiyonu içerisinde sayfalar arası veri aktarımı için kullanılan *Session["sonuc"] = "selection"* güncellemesi yapılır. Bu işlemin yapılmasının sebebi, sayfalar arası geçişte, hangi testin gerçekleşeceğine kolaylıkla karar vermektir. DataTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassDataTests sınıfındaki *DoDataSelectionTest()* fonksiyonu kullanılır.

DoDataSelectionTest() fonksiyonu, *collection* ve *myGridViewTable* isimlerinde iki parametre almaktadır. Bu parametrelerden ilk, yani collection, üzerinde işlem yapılacak olan koleksiyon bilgisini içeren karakter tipinde bir değişkendir. İkinci parametre, myGridViewTable, verileri listelemek için kullanılacak olan DataTable nesnesidir.

Herşeyden önce yapılması gereken, koleksiyondan verileri okuma süresini belirleyebilmek için bir başlangıç zamanı tanımlamaktır. Bu yüzden başlangıç zamanını tutan, *startTime* isminde bir değişken tanımlanır;

```
DateTime startTime = DateTime.Now;
```

Seçilen koleksiyon(collection) MongoDB'nin bir koleksiyonu olduğu ve C# içerisinde kullanılabilmesi için aktifleştirilmesi gerekir; FindDatabasesCollections sınıfının getCollection() fonksiyonu kullanılarak, gelen koleksiyon ismi bilgisi kullanılarak, veritabanına ait Mongo koleksiyonu aktifleştirilir ve *myCollection* isimli değişkende gelen koleksiyon bilgisi tutulur;

```
FindDatabasesCollections finder=new FindDatabasesCollections(Formlar.connectionString);
MongoCollection myCollection = finder.getCollection(collection);
```

Mongo koleksiyona ulaşıldıktan sonra, koleksiyon içerisindeki tüm dokümanları bulabilmek için IQueryable<BsonDocument> şeklinde bir sorgu tanımlanır. (Koleksiyonun içerisindeki

veriler hakkında bilgimiz olmadığı için, MongoDB içerisindeki bilgileri BsonDocument ile okumak çok daha kolaydır).

```
var query = myCollection.AsQueryable<BsonDocument>();
```

Koleksiyonun içindeki dokümanlara ulaştıktan sonra, bu dokümanlar myGridViewTable içinde listelenir ve ulaşılan süre kullanıcı ile paylaşılır. Sorgunun içindeki dokümanlara ulaşabilmek için okuyabilmek için foreach döngüsü kullanılır ve her doküman myGridViewTable içine eklenir. Bu sayede koleksiyonun içindeki dokümanlar kullanıcı ekranına listelenir.

```
foreach (var item in query){
        myGridViewTableRow = myNewGridViewTable.NewRow();
myGridViewTableRow["Documents"] = item.ToString();
        myNewGridViewTable.Rows.Add(myGridViewTableRow);
myNewGridViewTable.AcceptChanges();
}
```

Son olarak, verilerin koleksiyondan okunarak ekranda listelenmesi için geçen süre hesaplanır. Bunun için bitiş zamanı belirlenir ve *endTime* isimli değişkene atanır.

```
DateTime endTime = DateTime.Now;
```

Tüm işlemler tamamlandıktan sonra, toplam geçen süre bilgisine, başlangıç süresi ile bitiş süresinin farkı alınarak ulaşılır.

5.2.2. Veri Ekleme Testi

Veri Testleri menü başlığı altındaki testlerden bir diğeridir. Veritabanının seçilen koleksiyon içerisine istenen sayıda veriyi ekler ve bunların koleksiyonun içine doğru sayıda ve doğru biçimde eklenip eklenmediğini kontrol eder. İşlem sonunda, yeni üretilen dokümanlar/veriler, renklendirilmiş bir şekilde tablonun üst sıralarında, eski dokümanlar da hemen bu dokümanların altında listelenecektir.

Kullanıcı menü başlığı altında ilgili bölümü seçtiğinde, *dataInsertTest_click()* fonksiyonu ile, "DataTest.aspx" sayfasına yönlendirilir, veri testleri için gerçekleşen tüm işlemler ve sonuçları bu sayfa ile bağlantılıdır. dataSelectionTest_click() fonksiyonu içerisinde sayfalar arası veri aktarımı için kullanılan *Session["sonuc"] = "insert"* güncellemesi yapılır. Bu işlemin yapılmasının sebebi, sayfalar arası geçişte, hangi testin gerçekleşeceğine kolaylıkla karar vermektir. DataTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassDataTests sınıfındaki *DoDataInsertTest()* fonksiyonu kullanılır.
DoDataInsertTest() fonksiyonu, *collection*, *myGridViewTable* ve *count* isimlerinde üç parametre almaktadır. Bu parametrelerden ilki, yani collection, üzerinde işlem yapılacak

olan koleksiyon bilgisini içeren karakter tipinde bir değişkendir. İkinci parametre, myGridViewTable, verileri listelemek için kullanılacak olan DataTable nesnesi ve sonuncu parametrede, koleksiyona kaç tane veri eklenmesi gerektiğini belirten *count* isminde tamsayı değişkenidir.
Bu test içerisinde eklenecek olan yeni veriler, CollectionRandomInsert sınıfı kullanılarak, sistem tarafından üretilen verilerdir. Yani kullanıcının yeni eklenecek veriler için herhangi bir giriş yapmasına gerek yoktur.

Seçilen koleksiyon(collection) MongoDB'nin bir koleksiyonu olduğu ve C# içerisinde kullanılabilmesi için aktifleştirilmesi gerekir; FindDatabasesCollections sınıfının getCollection() fonksiyonu kullanılarak, gelen koleksiyon ismi bilgisi kullanılarak, veritabanına ait Mongo koleksiyonu aktifleştirilir ve *myCollection* isimli değişkende gelen koleksiyon bilgisi tutulur;

```
FindDatabasesCollections finder=new FindDatabasesCollections();
MongoCollection myCollection = finder.getCollection(collection);
```

İlk adım olarak, koleksiyona kayıt edilen yeni dokümanlar ile birlikte eski dokümanlar da ekranda listeleneceğinden, koleksiyona herhangi bir kayıt yapılmadan önce tüm dokümanlarına ulaşılır. Bunun sebebi, yeni kayıtlar yapıldıktan sonra, eski kayıtlar ile yeni kayıtların ayrılmasının oldukça güç olmasıdır. Koleksiyon içerisindeki tüm dokümanları çekmek için IQueryable<BsonDocument> şeklinde bir sorgu gerçeklenmiştir;

```
var query = myCollection.AsQueryable<BsonDocument>();
```

Testin amacı seçilen koleksiyona istenilen sayıda veriyi eklemek olduğundan, kullanıcının belirlediği sayıda veri üretilerek koleksiyona eklenir. Bu işlem için bir döngü kullanılır ve bu döngü istenilen sayıda veri eklenene kadar gerçeklenir.

```
for (int i = 0; i < count; i++){}
```

Döngünün içindeki işlemler ise şu şekildedir; CollectionRandomInsert sınıfı kullanılarak üretilen veri *lastData* isminde ve BsonDocument tipindeki değişkene atanır, ardından, üretilen tüm verilere rahatlıkla ulaşabilmek için kullanılacak olan *lastDatas* isimli liste tipindeki değişkene, lastData eklenir. Bunun nedeni, ilerleyen satırlarda, koleksiyona eklenen doküman sayısının doğru olup olmadığını kontrol edebilmektir.

```
lastData = InsertRandomDataInCollection(collection);
lastDatas.Add(lastData);
```

Eklenecek veriyi belirleme işlemi gerçekleştikten sonra, veri koleksiyona *Save()* komutu kullanılarak eklenir. Ardından da yeni üretilen veriler myGridViewTable nesnesi ile kullanıcıya gösterilir. İstenilen sayıda veriyi koleksiyona ekleme işlemini gerçekleştirip tabloya ekledikten sonra, koleksiyona ait eski/asıl veriler de tabloya eklenir. Bu işlem, daha önceden koleksiyondan okunarak *query* isimli sorguda tutulan dokümanları, bir döngü

yardımı ile okuyup tabloya ekleme şeklinde gerçekleşir. (Aynı işlem Data Select Test başlığı altında da yapıldığı için, burada tekrar detaylandırılmamıştır)

Buraya kadar olan işlemler tamamlandıktan sonra, son adım olarak, eklenen verilerin doğru ve istenen sayıda eklendiğini kontrol etme işlemi gerçekleştirilir; koleksiyona sonradan eklenen N tane , yani *count* kadar, veriyi ve daha önceden koleksiyona eklemek için üretilen veriler-lastDatas değişkeninde tutulan- karşılıklı olarak kıyaslanır. Bu şekilde hem istenen sayıda hem de doğru şekilde eklenip eklenmedikleri kontrol edilir. Kod tarafında açıklayacak olursak; koleksiyona eklenen N/count tane veriye ulaşabilmek için, IQueryable<BsonDocument> sorgu gerçeklenmiştir. Bu tip bir sorguyu gerçekleştirebilmek, yani koleksiyona en son eklenen N tane veriye ulaşmak, için yapılması gerekenler şu şekildedir; *FindAllAs<BsonDocument>* komutu ile MongoDB'deki bir koleksiyonda aranacak olan verilerin olduğu yer tespit edilir; *SetSortOrder()* komutu, içine alacağı değişken ile gerekli sıralama işlemlerini gerçekleyerek, ardından kullanılacak olan ve N tane dokümana ulaşmayı sağlayacak olan *SetLimit()* komutunun kullanılmasını sağlar. Buna göre;

```
databasesLastNDocument =
myCollection.FindAllAs<BsonDocument>().SetSortOrder(mySortByBuilder).SetLimit(count
);
```

Yukarıdaki sorgu sonucunda, seçili koleksiyondan N tane veri *databasesLastNDocument* ismi verilen değişkene sondan başa doğru olacak şekilde atanır. Ve daha önceden koleksiyona kayıt edilmiş verileri tutan lastDatas listesi tersine çevrilir.

```
lastDatas.Reverse();
```

databasesLastNDocument sorgusunda bulunan bilgiler bir listeye atanır, bu şekilde lastDatas listesi içindeki dokümanlar ile kıyaslama yapmak daha kolay olur. Son adım olarak, koleksiyondan çekilen N tane doküman ile koleksiyona sonradan eklenen veriler kıyaslanarak doğru biçimde eklenip eklenmedikleri kontrol edilir. Bu işlem için bir döngü yardımıyla, iki listeninde karşılıklı elemanları kıyaslanır ve uyumlu olup olmadıkları bilgisi daha önceden tanımlanmış ve ilk değer olarak *true,* yani eşitliğin sağlandığı bilgisini almış, *areDatasEqual* mantıksal değişkenine işlenir.

```
for (int i = 0; i < count; i++)
        if (!lastDatas[i].Equals(databasesLastNDocumentList[i]))
            areDatasEqual = false;
```

Tüm işlemler tamamlandıktan sonra üretilen verilerin, koleksiyona istenilen sayıda ve doğru şekilde eklenip eklenmediği bilgisi kullanılarak, kullanıcı bilgilendirilir.

5.2.3. Veri Silme Testi

Veri Silme Testi ile kullanıcının seçtiği koleksiyondan yine kullanıcının belirlediği sayıda verinin silinebilme durumu kontrol edilir. Eğer veriler silinebildiyse, bu silinen veriler ekranda tablo içerisinde listelenir. Eğer silinemediyseler, "Veriler silinememiştir" şeklinde bir mesaj kullanıcı ekranında gösterilir. Test için koleksiyondan silinecek olan veriler, önce CollectionRandomInsert sınıfı kullanılarak üretilir, koleksiyona eklenir; ardından bu üretilen veriler koleksiyondan silinir. Bu sayede, koleksiyona ait orjinal dokümanlarda bir değişiklik yapılmamış olur.

