©2022 Published in 10th International Symposium on Innovative Technologies in Engineering and Science 21-23 October 2022 (ISITES2022 Bursa - Turkey)
https://doi.org/10.33793/acperpro.05.03.3492

Uzamsal Verilerde İndeks Kullanımının Sorgu Verimliliği Üzerinde Etkileri

*[1]Enes Özdeniz, [2]Mustafa Utku Kalay
*[1]Mühendislik ve Doğa Bilimleri Fakültesi, Bilgisayar Mühendisliği Bölümü, Konya Teknik Üniversitesi, Türkiye
[2]Elektrik Elektronik Fakültesi, Bilgisayar Mühendisliği Bölümü, Yıldız Teknik Üniversitesi, Türkiye

Abstract

Indexing spatial data is of great importance in terms of accessibility and query performance. In this study, query performances on indexed and non-indexed tables and bulk loading times of different sizes of data into index structures were measured by performing some query experiments on a spatial database created on the Türkiye map. In this way, it is aimed to emphasize the importance of choosing the right index type and efficient index usage on spatial data.

Key words: Spatial Indexing, Query Performance, Bulk Loading, Geohash

Özet

Uzamsal verilerin indekslenmesi, bu verilere erişilebilirlik ve sorgu performansı açısından büyük önem taşır. Bu çalışmada Türkiye haritası üzerinde oluşturulmuş uzamsal bir veri tabanı üzerinde bazı sorgu deneyleri yapılarak, indeks kullanılan ve kullanılmayan tablolar üzerindeki sorgu performansları, farklı tip indekslerin performans karşılaştırmaları ve farklı boyutlardaki verilerin indeks yapılarına toplu yüklenme süreleri ölçülmüştür. Bu sayede uzamsal veriler üzerinde doğru indeks türünün seçilmesinin ve indeks kullanımının önemine vurgu yapılmak amaçlanmıştır.

Anahtar kelimeler: Uzamsal İndeksleme, Sorgu Performansı, Toplu Yükleme, Geohash

1. Giriş

Son yıllarda bilgi teknolojilerinin büyük gelişimi ve bilgi depolamanın kolaylığının bir getirisi olarak tüm dünya üzerinde büyük bir veri artışı yaşanmaktadır. Her türlü veriyi kapsayan bu astronomik artış, elbette uzamsal veriler için de söz konusudur. Örneğin, Google Maps uygulaması uydu görüntüleri ve kendi araçları ile topladığı harita verilerinin yanı sıra kullanıcılarının katkıları ile de veriler topluyor. Google Maps'in 2019 yılına ait verilerine göre insanlar her gün bu uygulamaya yaklaşık 20 milyon veri ekliyorlar [1]. Bu sadece insanlar tarafından eklenen saniyede 230 adetten fazla veri anlamına geliyor. Verilerin bu denli artışı; istediğimiz zaman onlara hızlıca erişebilmemiz adına, onları mümkün olan en doğru düzende saklama ihtiyacını da beraberinde getirir.

Tıpkı içerisinde yüz binlerce kitap barındıran bir kütüphanede, kütüphane yetkilisinin istenilen kitabı ivedilikle bulabilmesi için kitapların türlerine ve isimlerine göre bir düzende tutulması gibi verileri de istediğimizde en kolay ulaşabileceğimiz şekilde düzenlemeliyiz.

*İlgili yazar: Adres: Mühendislik ve Doğa Bilimleri Fakültesi, Bilgisayar Mühendisliği Bölümü, Konya Teknik Üniversitesi, TÜRKİYE. E-posta adresi: eozdeniz@ktun.edu.tr, Telefon: +905375060806

Verilere daha kolay erişimi sağlayabilmek için onları indeksleriz. Bu yöntem ile onları budanabilen arama ağaçlarının içlerine koymuş oluruz. Bu sayede tıpkı kütüphanedeki gibi aradığımız kitap 'tarih' türünde ise, diğer türlere hiç göz atmadan sadece bu türün bulunduğu kısımda aramayı devam ettiririz. Bu metodu kullanarak kütüphanede bu türün dışında kalan kitapları budamış oluruz. İndeksleme, verilerin daha kolay taranması ve istenen sonuçların çok daha az maliyetle elde edilmesi açısından büyük önem taşımaktadır [2]. İndeks kullanılmadığı takdirde, uzamsal veriler üzerinde yapılan sorgularda verilerin sıralı tarama (sequential scan) mantığı ile taranması, sorgu verimini son derece düşüren bir durum ortaya çıkarır.