Kullanıcı menü başlığı altında ilgili bölümü seçtiğinde, *dataRemoveTest_click()* fonksiyonu ile, "DataTest.aspx" sayfasına yönlendirilir, veri testleri için gerçekleşen tüm işlemler ve sonuçları bu sayfa ile bağlantılıdır. dataSelectionTest_click() fonksiyonu içerisinde sayfalar arası veri aktarımı için kullanılan *Session["sonuc"] = "remove"* güncellemesi yapılır. Bu işlemin yapılmasının sebebi, sayfalar arası geçişte, hangi testin gerçekleşeceğine kolaylıkla karar vermektir. DataTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassDataTests sınıfındaki *DoDataRemoveTest()* fonksiyonundan faydalanılır.

DoDataRemoveTest() fonksiyonu, *collection*, *myGridViewTable* ve *count* isimlerinde üç parametre almaktadır. Bu parametrelerden ilki, yani collection, üzerinde işlem yapılacak olan koleksiyon bilgisini içeren karakter tipinde bir değişkendir. İkinci parametre, myGridViewTable, verileri listelemek için kullanılacak olan DataTable nesnesi ve sonuncu parametrede, koleksiyondan kaç tane veri silinmesi gerektiğini belirten *count* isminde tamsayı değişkenidir.

Seçilen koleksiyon(collection) MongoDB'nin bir koleksiyonu olduğu ve C# içerisinde kullanılabilmesi için aktifleştirilmesi gerekir; FindDatabasesCollections sınıfının getCollection() fonksiyonu kullanılarak, gelen koleksiyon ismi bilgisi kullanılarak, veritabanına ait Mongo koleksiyonu aktifleştirilir ve *myCollection* isimli değişkende gelen koleksiyon bilgisi tutulur;

```
FindDatabasesCollections finder=new
FindDatabasesCollections(Formlar.connectionString);
MongoCollection myCollection = finder.getCollection(collection);
```

Seçilen koleksiyonda bulunan verilerde bir değişiklik yapmadan istenilen sayıda veriyi silmek için, önce veri ekleyip ardından eklenen verileri silme işlemi gerçekleneceğinden, öncelikle, kullanıcının belirlediği sayıda veri üretilerek koleksiyona eklenir. Bu işlem için bir döngü kullanılır ve bu döngü istenilen sayıda veri eklenene kadar gerçeklenir.

```
for (int i = 0; i < count; i++){}
```

Döngünün içindeki işlemler ise şu şekildedir; CollectionRandomInsert sınıfı kullanılarak üretilen veri *lastData* isminde ve BsonDocument tipindeki değişkene atanır, ardından, üretilen tüm verilere rahatlıkla ulaşabilmek için kullanılacak olan *lastDatas* isimli liste tipindeki değişkene, lastData eklenir. Bunun nedeni, ilerleyen satırlarda, koleksiyona eklenen doküman sayısının doğru olup olmadığını kontrol edebilmektir.

```
lastData = InsertRandomDataInCollection(collection);
lastDatas.Add(lastData);
```

Eklenecek veriyi belirleme işlemi gerçekleştikten sonra, veri koleksiyona *Save()* komutu kullanılarak eklenir.

Koleksiyona üzerinde işlem yapılacak olan veriler ekledikten sonra, koleksiyon içerisinden verileri silme işlemi gerçekleştirilir. Veri silme işleminde dokümanlara rahatlıkla ulaşabilmek için lastDatas listesi içerisindeki değerler kullanılacaktır. Zaten bu değerler koleksiyona eklenen, dolayısıyla koleksiyondan silinmesi gereken değerlerdir. lastDatas listesi içindeki her elemana tek tek ulaşabilmek için bir döngüden faydalanılır.

```
foreach (var item in lastDatas){}
```

Koleksiyon içinden bir dokümanı silebilmek için, o veriye en belirgin özelliğini kullanarak ulaşmak gerekir. lastDatas içerisindeki her veri koleksiyondan silinmesi gereken BsonDocument tipinde bir doküman şeklindedir. Dolayısıyla döngü içerisinden gelecek olan dokümanın anahtar/değer çiftine indis kullanılarak erişilebilir. Burada item[0] elemanı o dokümanın "_id" bilgisini verir, koleksiyondan silinecek veriye de, "_id" bilgisi yardımı ile ulaşılır. Bunun için IQueryable<BsonDocument> şeklinde "_id" bilgisini bulacak bir sorgu gerçekleştirilir;

```
buildDelete =
myCollection.AsQueryable<BsonDocument>().Where(p=>p["_id"].Equals(deleteThisId));
```

MongoDB içerisinden bir dokümanı C# kullanarak silmek için, *IMongoQuery* tipinde bir sorgu gerçekleştirmek gerekir. Bu yüzden buildDelete sorgusu, IMongoQuery şekline *GetMongoQuery()* komutu yardımı ile çevrilir ve dönüşümden sonra, istenen doküman koleksiyon içerisinden *Remove()* komutu kullanılarak silinir.

```
bDelete = (buildDelete as MongoQueryable<BsonDocument>).GetMongoQuery();
```

Silme işleminin başarılı bir şekilde gerçekleşip gerçekleşmediğini anlamak için, yukarıdaki gibi lastDatas listesinin elemanları, koleksiyon içerisinde aratılır. Arama sonucunda sorgunun aranan elemanı içermemesi, istenen dokümanın koleksiyondan silindiğini; içermesi ise, işlemin başarısız olduğunu yani koleksiyondan eleman silinemediğini gösterir.

```
foreach (var item in lastDatas)
 {
    ObjectId deleteThisId = item[0].AsObjectId;
```

```
        var buildDelete = myCollection.AsQueryable<BsonDocument>().Where(p =>
p["_id"].Equals(deleteThisId));
        if (buildDelete.Any())
        {
           remove = false;
           break;
        }
        else
           remove = true;
     }
```

Burada kullanılan Any() fonksiyonu, bir sorgunun içinin boş olup olmadığını kontrol eder. Yapılan sorgunun içi boş değil ise, silinmesi gereken eleman koleksiyondan silinememiştir. Bu durumu kullanıcıya bildirmek için, daha önceden tanımlanmış *remove* ismindeki mantıksal değişkenin değeri *false* olarak atanır ve koleksiyon içerisinde daha fazla sorgu gerçekleştirilmez(Silinmesi gereken kayıtlardan bir tanesinin bile silinememiş olması, silme işleminin başarısız olduğu anlamına gelir). Sorgunun içi boş ise silme işlemi başarılıdır ve remove değişkenine true bilgisi atanır.

Tüm işlemler tamamlandıktan sonra, silme işleminin başarılı olması, yani remove değişkenin değerinin true olması durumunda silinen dokümanlar, kullanıcıya gösterilmek üzere tablo içinde listelenir(Bu işlem Data Selection Test başlığı altında gerçekleştirildiğinden, tekrar anlatılmamıştır). Ayrıca geriye dönüş bilgisi olarak, işlemin başarılı olduğu mesajı gönderilir. Silme işleminin başarısız olması, yani remove değişkeninde false bilgisinin olması durumunda ise herhangi bir işlem yapılmaz, sadece geriye yapılan işlemin başarısız olduğu mesajı gönderilir.

5.2.4. Veri Güncelleme Testi

Son gerçekleştirilen test ise, Veri Güncelleme Testi'dir. Bu testte kullanıcının seçtiği koleksiyonda yine kullanıcının belirlediği sayıda verinin güncellenebilmesi durumu kontrol edilir. Eğer veriler güncellendiyse, güncellenen veriler ekranda tablo içerisinde listelenir. Eğer, güncellenemediyseler "Veriler güncellenememiştir" şeklinde bir mesaj kullanıcı ekranında gösterilir. Test için koleksiyondan silinecek olan veriler, önce CollectionRandomInsert sınıfı kullanılarak üretilir, koleksiyona eklenir; ardından bu üretilen veriler koleksiyondan silinir. Bu sayede, koleksiyona ait orjinal dokümanlarda bir değişiklik yapılmamış olur.

Kullanıcı menü başlığı altında ilgili bölümü seçtiğinde, *dataUpdateTest_click()* fonksiyonu ile, "DataTest.aspx" sayfasına yönlendirilir, veri testleri için gerçekleşen tüm işlemler ve

sonuçları bu sayfa ile bağlantılıdır. dataSelectionTest_click() fonksiyonu içerisinde sayfalar arası veri aktarımı için kullanılan *Session["sonuc"] = "update"* güncellemesi yapılır. Bu işlemin yapılmasının sebebi, sayfalar arası geçişte, hangi testin gerçekleşeceğine kolaylıkla karar vermektir. DataTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassDataTests sınıfındaki *DoDataUpdateTest()* fonksiyonundan faydalanılır.

DoDataUpdateTest() fonksiyonu, *collection*, *myGridViewTable* ve *count* isimlerinde üç parametre almaktadır. Bu parametrelerden ilki, yani collection, üzerinde işlem yapılacak olan koleksiyon bilgisini içeren karakter tipinde bir değişkendir. İkinci parametre, myGridViewTable, verileri listelemek için kullanılacak olan DataTable nesnesi ve sonuncu parametrede, koleksiyonda kaç tane veri güncellenmesi gerektiğini belirten *count* isminde tamsayı değişkenidir.

Seçilen koleksiyon(collection) MongoDB'nin bir koleksiyonu olduğu ve C# içerisinde kullanılabilmesi için aktifleştirilmesi gerekir; FindDatabasesCollections sınıfının getCollection() fonksiyonu kullanılarak, gelen koleksiyon ismi bilgisi kullanılarak, veritabanına ait Mongo koleksiyonu aktifleştirilir ve *myCollection* isimli değişkende gelen koleksiyon bilgisi tutulur;

```
FindDatabasesCollections finder=new
FindDatabasesCollections(Formlar.connectionString);
MongoCollection myCollection = finder.getCollection(collection);
```

Seçilen koleksiyonda bulunan verilerde bir değişiklik yapmadan istenilen sayıda veriyi güncellemek için, önce veri ekleyip ardından eklenen verileri güncelleme işlemi gerçekleneceğinden, öncelikle, kullanıcının belirlediği sayıda veri üretilerek koleksiyona eklenir. Bu işlem için bir döngü kullanılır ve bu döngü istenilen sayıda veri eklenene kadar gerçeklenir.

```
for (int i = 0; i < count; i++){}
```

Döngününün içinde;, CollectionRandomInsert sınıfı kullanılarak üretilen veriler *lastData* isminde ve BsonDocument tipindeki değişkene atanır, ardından, üretilen tüm verilere rahatlıkla ulaşabilmek için kullanılan *lastDatas* isimli liste tipindeki değişkene, lastData eklenir. Bunun nedeni, ilerleyen satırlarda, koleksiyona eklenen doküman sayısının doğru olup olmadığını kontrol edebilmektir.

```
lastData = InsertRandomDataInCollection(collection);
lastDatas.Add(lastData);
```

Eklenecek veriyi belirleme işlemi gerçekleştikten sonra, veri koleksiyona *Save()* komutu kullanılarak eklenir. İstenilen sayıda veriyi koleksiyona ekleme işlemini gerçekleştirip tabloya ekledikten sonra, koleksiyona ait eski/asıl veriler de tabloya eklenir. Bu işlem, daha önceden koleksiyondan okunarak *query* isimli sorguda tutulan dokümanları, bir döngü

yardımı ile okuyup tabloya ekleme şeklinde gerçekleşir. (Aynı işlem Data Select Test başlığı altında da yapıldığı için, burada tekrar detaylandırılmamıştır)

Koleksiyona üzerinde işlem yapılacak olan veriler ekledikten sonra, koleksiyon içerisinden verileri güncelleme işlemi gerçekleştirilir. Veri güncelleme işleminde dokümanlara rahatlıkla ulaşabilmek için lastDatas listesi içerisindeki değerler kullanılacaktır. Zaten bu değerler koleksiyona eklenen, dolayısıyla koleksiyonda güncellenmesi gereken değerlerdir. lastDatas listesi içindeki her elemana tek tek ulaşabilmek için bir döngüden faydalanılır.

foreach (var item in lastDatas){}

Koleksiyon içinden bir dokümanı güncelleyebilmek için, o veriye en belirgin özelliğini kullanarak ulaşmak gerekir. lastDatas içerisindeki her veri koleksiyonda güncellenmesi gereken BsonDocument tipinde bir doküman şeklindedir. Dolayısıyla döngü içerisinden gelecek olan dokümanın anahtar/değer çiftine indis kullanılarak erişilebilir. Burada item[0] elemanı o dokümanın "_id" bilgisini verir, koleksiyonda güncellenecek olan veriye de, "_id" bilgisi yardımı ile ulaşılır. Bunun için IQueryable<BsonDocument> şeklinde "_id" bilgisini bulacak bir sorgu gerçekleştirilir;

buildUpdate =
myCollection.AsQueryable<BsonDocument>().Where(p=>p["_id"].Equals(deleteThisId));

MongoDB içerisinden bir dokümanı C# kullanarak güncellemek için, *IMongoQuery* tipinde bir sorgu gerçekleştirmek gerekir. Bu yüzden buildUpdate sorgusu, IMongoQuery şekline *GetMongoQuery()* komutu yardımı ile çevrilir. Dönüşümden sonra, doküman üzerinde güncelleme işlemi yapılır. Bunun için dokümanın anahtar bilgisine ve değer tipine ulaşılır ve her biri için tiplerine uygun olarak yeni elemanlar üretilir. Üretilen yeni elemanlar anahtar bilgilerine göre koleksiyondan güncelleneceklerinden, yeni elemanlar öncelikle BsonDocument şeklindeki bir listeye eklenirler. Ve *.Update()* komutu kullanılarak, liste içerisinde bulunan elemanlar ile koleksiyon güncellenir.