Basit bir mantıkla, milyonlarca sayıyı içeren ve üzerinde indeks kullanılmayan bir veri tabanında aradığımız sayıyı bulmak için yaptığımız sorgu esnasında, sıralı tarama metodundan dolayı defalarca kez disk erişimi yapmak zorunda kalacağız. Fakat bu sayılar en temel veri yapılarından birisi olan B-Tree'de indekslenmiş halde olsaydı, birkaç disk erişimi ile bu durumu kurtarabilirdik.

İndeksler, büyük veri kümeleri için uzamsal bir veri tabanı kullanmayı mümkün kılar. Bu veri türleri, bir nesnenin veya noktanın uzaydaki konumunu temsil eder. Uzamsal verilerin gün geçtikçe büyüyen miktarı, sorgu performanslarında iyileştirmeler yapılması gereksinimi doğurmaktadır. Veri tabanlarındaki uzamsal verilere uygulanan sorguların maliyeti, uzamsal verilerin indekslenmesi için bir ihtiyaç ortaya çıkarmıştır.

Sayısal verileri ardışık olarak sıralayabiliriz. Çünkü bu veri türleri sayı doğrusu denen tek bir çizgi üzerinde konumlanmıştır. Ancak diğer veri türlerinden farklı olarak, uzamsal veriler bir uzayın içerisinde bulunurlar. Bundan dolayı uzamsal verileri sıralamak için matematik alanında kullanılan bazı eğrileri kullanmak gibi farklı yöntemler kullanılmaktadır. Bu da uzamsal verilerin bir düzene sokulmasını daha da karmaşık bir hale getirmektedir. Bu sebepten dolayı en verimli veri yapılarından biri olarak görülen ve birçok veri tabanı indeks yapısının temelinde yer alan B-Tree'ler uzamsal verileri üzerinde 'direkt' olarak kullanılamazlar. B-Tree'ler farklı sorgu türlerinde yüksek alan verimliliği ve hız açısından avantajlar sağladığı için veri tabanı sistemleri için oldukça popüler bir veri yapısıdır [3]. Bu nedenle uzamsal veriler için ortaya atılan veri yapıları fikirleri genel olarak B-Tree'nin dengeli yapısını yakalamaya çalışırlar ve bundan ötürü bu veri yapısının mimarisine benzeme eğilimindedirler.

Kabul edilmiş bir sıralama kriteri bulunmayan uzamsal veriler aynı zamanda, farklı boyutlardaki nesneleri temsil edebileceğinden her veri formunda optimum performans gösteren bir düzen de mevcut değildir. Diğer bir neden ise; ilk başta bahsettiğimiz gibi, uzamsal verilerin temsil ettiği nesnelerin geometrisi karmaşıktır. Bu nedenle, bir sorgu koşulunu test etmek, yüksek hesaplama maliyetiyle sonuçlanır. Tüm nesnelerin kapsamlı bir şekilde test edilmesi ise, önemli miktarda işlemci gücü ve disk erişimi gerektirebilir.

2. Kavramlar

Deneylerin yapılabilmesi için PostgreSQL'in [4] bünyesinde barındırdığı üç farklı indeks yapısından yararlanılmıştır. Bunlar; GiST, SP-GiST ve BRIN indeks yapısıdır. Aynı zamanda bu indeks yapılarının sorgu performanslarını arttırmak ve karşılaştırmalar yapabilmek için geohash sıralaması ve cluster yönteminden yararlanılmıştır. Bu bölümde kullanılan indekslerin ve yardımcı yöntemlerin tanımı yapılacaktır. GiST indeks, verileri gruplayarak saklama esasına dayanırken, SP-GiST indeks ise verileri saklamak için uzayı bölümleme esasına dayanır. Bölüm

2.1'de yararlanılan indeks yapılarından bahsedilirken, bölüm 2.2'de ise yararlanılan yardımcı yapılardan bahsedilmiştir. Öncelikle bu iki indeks sınıfından bahsetmekte yarar var. Açıklamalarımız ve deneylerimiz 2-boyutlu veri setleri için olmakla beraber; uzamsal verinin geçerli olduğu 3-boyutlu veri setleri içinde geçerli olacağını öngörebiliriz.

Uzay bölümleme tabanlı yapılardan farklı olarak verileri gruplama tabanlı yapılar; uzayda birbirine yakın olan verileri gruplamayı esas alır. Gruplama esnasında ortaya çıkan dikdörtgen bölgelere minimum kapsayıcı kutu (Minimum Bounding Box) veya minimum kapsayıcı dikdörtgen (Minimum Bounding Rectangle) denilir. Bu gruplama yapısını esas alan ağaçlarda, yaprak düğümleri nesneleri tutarken, ata düğümler ise nesneleri kapsayan dikdörtgenleri veya bu dikdörtgenleri kapsayan diğer dikdörtgenleri tutar. Bu veri yapısı türleri MBR'ları oluştururken nesneler arasındaki ölü alan, nesnelerin birbirine yakınlığı, nesnelerin örtüşme durumları ve oluşan grubun kareselliği gibi parametreleri göz önünde bulundurabilir. Bu sayede verilerin indekse yüklenmesi ve sorguların yürütülmesi uzay bölümleme tabanlı veri yapılarından daha uzun ve maliyetli olabilir. Ancak bu veri yapıları nesneleri tam olarak geometrik şekilleri ile tutmayı hedeflediği için bizlere nesnelerin geometrik ilişkileri ile alakalı sorgular yapma imkanı tanır.