Güncelleme işleminin gerçekleşmesi durumunda, koleksiyona son eklenen/güncellenen veriler koleksiyon içerisinden okunur ve eski veriler ile birlikte tabloya listelenir(Koleksiyon içinden N/count tane veri çekme işlemi Data Insert Test başlığında açıklanmıştır).

5.3. Replikasyon Testleri

Yapılan bir diğer test de replikasyon ile ilgili testlerdir. Replikasyon Testleri, replikasyon yapılmış bir veritabanının replika üyelerini inceler. Bu testlerin istenen şekilde sonuçlanabilmesi için, replikasyon yapılmış veritabanları üzerinde uygulanmaları gerekmektedir. Menüde Replikasyon Testleri başlığına tıklandığında, *replicaTests_click()*

yardımı ile sistem “ReplicaTestsMenuPage.aspx” sayfasına yönlendirir. Bu sayfa içerisinde, başlığın altındaki testler ile ilgili genel bir bilgi verilmektedir.

Replikasyon Testleri için yapılan her test ClassReplicaTest sınıfı içerisinde gerçekleştirilir

Kullanıcı Replica Test başlığı altındaki testlerden yapmak istediğinde, 6 tane seçenek ile karşılaşır. Ve bu 6 test “ReplicaTest.aspx” sayfasına yönlendirilerek gerçekleştirilir. Ayrıca bu testler ClassReplicaTests sınıfı kullanılarak yapılır. ReplicaTest.aspx sayfasında; replicaLabel isminde bir Label nesnesi bulunur. Yapılan testlerin sonuçları bu replicaLabel üzerinden kullanıcıya gösterilir. ReplicaTest.aspx’in kod kısmında, hangi testin yapılacağına dair karar verilerek ona uygun işlemler gerçekleştirilmektedir. Bunun için, her test için ayrıca tanımlanan ve bir değer atayanan *Session["replica"]* nesnesi kullanılır. *Session["replica"]*, tüm testler ReplicaTest.aspx sayfasında gerçeklendiği için, kullanıcının hangi testi seçtiğine karar vermek için kullanılır. *Session["replica"]* ile atanan değerler testleri açıklarken belirtilmiştir. Hangi testin gerçekleneceğine ise switch-case komutu yardımı ile karar verilmiştir.

Kullanıcı menü başlığı altında ilgili bölümü seçtiğinde, o test ile ilgili yordam/fonksiyon ile “ReplicaTests.aspx” sayfasına yönlendirilir, replikasyon ile yapılan tüm testler ve sonuçları bu sayfa ile bağlantılıdır. İlgili yordam/fonksiyon içerisinde sayfalar arası veri aktarımı için kullanılan *Session["replica"]* = güncellemesi yapılır. Bu işlemin yapılmasının sebebi, sayfalar arası geçişte, hangi testin gerçekleşeceğine kolaylıkla karar vermektir. ReplicaTest.aspx sayfasında hangi testin yapılacağına karar verilir.

Yapılan tüm testlerde, veritabanı içinde işlem yapılma süresi hakkında bir bilgi kullanıcı ile paylaşılmıştır. Testler aşağıda detaylıca anlatılmıştır.

5.3.1. Birincil Üye Kontrolü

Replikasyon Testleri menü başlığı altındaki testlerden ilkidir. MongoDB‘de her veritabanı Birincil(Primary) bir elemana sahiptir. Dolayısıyla bu test, veritabanında replikasyon uygulanmamış olsa dahi *True* sonucunu verecektir.

Sayfalar arası veri aktarımı ve yapılacak testin “Has got a Primary Member?” olduğunu belirtmek için *Session["replica"] = "primary"* güncellemesi yapılır.

```
Session["replica"] = "primary";
Response.Redirect("ReplicaTest.aspx");
```

ReplicaTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassReplicaTests sınıfındaki *LookDatabasePrimaryMember()* fonksiyonundan faydalanılır.

LookDatabasePrimaryMember() fonksiyonu içerisinde; sunucunun içerdiği elemanları belirleyebilmek için sunucu bilgilerine dolayısıyla bağlantı dizesine ihtiyaç vardır. Bunun için FindDatabasesCollections sınıfı tipinde *finder* isimli bir değişken ve MongoServer tipinde *server* isimli bir değişken tanımlanır; *finder* yardımı ile FindDatabasesCollections sınıfı içerisindeki getServer() fonksiyonu kullanılarak, sunucu belirlenir;

```
MongoServer server = finder.getServer();
```

Sunucu belirlendikten sonra; önce sunucunun elemanlarına *.Instances* komutu ile ulaşılır ardından bu elemanların sayısı *instancesLength* isimli değişken içerisine *.Length* komutu yardımı ile atanır;

```
int instancesLength = server.Instances.Length;
```

Dolayısıyla yapılan ikinci işlem, bu eleman sayısı yardımı ile tek tek elemanları kontrol etmektir. Bunun için bir döngü kullanılmış ve sunucunun her elemanının Birincil eleman olma özelliği *.IsPrimary* komutu yardımı ile aşağıdaki gibi kontrol edilmiştir;

```
if (server.Instances[i].IsPrimary)
    isPrimary = true;
```

Buradaki *isPrimary* değişkeni veritabanının Birincil eleman içermesi durumunu belirlemek için tanımlanmış, mantıksal bir değişkendir. Son adım olarakta, isPrimary değişkeninin doğru olması durumunda "Your Database Has a Primary Member!", yanlış olması durumunda ise "Your Database Does Not Has a Primary Member!" mesajınları ekrana çıkartmak için geri bilgi olarak gönderilir.

5.3.2. İkincil Üye kontrolü

Replikasyon Testleri menü başlığı altındaki testlerden bir diğeridir. MongoDB'de replikasyon yapılmış her veritabanı İkincil(Secondary) –bir veya birden fazla- elemana sahip olabilir. Dolayısıyla bu test, veritabanına replikasyon uygulanmış ise İkincil elemanların varlığını kontrol etmek için tasarlanmıştır.

Sayfalar arası veri aktarımı ve yapılacak testin "Has got a Secondary Member?" olduğunu belirtmek için *Session["replica"] = "secondary"* güncellemesi yapılır.

```
Session["replica"] = "secondary";
Response.Redirect("ReplicaTest.aspx");
```

ReplicaTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassReplicaTests sınıfındaki *LookDatabaseSecondaryMember()* fonksiyonundan faydalanılır.

LookDatabaseSecondaryMember() fonksiyonu içerisinde; sunucunun içerdiği elemanları belirleyebilmek için sunucu bilgilerine dolayısıyla bağlantı dizesine ihtiyaç vardır. Bunun için FindDatabasesCollections sınıfı tipinde *finder* isimli bir değişken ve MongoServer tipinde *server* isimli bir değişken tanımlanır; *finder* yardımı ile FindDatabasesCollections sınıfı içerisindeki getServer() fonksiyonu kullanılarak, sunucu belirlenir;

```
MongoServer server = finder.getServer();
```

Sunucu belirlendikten sonra; önce sunucunun elemanlarına *.Instances* komutu ile ulaşılır ardından bu elemanların sayısı *instancesLength* isimli değişken içerisine *.Length* komutu yardımı ile atanır;

```
int instancesLength = server.Instances.Length;
```

Dolayısıyla yapılan ikinci işlem, bu eleman sayısı yardımı ile tek tek elemanları kontrol etmektir. Bunun için bir döngü kullanılmış ve sunucunun her elemanının İkincil eleman olma özelliği *.IsSecondary* komutu yardımı ile aşağıdaki gibi kontrol edilmiştir;

```
if (server.Instances[i].IsSecondary)
    isSecondary = true;
```

Buradaki *isSecondary* değişkeni veritabanının İkincil eleman içermesi durumunu belirlemek için tanımlanmış, mantıksal bir değişkendir. Son adım olarakta, isSecondary değişkeninin doğru olması durumunda "Your Database Has Secondary Member!", yanlış olması durumunda ise "Your Database Does Not Has Secondary Member!" mesajıları ekrana çıkartmak için geri bilgi olarak gönderilir.

5.3.3. Hakem Üte Kontrolü

MongoDB'de replikasyon yapılmış her veritabanı Arbiter elemana sahip olabilir. Dolayısıyla bu test, veritabanına replikasyon uygulanmış ise Arbiter elemanların varlığını kontrol etmek için tasarlanmıştır.

Sayfalar arası veri aktarımı ve yapılacak testin "Has got a Arbiter Member?" olduğunu belirtmek için *Session["replica"] = "arbiter"* güncellemesi yapılır.

```
Session["replica"] = "arbiter";
Response.Redirect("ReplicaTest.aspx");
```

ReplicaTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassReplicaTests sınıfındaki *LookDatabaseArbiterMember()* fonksiyonundan faydalanılır..

LookDatabaseArbiterMember() fonksiyonu içerisinde; sunucunun içerdiği elemanları belirleyebilmek için sunucu bilgilerine dolayısıyla bağlantı dizesine ihtiyaç vardır. Bunun için FindDatabasesCollections sınıfı tipinde *finder* isimli bir değişken ve MongoServer

tipinde *server* isimli bir değişken tanımlanır; *finder* yardımı ile FindDatabasesCollections sınıfı içerisindeki getServer() fonksiyonu kullanılarak, sunucu belirlenir;

MongoServer server = finder.getServer();

Sunucu belirlendikten sonra; önce sunucunun elemanlarına *.Instances* komutu ile ulaşılır ardından bu elemanların sayısı *instancesLength* isimli değişken içerisine *.Length* komutu yardımı ile atanır;

int instancesLength = server.Instances.Length;

Dolayısıyla yapılan ikinci işlem, bu eleman sayısı yardımı ile tek tek elemanları kontrol etmektir. Bunun için bir döngü kullanılmış ve sunucunun her elemanının Arbiter eleman olma özelliği *.IsArbiter* komutu yardımı ile aşağıdaki gibi kontrol edilmiştir;

if (server.Instances[i].IsArbiter)
isArbiter = true;

Buradaki *isArbiter* değişkeni veritabanının Arbiter eleman içermesi durumunu belirlemek için tanımlanmış, mantıksal bir değişkendir. Son adım olarakta, isArbiter değişkeninin doğru olması durumunda "Your Database Has Arbiter Member!", yanlış olması durumunda ise "Your Database Does Not Has Arbiter Member!" mesajları ekrana çıkartmak için geri bilgi olarak gönderilir.

5.3.4. Saklı Üye Kontrolü

MongoDB'de replikasyon yapılmış her veritabanı Hidden elemana sahip olabilir. Dolayısıyla bu test, veritabanına replikasyon uygulanmış ise Hidden elemanların varlığını kontrol etmek için tasarlanmıştır.

Sayfalar arası veri aktarımı ve yapılacak testin "Has got a Hidden Member?" olduğunu belirtmek için *Session["replica"] = "hidden"* güncellemesi yapılır.

Session["replica"] = "hidden";
Response.Redirect("ReplicaTest.aspx");

ReplicaTest.aspx sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassReplicaTests sınıfındaki *LookDatabaseHiddenMember()* fonksiyonundan faydalanılır.

LookDatabaseHiddenMember() fonksiyonu içerisinde gerçekleştirdiğimiz işlemler ise şu şekildedir; sunucunun içerdiği elemanları belirleyebilmek için sunucu bilgilerine dolayısıyla bağlantı dizesine ihtiyaç vardır. Bunun için FindDatabasesCollections sınıfı tipinde *finder* isimli bir değişken ve MongoServer tipinde *server* isimli bir değişken tanımlandı; *finder*

yardımı ile FindDatabasesCollections sınıfı içerisindeki getServer() fonksiyonu kullanılarak, sunucu belirlendi;

MongoServer server = finder.getServer();

Sunucu belirlendikten sonra; önce sunucunun elemanlarına *.Instances* komutu ile ulaşılır ardından bu elemanların sayısı *instancesLength* isimli değişken içerisine *.Length* komutu yardımı ile atanır;

int instancesLength = server.Instances.Length;

Dolayısıyla yapılan ikinci işlem, bu eleman sayısı yardımı ile tek tek elemanları kontrol etmektir. Bunun için bir döngü kullanılmış ve sunucunun her elemanının Hidden eleman olma özelliği *.IsHidden* komutu yardımı ile aşağıdaki gibi kontrol edilmiştir;

if (server.Instances[i].IsHidden)
isArbiter = true;

Buradaki *isHiddenr* değişkeni sistemimizin Hidden eleman içermesi durumunu belirlemek için tanımlanmış, mantıksal bir değişkendir. Son adım olarakta, isHidden değişkeninin doğru olması durumunda "Your Database Has Hidden Member!", yanlış olması durumunda ise "Your Database Does Not Has Hidden Member!" mesajınları ekrana çıkartmak için geri bilgi olarak gönderilir.

5.3.5. Kullanıcı Tanımlı Ayarların Kontrolü

Replika kümesini oluşturan üyeler dışında, kullanıcı oluşturduğu sistemin bilgilerinin doğruluğunu da kontrol etmek isteyebilir. “Settings are true?” testi ile, replikasyon yapılmış veritabanının ayarlarına ulaşılır ve bu ayarların doğruluğunu kullanıcının teyit etmesi istenir.

Sayfalar arası veri aktarımı ve yapılacak testin “Settings are True?” olduğunu belirtmek için *Session["replica"] = "settings"* güncellemesi yapılır.

Session["replica"] = "settings";
Response.Redirect("ReplicaTest.aspx");

“ReplicaTest.aspx” Sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassReplicaTests sınıfındaki *LookDatabaseSettingsMember()* fonksiyonundan faydalanılır.

LookDatabaseSettingsMember() fonksiyonunda veritabanının ayarlarına ulaşılır. Bunun için öncelikle sunucu bilgileri bulunur ve *server* isimli değişkene atanır, ardından *settings* isimli ve MongoServerSettings tipindeki değişkene *.Settings()* fonksiyonu kullanılarak, sunucu/veritabanı bilgileri atanır;

MongoServerSettings settings = server.Settings;

Ve bu settings bilgisi kullanıcı ekranında gösterilmek üzere geri bilgi olarak gönderilir.

ReplicaTest.aspx.cs sayfasında gelen bilgi yine *settings* ismi verilmiş değişkene aktarılır. Ve bu sayede, *ListSettingParameters(settings)* fonksiyonu aktifleştirilir. Peki ne yapar bu fonksiyon?
ListSettingsParameters(settings) fonksiyonu, içerisine gelen veritabanı ayarlarını sayfada bulunan checkBox nesnelerine ekler, bu sayede kullanıcı doğru olan ayarları seçebilir. Fonksiyonda yapılan işlemler ise şu şekildedir; settings parametresinde bulunan bilgiler, daha önceden ReplicaTest.aspx sayfasına eklenmiş olan checkBox nesnelerine tek tek yerleştirilir. Settings parametresinde bulunan bilgilere ulaşmak için, hazır olarak bulunan şu fonksiyonlardan faydalanır; *.ReadPreference* sistemin o an ki işlem modu hakkındaki bilgiyi, *.ReplicaSetName* replikasyon kümesinin ismini, *.Servers* replikasyon yapmak için kullanılan sunucuların bilgilerini, *.WriteConcern* yapılan işlemlerin writeConcern durumları ile ilgili bilgiyi, *.ConnectionMode* replika kümesinin bağlantı modunu, *.Credentials* replika kümesinin kullandığı diğer bilgileri verir. Kullanıcı bunlardan doğru olanları seçip, "Next" yazan replicaTestSettingsButton nesnesine tıkladığında, seçilen checkBox nesnesi sayısını bulur ve testin başarısı ile ilgili bir mesaj alır.

5.3.6. Admin Koleksiyonunun Üye Kontrolü

Kullanıcıların oluşturdukları koleksiyonlar dışında MongoDB içerisinde sabit olarak bulunan ve hazır gelen veritabanları ve koleksiyonları da bulunmaktadır. Bu test kullanıcının oluşturduğu veritabanı dışında kalan veritabanını yani, "admin" veritabanını kontrol etmek için tasarlanmıştır. CountingFunctions sınıfını da kullanarak, "admin" veritabanında bulunan Birincil ve İkincil üye sayısını kullanıcıya bildirir.

Sayfalar arası veri aktarımı ve yapılacak testin "Three Member Set Scenario Should Be Started." olduğunu belirtmek için *Session["replica"] = "three"* güncellemesi yapılır.

Session["replica"] = "three";
Response.Redirect("ReplicaTest.aspx");

"ReplicaTest.aspx" Sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassReplicaTests sınıfındaki *ThreeMemberSetScenarioShouldBeStarted()* fonksiyonundan faydalanılır.

ThreeMemberSetScenarioShouldBeStarted() fonksiyonu içerisinde yapılanlar şu şekildedir; FindDatabasesCollections sınıfı tipinde *finder* isimli bir değişken ve MongoServer tipinde

server isimli bir değişken tanımlanır ve *finder* yardımı ile FindDatabasesCollections sınıfı içerisindeki getServer() fonksiyonu kullanılarak, sunucu belirlenir;

MongoServer server = finder.getServer();

Sunucu belirlendikten sonra; ".GetDatabase("admin")" komutu ile, admin veritabanına bağlantı sağlanır. Bunun sebebi, bu veritabanındaki replikasyon sistemi ile ilgili bilgilere ulaşabilmek için veritabanına bağlı olunması gerektiğidir. Veritabanına ulaştıktan sonra, içerisindeki replikasyon bilgilerine ulaşmak için, bir MongoDB komutu olan "replSetGetStatus" komutu .RunCommand yardımı ile aşağıdaki gibi çalıştırılır.

CommandResult doc = server.GetDatabase("admin").RunCommand("replSetGetStatus");

Burada tanımlanmış olan *doc* CommandResult tipindedir. CommandResult, C# aracılığı ile MongoDB komutlarını çalıştırmayı sağlar.

Veritabanına ait replikasyon bilgilerine ulaştıktan sonra, sınıflar başlığı altında tanımlanan CountingFunctions sınıfından faydalanılır. Bu sınıf ve fonksiyonlarının yaptığı işlemler, CountingFunctions başlığı altında detaylıca anlatılmıştır. Dolayısıyla burada test ile alakalı olan kısımdan bahsedilmiştir. CountingFunctions sınıfı ClassReplicaTests sınıfının Base Class'ı olduğundan, fonksiyonları rahatlıkla kullanılır. Sınıfın fonksiyonlarından önce countPrimary(doc) daha sonra da countSecondary(doc) fonksiyonları kullanılır ve sonuçlarını sırası ile *countOfPrimary* ve *countOfSecondary* tamsayı değişkenlerine atanır. Ve işlemin sonunda kullanıcı ekranına mesaj çıkartmak üzere,

"Primary sayisi: " + countOfPrimary.ToString() + " Secondary sayisi: " + countOfSecondary.ToString()"

bilgi olarak gönderilir.

5.3.7. Replika Kümelerinin Tutarlılığı Kontrolü

Replikasyon yapılan veritabanlarında yaşanan en büyük problem veriler arasında tutarlılık sağlanamamasıdır. Yani bir sunucu içerisindeki koleksiyona kayıt edilen veri başka bir sunucu içindeki koleksiyona kayıt edilmeyebilir veya yanlış kayıt edilebilir. Özellikle büyük önem taşıyan veritabanı sistemlerinde, bu tip bir hata çok büyük kayıplara sebep olmaktadır. Bu durum göz önünde bulundurularak farklı sunucular içinde bulunan replika edilmiş veritabanlarının eşitliği test edilmiştir. Genel olarak testin içeriği şu şekildedir; kullanıcının ekrandaki bir listeden seçeceği koleksiyonun tüm sunuculardaki kayıtları kontrol edilir ve herhangi bir uyumsuzluk olması durumunda ekrana bu bilgiler listelenir. Test için sisteme giriş esnasında tüm kayıt bilgileri bir XML dosyasına kayıt edilmiş ve gereken yerde bu dosyadan çekilerek kullanılmıştır.

Sayfalar arası veri aktarımı ve yapılacak testin "TestReplicaEquation" olduğunu belirtmek için *Session["replica"] = "equalreplica"* güncellemesi yapılır. Ayrıca ekranda bulunan bir liste içerisinden kullanıcıya bir koleksiyon seçtirilir.

```
Session["replica"] = "equalreplica";
Response.Redirect("ReplicaTest.aspx");
```

"ReplicaTest.aspx" Sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassReplicaTests sınıfındaki *TestReplicaEquation()* fonksiyonundan faydalanılır.

TestReplicaEquation() fonksiyonu içerisinde yapılanlar şu şekildedir; daha önceden bir XML dosyası içine kayıt edilen kayıt bilgileri dosyanın içerisinden okunur ve kullanıcıdan alınmış olan koleksiyon her bir sunucu içerisinde tek tek incelenir. Her sunucudaki koleksiyonun içinde bulunan kayıtlar bir DataTable içerisine kayıt edilir. Bu esnasında kayıt ile birlikte hangi ip/hostname bilgisine sahip sunucuda olduğuna dair bir bilgide DataTable içerisine "iport" isimli bir kolon ile kayıt edilir. Tüm sunuculardaki kayıtlar okunarak DataTable içine kayıt edildiği zaman, kayıtların eşitliği kontrol edilir. Bu eşitliği ve hangi sunucuya ait kayıda bakılacağına karar vermek için de "iport" isimli tablo kolonu kullanılır.

5.4. Genel Testler

Veritabanını ve içerisindeki koleksiyonları detaylı olarak inceleme amaçlı geliştirilmiş testlerdir. 15 tane test geliştirilmiştir ve bu testlerin sonuçları bir tablo halinde kullanıcı ekranında gösterilir. Testlerin nasıl yapıldığı ve detayları Classlar başlığında ClassGeneralDatabaseTesting isimli sınıfta detaylıca anlatılmıştır. Aşağıda yapılan testler hakkında kısa açıklamalar bulunmaktadır;

- *TestDbName;* Veritabanının isminin durumunu test eder.
- *TestDbNull;* Veritabanının içinin boş olması durumunu test eder.
- *TestDbCollections;* Veritabanının içinin dolu olması yani koleksiyon bulundurmasını test eder.
- *TestNewDbCollection;* Veritabanına yeni eklenecek koleksiyonun, sistemde daha önceden kayıtlı olma durumunu test eder.
- *TestNewDbCollectionName;* Veritabanına yeni eklenen koleksiyonun, isminin doğru şekilde eklenmesini test eder.
- *TestNewDbCollectionDocument;* Veritabanında yeni oluşturulan koleksiyona veri eklemenin doğru şekilde gerçekleşmesini test eder. Burada koleksiyona eklenecek dokümanın(document) içindeki elemanlar integer, string ve dateTime olarak seçilmiştir.
- *TestNewDbCollectionDocumentFail;* TestNewDbCollectionDocument ile çok benzer. Ancak burada şartın sağlanmaması durumunda sistem fail olmalıdır.

- *TestNewDbCollectionCreate;* Veritabanına eklenen yeni koleksiyonun, oluşturulması durumunu test eder.
- *TestDbCollectionConnection;* Veritabanına ait bir koleksiyona bağlanabilme durumunu test eder.
- *TestDbCollectionTrueKeyValue;* Veritabanına ait bir koleksiyona eklenen bir verinin, anahtar/değer bilgilerinin doğru kayıt edilmesini test eder.
- *TestDbCollectionMoreKeyValue;* Veritabanına ait bir koleksiyona eklenen bir verinin, anahtar/değer sayısının koleksiyondakinden fazla olması durumunu test eder.
- *TestDbCollectionKeyValue;* Veritabanına ait bir koleksiyona eklenen bir verinin istenen biçimde kayıt edilmesi durumunu test eder.
- *TestDbCollectionLessKeyValue;* Veritabanına ait bir koleksiyona eklenen bir verinin, anahtar/değer sayısının koleksiyondakinden az olması durumunu test eder.
- *TestDbCollectionDocument;* Veritabanına ait bir koleksiyona eklenen bir verinin gerçekten kayıt edilip edilmediğini test eder.
- *TestNewDbCollectionCount;* Veritabanına eklenen koleksiyonun, birden fazla eklenmesi durumun; yani veritabanındaki koleksiyon sayısını test eder.

Testler için bir tane veritabanından koleksiyon seçilmesi, bir tane de yeni bir koleksiyon ismi girilmesi gerekmektedir. Bu durum ile ilgili kullanıcıya iki seçenek sunulmuştur; kullanıcı koleksiyon isimlerini ekranda uygun yerlerden girerek testleri başlatabilir; bunun için var olan koleksiyon üzerinde işlem yapabilmek için *generalListBox* isimli ListBox nesnesinde koleksiyonlarından seçer, yeni bir koleksiyon üzerinde işlem yapılabilmesi için de *txtBoxNewCollectionName* isimli TextBox nesnesine bilgi girişi yapar ve "Start Test" nesnesine tıklar veya "You do the Test" diyerek koleksiyonların sistem tarafından belirlenmesini sağlayarak testleri gerçekleştirir.

5.5. Sistem Testleri

Eğer bir veritabanı inceleniyor ise, bu veritabanını ve sunucusunu da test gerekir. Bu bilgi göz önünde bulundurularak, sistemin bir diğer testi olan System Tests geliştirilmiştir. Yapılan testlerde sonuçlar ile birlikte işlemlerin süresi de kullanıya bilgi olarak gösterilmiştir.

Bu bağlamda; menüde System Tests başlığına tıklandığında, *systemTests_click()* yardımı ile sistem "SystemTestsMenuPage.aspx" sayfasına yönlendirir. Bu sayfa içerisinde, başlığın altındaki testler ile ilgili genel bir bilgi verilmektedir.

Kullanıcı System Tests başlığı altındaki testlerden yapmak istediğinde, 2 tane seçenek ile karşılaşır. Ve bu 2 test "SystemTests.aspx" sayfasına yönlendirilerek gerçekleştirilir. Ayrıca bu testler ClassSystemTests sınıfı kullanılarak yapılır.

ClassSystemTests systemTests = new ClassSystemTests();

systemTests.aspx sayfası systemLabel isminde bir Label nesnesi içermektedir. Yapılan testlerin sonuçları bu systemLabel yardımı ile kullanıcıya gösterilmektedir. SystemTests.aspx'in kod kısmında, hangi testin yapılacağına dair karar verilerek ona uygun işlemler gerçekleştirilmektedir. Bunun için, her test için ayrıca tanımlanmış ve bir değer atanmış olan *Session["system"]* nesnesi kullanılır. *Session["system"]*, tüm testler SystemTests.aspx sayfasında gerçeklendiği için, kullanıcının hangi testi seçtiğine karar vermek için kullanılır. *Session["system"]* ile atanan değerler testleri açıklarken belirtilmiştir. Hangi testin gerçekleneceğine ise switch-case komutu yardımı ile karar verilmiştir.

5.5.1. Sunucu Bağlantı Testi

Bir veritabanına ulaşabilmek, onun üzerinde işlem yapabilmek için öncelikle doğru sunucuya, doğru bir şekilde bağlanmış olmak gerekir. Bu başlık altında, kullanıcının veritabanı sunucusuna bağlantı durumunu test edilerek, elde edilen sonuç kullanıcı ile paylaşılmıştır. Bu testler gerçeklerken daha önceden bahsedilmiş olan ClassSystemTests ve FindDatabasesCollections sınıflarından faydalanılmıştır.

Kullanıcı menü başlığı altında ilgili bölümü seçtiğinde, TestMyServerStatus_click() fonksiyonu ile "SystemTests.aspx" sayfasına yönlendirilir, sistem testleri için gerçekleşen tüm işlemler ve sonuçları bu sayfa ile bağlantılıdır. TestMyServerStatus_click() fonksiyonu içerisinde sayfalar arası veri aktarımı için kullanılan *Session["system"] = "testmyserverstatus"* güncellemesi yapılır. Bu işlemin yapılmasının sebebi, sayfalar arası geçişte, hangi testin gerçekleşeceğine kolaylıkla karar vermektir. "SystemTests.aspx" sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassSystemTests sınıfındaki *TestMyServerStatus()* fonksiyonundan faydalanılır.

TestMyServerStatus() fonksiyonu içerisinde; FindDatabasesCollections sınıfı şeklinde *finder* isminde bir değişken tanımlanır, ve bu değişken yardımı ile veritabanı sunucu bilgilerine ulaşılır. Bunun için FindDatabasesCollections sınıfında bulunan *getServer*() fonksiyonu kullanılır. Elde edilen sunucu bilgilerini MongoServer tipinde *server* isimli bir değişkene atanır;

MongoServer server = finder.getServer();

Bu işlemden sonra, C#'ın sağladığı, sunucunun durumu hakkında bir bilgiye ulaşabilmeyi sağlayan *.State()* fonksiyonu kullanılır, bu fonksiyon sunucu hakkında iki bilgi verir, eğer sunucu doğru bir şekilde bağlanmış ise "Connected", bağlanamamış ise "Disconnected".

MongoServerState sonuc = server.State;

Son olarak da, sunucunun durumu hakkında elde edilen bilgi, kullanıcıya gösterilmek üzere geri bilgi olarak gönderilir.

5.5.2. Veritabanı Kontrolü Testi

Veritabanı üzerinde işlem yapılacak ise, veritabanının çalışır durumda olduğundan emin olunması gerekmektedir. Bu başlık altında, veritabanının son durumunu test ederek, kullanıcıya hakkında bilgi verme işleminin nasıl gerçeklendiği anlatılmıştır. Test gerçeklenirken daha önceden bahsedilmiş olan ClassSystemTests sınıfından faydalanılmıştır.

Kullanıcı menü başlığı altında ilgili bölümü seçtiğinde, TestMyServerStatus_click() fonksiyonu ile "SystemTests.aspx" sayfasına yönlendirilir, sistem testleri için gerçekleşen tüm işlemler ve sonuçları bu sayfa ile bağlantılıdır. TestMyServerStatus_click() fonksiyonu içerisinde sayfalar arası veri aktarımı için kullanılan *Session["system"] = "testmydatabasestatus"* güncellemesi yapılır. Bu işlemin yapılmasının sebebi, sayfalar arası geçişte, hangi testin gerçekleşeceğine kolaylıkla karar vermektir. "SystemTests.aspx" sayfasında hangi testin yapılacağına karar verilir ve bu test için ClassSystemTests sınıfındaki *TestMyDatabaseStatus()* fonksiyonundan faydalanılır.

TestMyDatabaseStatus() fonksiyonu içerisinde; FindDatabasesCollections sınıfı şeklinde *finder* isminde bir değişken tanımlanır, ve bu değişken yardımı ile veritabanı sunucu bilgilerine ulaşılır. Bunun için FindDatabasesCollections sınıfında bulunan *getServer*() fonksiyonu kullanılır. Elde edilen sunucu bilgileri MongoServer tipinde server isimli değişkene aktarılırdık;

```
MongoServer server = finder.getServer();
```

Bu işlemden sonra, C#'ın sağladığı, sunucunun durumu hakkında bir bilgiye ulaşabilmeyi sağlayan *.State()* fonksiyonu kullanılır, bu fonksiyon sunucu hakkında iki bilgi verir, eğer sunucu doğru bir şekilde bağlanmış ise "Connected", bağlanamamış ise "Disconnected".

```
MongoServerState sonuc = server.State;
```

Burada önemli olan *sonuc* değişkeninin, yani sunucu durumunun, "Connected" olmasıdır. Aksi halde veritabanına bir bağlantı gerçekleştirilemez. Bu yüzden *sonuc* değişkeninin içerdiği bilginin "Connected" olduğundan emin olduktan sonra, veritabanının durumu kontrol edilir.

```
if (sonuc.ToString() == "Connected"){
    sonuc2 = server.GetDatabase(name).GetStats().Ok;
    return sonuc2.ToString(); }
```

Yukarıda yapılan işlemler şu şekildedir; daha önceden, FindDatabasesCollections sınıfında tanımlanmış ve kullanıcının tanımladığı veritabanı ismine ulaşmak için hazırlanmış olan

getDatabaseName() fonksiyonu yardımı ile veritabanı ismine ulaşılır ve *name* değişkenine aktarılır;

String name = finder.getDatabaseName();

Bunun yapılma sebebi veritabanına bağlanabilmek için, ismine ihtiyaç olmasıdır. *sonuc* değişkeninin "Connected" olduğundan emin olduktan sonra, C#'ın sağladığı, *GetDatabase()* ve *GetStats()* fonksiyonlarını kullanılır. Bu fonksiyonlardan GetDatabase(), veritabanına bağlanmayı sağlar ve *server.GetDatabase(name)* şeklinde kullanılmıştır. *GetStats()* ise veritabanı hakkındaki genel bilgileri içerir ve *server.GetDatabase(name).GetStats()* şeklinde kullanılmıştır. Burada önemli olan *Ok* komutur. Bu komut sonuç olarak True/False mantıksal değerlerini verir. Sonucun True olması durumunda veritabanı ile bağlantı sağlanmış, False olması durumunda ise sağlanamamış olur.

Son olarak tüm işlemler tamamlandığında kullanıcıya bilgileri gösterebilmek için bulunan sonuc *sonuc2* değişkeni kullanılarak geri gönderilir.

5.6. İndeks Testi

Dağıtık Veritabanlarının en önemli özelliklerinin tablolar-koleksiyonlar arası ilişki kurulamamasıdır. Yani SQL veritabanlarında kullanılan JOIN, INNER JOIN gibi tablolar arası ilişki kurmayı sağlayan komutlar, NoSQL sistemlerde kullanılmayan komutlardır. Dolayısıyla NoSQL sistemler ilişkilendirmeye alternatif olarak, veritabanı içerisinde indeksleme yöntemini kullanmaya başlamışlardır.

MongoDB, NoSQL veritabanının en önemli özelliklerinden biri, anlaşılabilir ve pratik bir şekilde gerçekleştirilen "Indexing". Dolayısıyla üzerinde test yapılan veritabanının, indekslerinin de test edilmesi gerekir. Geliştirilen sistemde bu durum göz önünde bulundurulmuş ve, MongoDB'nin indekslerini inceleyecek bir test tasarlanmıştır. Aşağıdaki maddelerde testin amacı açıklanmıştır;

- Kullanıcının seçeceği herhangi bir koleksiyonda bulunan indekslerin sayısını belirleyerek, gereksiz ve kullanılmayan indekslerden kurtulması.
- Seçilen koleksiyonun içerisinde yapılan sorgularda, indekslenmemesine rağmen çok fazla kullanılan anahtar bilgisini bularak, bu anahtarın indekslenmesi için kullanıcıya öneride bulunmak.

Yukarıda belirtilmiş iki maddenin de ortak noktası performans artışı sağlayabilmektir. Hatta bu doğrultuda, indekslenmesi gereken anahtarlar için öneride bulunduktan sonra; seçilen koleksiyon içerisinde önce o alanların indekslenmemiş şekliyle, daha sonra indekslenmiş şekliyle koleksiyonun tüm verileri çekilerek, aradaki performans artışını görmelerini sağlanmaya çalışılmıştır.

Testi yapmak için izlenen yol şu şekildedir; menüden bu testin yapılması için gerekli kısım seçildiği zaman,

checkIndexStatus_click()

aktifleşir ve *"IndexTestsMainPage.aspx"* sayfasına yönlendirir. Bu sayfanın içerisinde indexLabel ismi verilen bir Label, indexListBox ismi verilmiş bir ListBox ve buttonIndexTest ismi verilen bir Button nesnesi bulunur. IndexTestsMainPage.aspx içerisinde belirli işlemler yapılarak, test gerçekleştirilir ve ekrana sonuçlar listelenir.

Veritabanının indeksleri için yapılan bu testte, daha önceden oluşturulmuş IndexTestsClass sınıfından faydalanılır. Sınıflar başlığında, bu sınıfın zaten indeks testleri için geliştirdiğinden ve sınıf içerisindeki fonksiyonların yaptıklarından detaylıca bahsedilmiştir. Burada sınıfın fonksiyonları kullanılarak sonuca nasıl ulaşıldığı detaylıca anlatılmıştır.

Sayfanın ilk açılışında, kullanıcının hangi koleksiyonunun hangi indekslere sahip olduğunu görmesi için, IndexTestsClass sınıfın *gettingYourIndexes()* fonksiyonu kullanılarak, tüm indeksler elde edilmiş ve indexLabel nesnesi içerisinde ekranda kullanıcıya gösterilmiştir.(Bu işlemin detaylı anlatılmamasının sebebi, ilerleyen satırlarda aynı işlemin tekrarlanıyor oluşudur)

Kullanıcı, üzerinde işlem yapılmasını istediği koleksiyonu seçerek "For Index Testing" nesnesine tıkladıktan sonra gerçekleşen işlemler şu şekildedir; *indexListBox* içerisinden seçilen koleksiyon, *eklenmekIstenen* isimli karakter değişkenine atanır. ListBox içerisine listelenen koleksiyonlar "Veritabanı_ismi.Koleksiyon_ismi" şeklinde olduğundan ve bu durum koleksiyon üzerinde işlem yapmayı engellediğinden, eklenmekIstenen değişkeni Substring() fonksiyonu yardımı ile kullanıma uygun hale getirilir.

```
eklenmekIstenen = eklenmekIstenen.Substring(eklenmekIstenen.IndexOf(".") + 1);
```

Bu işlemin ardından, üzerinde işlem yapılmak istenen koleksiyonun C# içerisinde aktifleştirilir. Bunun için FindDatabasesCollections sınıfının getCollection() fonksiyonu kullanılarak, *collection* isimli MongoCollection tipinde bir değişkende, kullanıcının üzerinde test yapılmasını istediği(*eklenmekIstenen)* koleksiyonun bilgisi tutulur;

```
FindDatabasesCollections finder=new
FindDatabasesCollections(Formlar.connectionString);
MongoCollection collection = finder.getCollection(eklenmekIstenen);
```

Testin gerçekleşebilmesi için öncelikle, üzerinde işlem yapılmak istenen koleksiyonun(collection) hangi indekslere sahip olduğunu belirlemek gerekir. Bunun için IndexTestsClass sınıfının *gettingYourIndexes()* fonksiyonu kullanılarak, koleksiyonun indeksleri belirlenmiş ve *collectionIndexes* ismi verilen, daha önceden karakter listesi şeklinde tanımlanmış değişkene atanmıştır.

collectionIndexes = getThem.gettingYourIndexes(collection);

Ardından öneride bulunmak için incelenmek üzere kullanılacak olan, koleksiyonun dokümanları içerisinde en çok sorgulanmış anahtar/değer çiftlerine ulaşılmıştır. Bunu yapmak için IndexTestsClass sınıfının *analyzeMyIndexesAndSearchedDocuments()* fonksiyonu kullanılmış ve elde edilen değerler daha önceden tanımlanmış olan *analyzedResults* karakter listesine atanmıştır.
analyzedResults = getThem.analyzeMyIndexesAndSearchedDocuments(collectionIndexes, collection);

analyzedResults değişkeninin bilgi içermesi durumunda, koleksiyonun(collection) indekslerine ve en çok aranan anahtar bilgilerine ulaştıktan sonra, indeks önerisinde bulunabilmek için IndexTestsClass sınıfının *writeMyAnalyzedResultAndOfferIndex()* fonksiyonu kullanılmış ve indeks önerileri ile birlikte, performansın değişimi ile ilgili örnekler gerçeklenmiştir. Bu işlem sonucunda elde edilen bilgiler, *writeResult* ismi verilen karakter değişkene atanmıştır.
if (analyzedResults.Any()) writeResult =
getThem.writeMyAnalyzedResultAndOfferIndex(analyzedResults, collection);

Ve son olarak elde edilen tüm veriler, uygun bir biçimde kullanıcı ekranında gösterilmiştir.

6. SONUÇ VE GELECEK ÇALIŞMALAR

Geliştirilen testlerin sisteme olan etkisini görebilmek için, örnek bir veritabanı yardımı ile tüm testler gerçeklenmiş ve öneriler doğrultusunda hareket edilmiştir. Burada kullanılarak örnek veritabanı İlişkisel Veritabanlarında da bulunan "Northwind" veritabanıdır. Ancak sistemi test etmek için kullanılan veritabanı, Dağıtık Sistemlerde kullanıma uygun hale getirilmiş olanıdır.

Geliştirilen projenin sistem üzerindeki etkisini test etmek için 2 farklı test ortamı oluşturulmuştur, bunlar; tek bir sunucu üzerinde replikasyon veya Sharding işlemler yapılmamış sistemi test yapma, 3 sunucu kullanılarak replikasyon yapmış sistemi test etme ve Sharding yapılmış sistemi test etme'dir. Geliştirme sonrası testler yapılarak performans değişimleri ile ilgili ulaşılan sonuçlar bir tablo ve grafik halinde gösterilmiştir.

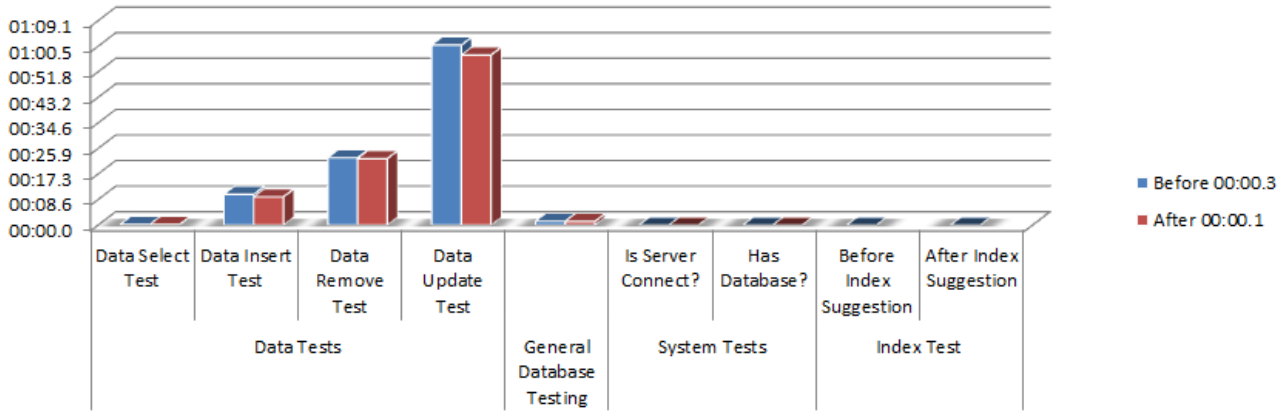
Her ortamda aşağıdaki işlemler yapılarak testler gerçekleştirilmiştir;

- Northwind veritabanına ait her koleksiyonda ekleme, silme ve güncelleme işlemleri için 1000 adet veri kullanılmıştır.
- Testler gerçekleştirilip (Before) sonuçlara ve önerilere ulaşıldıktan sonra, sistemin Index Test sonucundaki önerisi olan indekslere göre –varsa- yeni indeksler eklenmiş ve ardından testler tekrar yapılmıştır. Ve farklı tablolarda listelemeler yapılmıştır.
- Bazı test durumlarında yapılan öneriyle oluşan değişiklikler sistemdeki test süresini etkilememektedir. Bunlar, bağlantıyı kontrol eden "System Tests" ve replikasyon yapılmış sistemler için geliştirilmiş "Replikasyon Testleri" testleridir.

İlk olarak tek bir sunucu üzerinde replikasyon veya Sharding işlemler yapılmamış bir sistem test edilmiş ve sonuçları Tablo 6.1 içine listelenmiştir. Burada görüldüğü üzere test sonuçlarında "Replikasyon Testleri" bilgileri bulunmamaktadır. Bunun sebebi replikasyon yapılmamış bir sistemde test sonuçlarınında bir anlam ifade etmeyecek oluşudur. Aşağıdaki tablo ve grafik üzerinde test yapılan veritabanının, geliştirilen sistemin Index önerisinden önce ve sonra nasıl değiştiğini göstermektedir. Tablo/grafikte görüleceği üzere milisaniyelik bile olsa sistem performansında hızlanma yaşanmıştır.

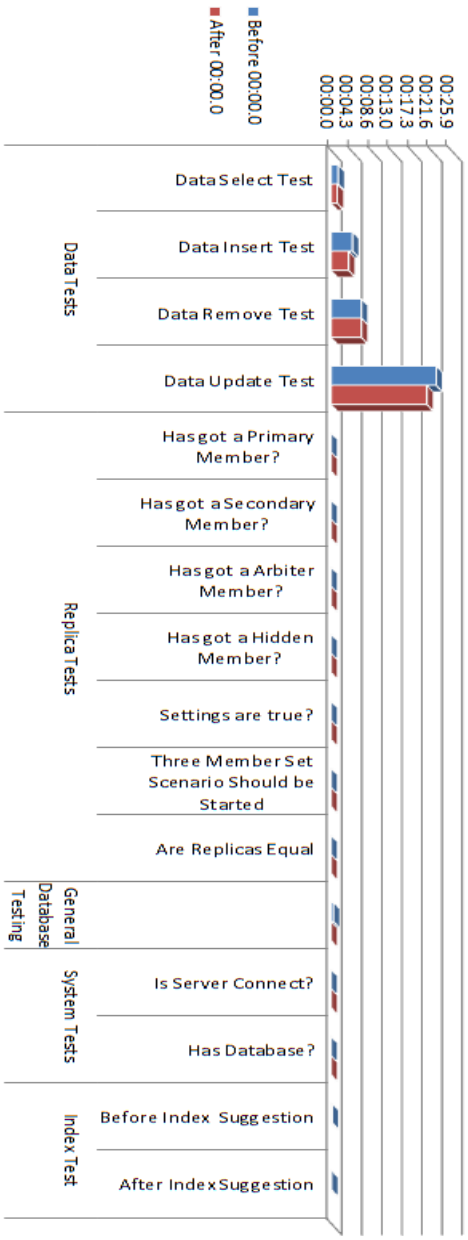
		Before	After
Structure Test		00:00:0 0.28400 51	00:00:0 0.12207 58
Veri Testleri	*Data Select Test*	00:00 :00.2 1200 54	00:00 :00.2 0113 30
	Data Insert Test	00:00 :10.4 3637 76	00:00 :09.5 6523 37
	Data Remove Test	00:00: 22.765 1420	00:00: 22.485 4068
	Data Update Test	00:01 :00.8 7287 56	00:00 :57.5 2123 89
General Database Testing		00:0 0:01 .318 775 2	00:0 0:01 .247 522 5
System Tests	*Is Server Connect?*	00:00: 00.000 8006	00:00: 00
	Has Database?	00:00:0 0.01599 86	00:00:0 0.01400 89
Index Test	*Before Index Suggestion*	00:00:00 .000993 3	
	After Index Suggestion	00:00:00	

Tablo 6.1 Tek Sunuculu Sonuç Tablosu



Şekil 6.1 Tek Sunuculu Sonuç Grafiği

Bir diğer test ortamı 3 sunucu kullanılarak replikasyon yapmış sistemi test etmektir. Burada test işlemleri üç aşamada gerçekleşmiştir. İlk önce sistem 1000 tane veri içerecek şekilde test edilmiş ve sonuçlar Tablo 6.2 ve Şekil 6.2 ile gösterilmiştir. Ardından sistem 10.000 tane veri içerecek şekilde test edilmiş ve sonuçlar Tablo 6.3 ve Şekil 6.3 ile gösterilmiştir. Son olarak sistem 1.000.000 veri bulunduracak şekilde test edilmiş ve sonuçlar Tablo 6.4 ve Şekil 6.4 ile gösterilmiştir. Geliştirilmiş sistemin önerisi ile birlikte gerçekleştirilen değişiklikler sayesinde performanstaki artış gözle görülür şekilde değişmektedir. Aşağıdaki tablo ve grafik üzerinde test yapılan veritabanının, geliştirilen sistemin Index önerisinden önce ve sonra nasıl değiştiğini göstermektedir.



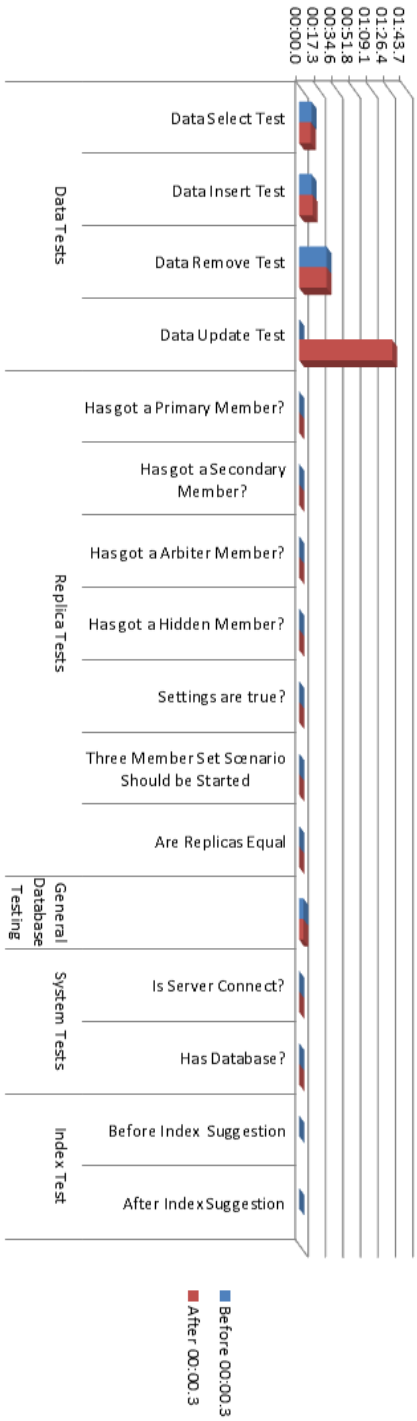
Şekil 6.2 Replikasyon Yapılmış Sistem Sonuç Grafiği

		Before	After
Structure Test		00:00:00.0210098	00:00:00.0190130
Veri Testleri	*Data Select Test*	00:00:01.5276382	00:00:01.3098770
	Data Insert Test	00:00:04.6377013	00:00:03.7691319
	Data Remove Test	00:00:06.5277362	00:00:06.5106167
	Data Update Test	00:00:22.6328281	00:00:20.6328302
Replikasyon Testleri	*Has got a Primary Member?*	00:00:00.0001420	00:00:00.0001098
	Has got a Secondary Member?	00:00:00	00:00:00.0001098
	Has got a Arbiter Member?	00:00:00	00:00:00.0001098
	Has got a Hidden Member?	00:00:00	00:00:00.0001098
	Settings are true?	00:00:00	00:00:00.0011028
	Three Member Set Scenario Should be Started	00:00:00 0	00:00:00.0010210
	Are Replicas Equal	00:00:00.0010300	00:00:00.0010100
General Database Testing		00:00:00.5203472	00:00:01/1439870
System Tests	*Is Server Connect?*	00:00:00.0001023	00:00:00.0000897

	Has Database?	00:00:00.0001028	00:00:00.000925
Index Test	*Before Index Suggestion*	00:00:00.2331556	
	After Index Suggestion	00:00:00.10270130	

Tablo 6.2 Replikasyon Yapılmış Sistem Sonuç Tablosu

10.000 veri bulunduran sistem için grafik ve tablo gösterilimi aşağıdaki gibidir;



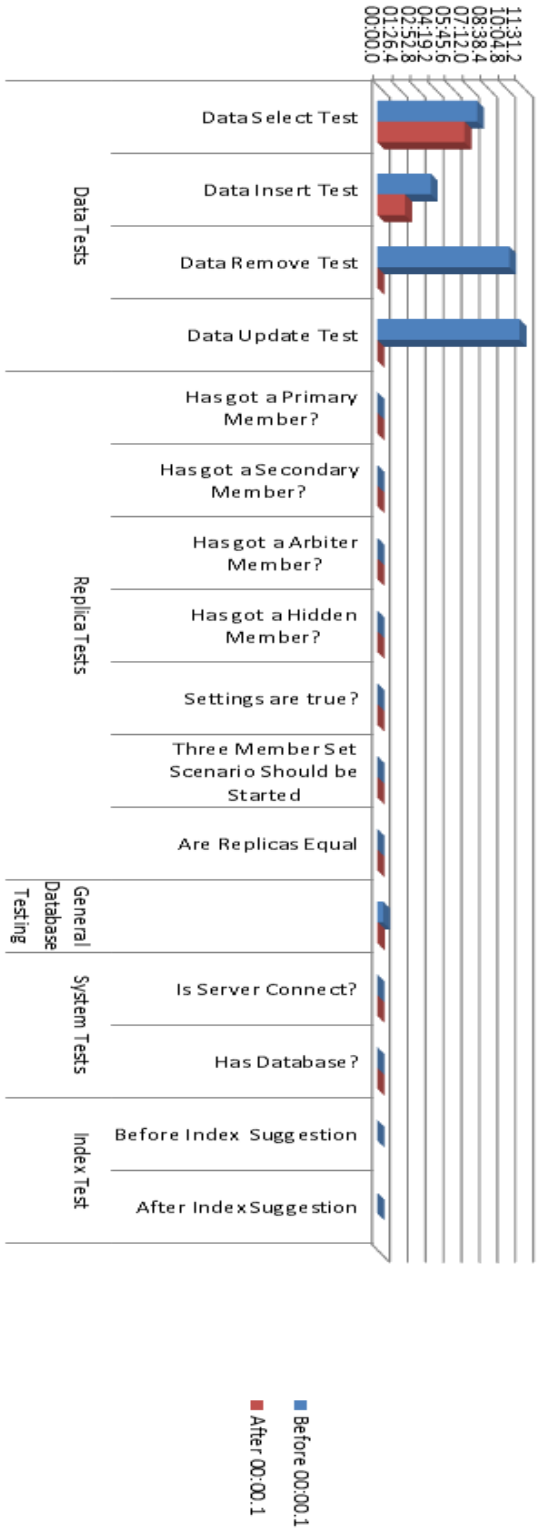
Şekil 6.3 10.000 Verili Sonuç Grafiği

		Before	After
Structure Test		00:00:00.2724777	00:00:00.2705169
Veri Testleri	*Data Select Test*	00:00:12.3255320	00:00:11.4895407
	Data Insert Test	00:00:12.3255320	00:00:13.2412534
	Data Remove Test	00:00:27.0673402	00:00:27.0784538
	Data Update Test	00:01:33.0036394	00:01:31.9653437
Replikasyon Testleri	*Has got a Primary Member?*	00:00:00.0019959	00:00:00.0019960
	Has got a Secondary Member ?	00:00:00.0009987	00:00:00
	Has got a Arbiter Member ?	00:00:00	00:00:00
	Has got a Hidden Member ?	00:00:00	00:00:00
	Settings are true?	00:00:00	00:00:00
	Member Set Scenario Should b	00:00:00	00:00:00
	Are Replicas Equal	00:00:00.0010300	00:00:00.0010100
General Database Testing		00:00:04.1332927	00:00:04.0773187
System Tests	*Is Server Connect ?*	00:00:00.0010027	00:00:00.0009928

	Has Database?	00:00:00.0001028	00:00:00.0009633
Index Test	*Before Index Suggestion*	00:00:00.0009982	
	After Index Suggestion	00:00:00	

Tablo 6.3 10.000 Verili Sonuç Tablosu

100.000 veri bulunduran sistem için grafik ve tablo gösterilimi aşağıdaki gibidir;



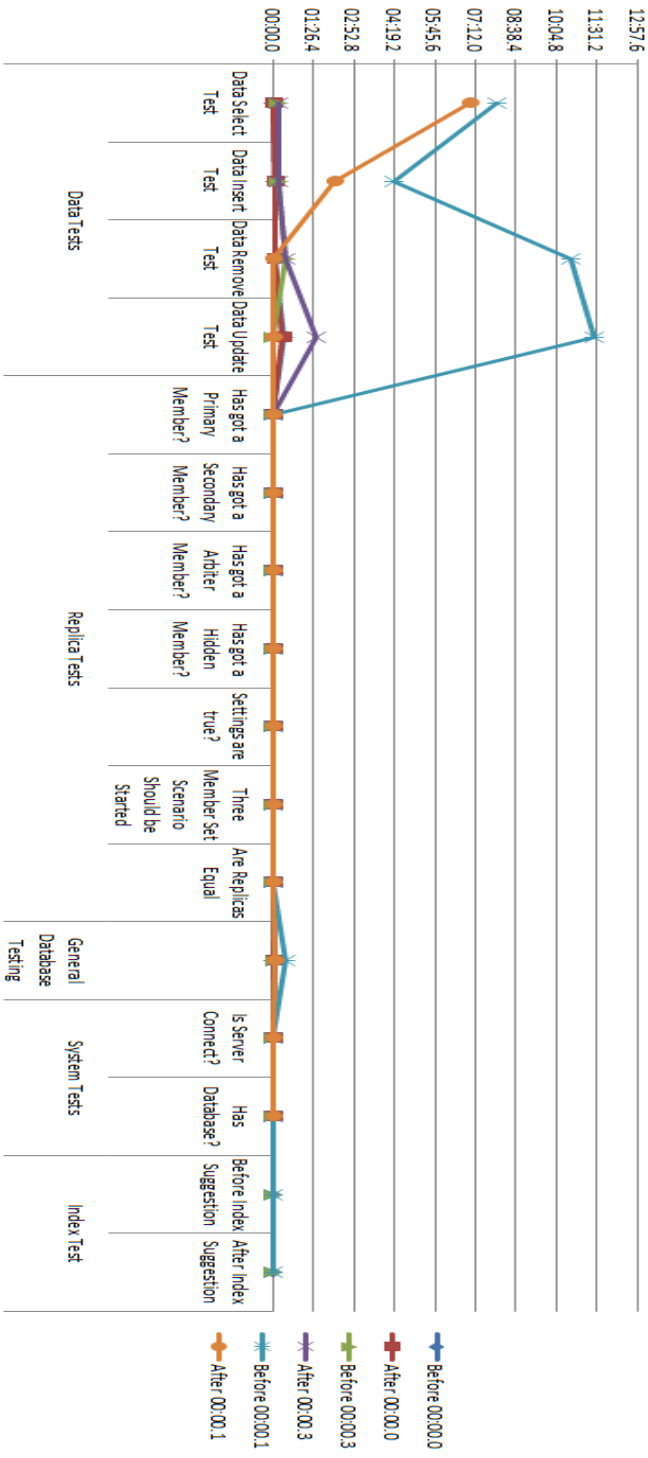
Şekil 6.4 100.000 Verili Sonuç Grafiği

		Before	After
Structure Test		00:00:00.0618780	00:00:00.0705169
Veri Testleri	*Data Select Test*	00:08:00.0049879	00:07:00.4011879
	Data Insert Test	00:04:17.5239122	00:02:13.2459534
	Data Remove Test	00:10:35.0626660	00:08:271787638
	Data Update Test	00:11:24.7909499	00:11:3000600007
Replikasyon Testleri	*Has got a Primary Member?*	00:00:00.0019951	00:00:00.0010960
	Has got a Secondary Member?	00:00:00	00:00:00
	Has got a Arbiter Member?	00:00:00	00:00:00
	Has got a Hidden Member?	00:00:00	00:00:00
	Settings are true?	00:00:00	00:00:00
	Member Set Scenario Should be	00:00:00	00:00:00
	Are Replicas Equal	00:00:00.0011200	00:00:00.0010100
General Database Testing		00:00:28.0318705	00:00:04.0773187
System Tests	*Is Server Connect?*	00:00:00.0009969	00:00:00.0009928

	Has Database?	00:00:00.0159593	00:00:00.0009633
Index Test	*Before Index Suggestion*	00:00:00.0009982	
	After Index Suggestion	00:00:00	

Tablo 6.4 100.000 Verili Sonuç Tablosu

Farklı veri sayısı ile test işlemleri gerçekleştirildiğinde, MongoDB Tester kullanımın ile yapılan öneri ile veritabanında belirli durumlarda işlem performansında iyileşmeler olduğu görülmüştür. Veri sayısı arttıkça sistemin işlem süresi artsada, öneride bulunulan yeni indeksler sisteme tanımlandığında işlem hızının arttığı saptanmıştır. Tablo 6.2, Tablo 6.3 ve Tablo 6.4 içerisinde bulunan değerlerin genel değişimini gösterek grafik aşağıdaki gibidir;



Şekil 6.5 Genel Grafik

7. REFERANSLAR

[1] Dragomir, G. & Ignat, I. (2006).Distributed Database Environment Testing. IEEE International Conference on Automation, Quality and Testing, Robotics Vol.2. (pp. 90-95).

[2] Soto, A. (2012). NoSQLUnit. Web site: www.lordofthejars.com.

[3] Ibrahim, H., Alwan, A. A., Udzir, N. I. (2007). Checking Integrity Constraints with Various Types of Integrity Tests for Distributed Databases. Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (pp. 1-2).

[4] Qian, H. & Zheng, C. (2009). A Embedded Software Testing Process Model. Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on. (pp. 1-5).

[5] Kapur, P.K. & Singh, G. & Sachdeva, N. & Tickoo, A. (2014). Measuring Software Testing Efficiency Using Two-Way Assessment Technique. Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 2014 3rd International Conference on. (pp. 1-6).

[6] Zhang, B. & Shen, X. (2011). The Effectiveness of Real-time Embedded Software Testing. Reliability, Maintainability and Safety (ICRMS), 2011 9th International Conference on. (pp. 661-664).

[7] Sekar, R. (2005). An Efficient Black-box Technique for DefeatingWeb Application Attacks.

[8] Marin, M. (2014). A data-agnostic approach to automatic testing of multi-dimensional databases. 2014 IEEE International Conference on Software Testing, Verification, and Validation. (pp. 133-142).

[9] Pan, K. & Wu, X. (2013). Automatic Test Generation for Mutation Testing on Database Applications. Automation of Software Test (AST), 2013 8th International Workshop on. (pp. 111-117).

[10] Khalek, S. A. & Khurshid, S. (2011). Systematic Testing of Database Engines Using a Relational Constraint Solver. 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. (pp. 50-59).

[11] Jingsong, Z. (2014). Research and Application of Testing Technology of the Cloud Computing Database. 2014 IEEE Workshop on Electronics, Computer and Applications. (pp. 699-702).

[12] Deng, Y. & Chays, D. (2005). Testing Database Transactions with AGENDA. Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on. (pp. 78-87).

[13] [Online] CUnit. http://cunit.sourceforge.net/.

[14] [Online] JUnit. http://junit.org/.

[15] [Online] Jakarta Cactus. http://attic.apache.org/projects/jakarta-cactus.html.

[16] [Online] CppUnit. http://cppunit.sourceforge.net/doc/cvs/.

[17] [Online] Google Test. https://code.google.com/p/googletest/.

[18] Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise?. Computer Technology News. 43(2). (pp. 12-14).

[19] Lomotey, R.K. & Deters, R. (2014). Terms Mining in Document-Based NoSQL: Response to Unstructured Data. Big Data (BigData Congress), 2014 IEEE International Congress on. (pp. 661-668).

[20] Konstantinou, I. & Angelou, E. & Boumpouka, C. & Tsoumakos, D. & Koziris, N. (2011). On the Elasticity of NoSQL Databases over Cloud Management Platforms. Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on. (pp. 496-497).

[21] Cai, L. & Huang, S. & Chen, L. & Zheng, Y. (2013). Performance Analysis and Testing of HBase Based on Its Architecture. Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on. (pp. 353-358).

[22] Haque, W. & Stokes, P.R. (2005).Simulation of a Complex Distributed Real-Time Database System. Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on. (pp. 589-593).

[23] Dede, E. & Govindaraju, M. & Gunter, D. & Canon, R.S. & Ramakrishnan, L. (2013). Performance Evaluation of a MongoDB and Hadoop Platform for Scientific Data Analysis. Proceedings of the 4th ACM workshop on Scientific cloud computing. (pp. 13-20).

[24] Dragomir, G. & Ignat, I. (2006). Distributed Database Environment Testing. Automation, Quality and Testing, Robotics, 2006 IEEE International Conference on. (pp. 90-95).

[25] Codecentric, Agile Software Factory. Test Automation for NoSQL Databases. NoSQL Unit & Travis CI.

[26] Nidhra, S. & Dondeti, J. (2012). Black Box And White Box Testing Techniques –A Literature Review. International Journal of Embedded Systems and Applications (IJESA). Vol2.

[27] Khan, M.E. (2011). Different Approaches to White Box Testing Technique for Finding Errors. International Journal of Software Engineering and Its Applications. 5(3).

[28] Daou, B. & Haraty, R.A. & Mansour, N. (2001). Regression Testing of Database Applications. Proceedings of the 2001 ACM symposium on Applied computing. (pp. 285-289).

[29] Khalek, S.A. & Khurshid, S. (2011). Systematic Testing of Database Engines Using a Relational Constraint Solver. 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. (pp. 50-59).

[30] MongoDB Documentation, September 16, 2014.

[31] Seguin, K. (2014). The Little MongoDB Book.

[32] Boicea, A. & Radulescu, F. & Agapin, L.I. (2012). MongoDB vs Oracle - database comparison. 2012 Third International Conference on Emerging Intelligent Data and Web Technologies. (pp. 330-335).

[33] Liu, Y. & Wang, Y. & Jin, Y. (2012). Research on The Improvement of MongoDB Auto-Sharding in Cloud Environment. The 7th International Conference on Computer Science & Education (ICCSE 2012). (pp. 881-884).

[34] Zhao, G. & Huang, W. & Liang, S. & Tang, Y. (2013). Modeling MongoDB with Relational Model. 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies. (pp. 115-121).

[35] Murugesan, P. & Ray, I. (2014). Audit Log Management in MongoDB. 2014 IEEE 10th World Congress on Services. (pp. 53-57).

[36] Prabagaren, G. (2014). Systematic Approach for validating Java-MongoDB Schema. ICICES2014 - S.A.Engineering College, Chennai, Tamil Nadu, India. (pp. 1-4).

[37] Kanade, A. & Gopal, A. & Kanade, S. (2014). A Study of Normalization and Embedding in MongoDB. Advance Computing Conference (IACC), 2014 IEEE International. (pp. 416-421).

[38] Wassan, J. T. (2015). Discovering Big Data Modelling for Educational World. Procedia - Social and Behavioral Sciences. Volume 176. (pp. 642-649).

[39] Abramova, V. & Bernardino, J. (2013). NoSQL Databases: MongoDB vs Cassandra. Proceedings of the International C* Conference on Computer Science and Software Engineering. (pp. 14-22).

[40] Gardikiotis, S. K. & Malevris, N. (2006). Program Analysis and Testing of Database Applications. Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06) (pp. 368-373).

[41] Reza, H. & Zarns, K. (2011). Testing Relational Database Using SqlLint. 2011 Eighth International Conference on Information Technology: New Generations (pp. 608 - 613).

[42] General Dynamics. (2014). Current Data Security Issues of NoSQL Databases.

[43] [Online] Couchbase Survey Shows Accelerated Adoption of NoSQL in 2012, "http://www.couchbase.com/press-releases/couchbase-survey-shows-accelerated-adoption-nosql-2012".

[44] Zahid, A., Masood, R., & Shibli, M. A. (2014). Security of Sharded NoSQL Databases:A Comparative Analysis. 2014 Conference on Information Assurance and Cyber Security (CIACS) (pp. 1-8).

[45] Qi, M. (2011). Digital Forensics and NOSQL Databases. 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD) (pp. 734-739).

[46] Atzeni, P., Bugiotti, F. & Rossi, L. (2013). Uniform access to NoSQL systems. InformationSystems, 43(2014) 117–133.

[47] Tudorica, G. B. & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. 10th Roedunet International Conference (RoEduNet) (pp. 1-5).

[48] Chodorow, K. (2013). MongoDB The Definitive Guide. http://www.gocit.vn/files/MongoDB-www.gocit.vn.pdf.

[49] Liu, Y., Wang, Y. & Jin, Y. (2012). Research on the improvement of MongoDB Auto-Sharding in cloud environment. 7th International Conference on Computer Science & Education (ICCSE) (pp. 851-854).

[50] Chen, Z., Yang, S., Zhao, H. & Yin, H. (2012). An Objective Function for Dividing Class Family in NoSQL Database. International Conference on Computer Science & Service System (CSSS) (pp. 2091 - 2094).

[51] Han, J., Song, M. & Song, J. (2011). A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing. IEEE/ACIS 10th International Conference on Computer and Information Science (ICIS) (pp. 351 - 355).

[52] Wang, G. (2012). The NoSQL Principles and Basic Application of Cassandra Model. International Conference on Computer Science & Service System (CSSS) (pp. 1332 - 1335).

[53] Benefico, S., Gjeci, E., Gomarasca, R.G., Lever, E., Lombardo, S., Ardagna, D. & Di Nitto, E. (2012). Evaluation of the CAP Properties on Amazon SimpleDB and Windows Azure Table Storage. 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) (pp. 430 - 435).

[54] Gilbert, S. & Lynch, N. A. (2012). Perspectives on the CAP Theorem. Computer Journal. From http://dspace.mit.edu/handle/1721.1/79112.

[55] Li, Y. & Manoharan, S. (2013). A Performance Comparison of SQL and NoSQL Databases. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) (pp.15-19).

[56] Naheman, W. & Wei, J. (2013). Review of NoSQL Databases and Performance Testing on HBase. Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC) (pp. 2304 - 2309).

[57] Han, J., E, H., Le, G. & Du, J. (2011). Survey on NoSQL Database. 6th International Conference on Pervasive Computing and Applications (ICPCA) (pp. 363 - 366).

[58] Brewer, E. (2012). CAP twelve years later: How the "rules" have changed. Compuer, 45(2), 23 – 29.

[59] Seguin, K. The Little MongoDB Book. Web site: https://github.com/karlseguin/the-little-mongodb-book.

[60] MongoDB Documentation Project. (2015). Indexes and MongoDB.