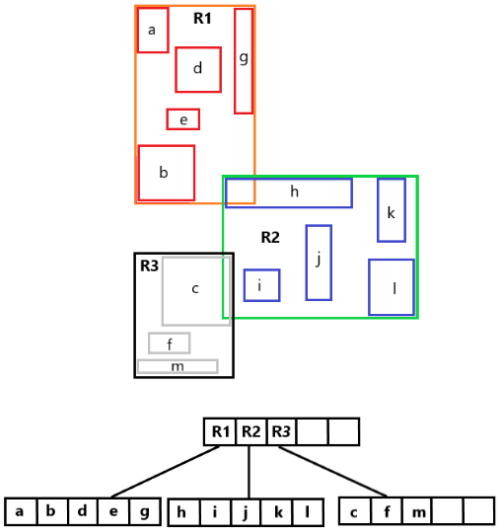
Uzay bölümleme tabanlı veri yapıları, verilerin içerisinde bulunduğu uzayın rekürsif dörtgen parçalara ayrılması esasına dayanır. Bu parçaları hiyerarşik bir şekilde ağaca ekleyerek veri yapısı inşa edilir. Nokta verisi üzerinde uzay bölümleme tabanlı yapılar çok daha avantajlıdır. Noktalar iki veya daha fazla boyutlu bir uzayda bulunuyor olabilir, ancak uzamsal bir uzantıları yoktur [5]. İki boyutlu bir dikdörtgenden veya üç boyutlu bir prizmadan bu konuda farklılık gösterir. Uzayı bölümlediğimizde bir noktanın aynı anda iki uzay parçasında bulunmadığından emin oluruz. Ama iki boyutlu bir dikdörtgenin bir kısmı uzayın bir bölümündeyken diğer kısmı ise başka bölümünde kalıyor olabilir. Ancak bir dikdörtgeni çok boyutlu uzayda bir noktaya dönüştürürsek bu sayede uzay bölümleme tabanlı bir veri yapısında saklayabiliriz. Bu durumda da dikkat etmemiz gereken şey; yaptığımız sorgunun nesnelerin geometrik ilişkileri (örtüşme, kesişme vs.) ile alakalı olamayacağıdır. Nokta formunda tutulan bir dikdörtgen için bu ilişkileri sorgulayamayız. Çünkü bir nesneyi nokta formuna dönüştürdüğümüzde onun geometrik ilişkilerini ve gerçek geometrisini kaybetmiş oluyoruz.

2.1. Yararlanılan İndeks Yapıları

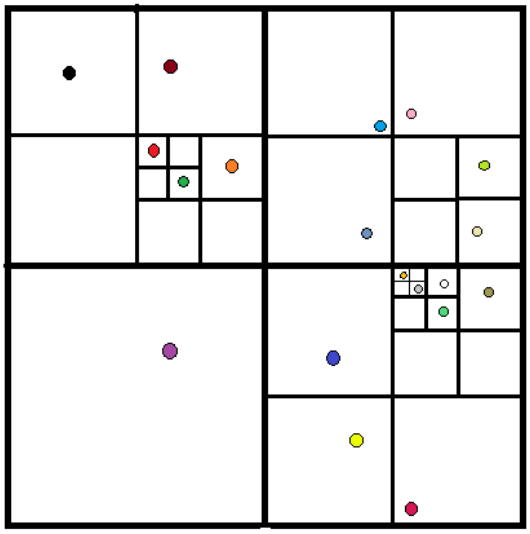
2.1.1. GiST (Generalized Search Trees)

GiST, yani genelleştirilmiş arama ağaçları yapısı, tek türde bir indeks değildir ve bundan dolayı içerisinde birçok indeks yapısı barındırır. Ancak bizim çalışmamızda kullanılan yapı; GiST üzerine R-Tree indeks yapısıdır.

R-Tree, nesneleri birbirlerine yakınlığına göre gruplayan ve temel olarak B-Tree [6] veri yapısının mimarisine benzeyen dengeli bir ağaç yapısıdır. Ağacın hiyerarşisinin oluşmasında nesneleri kapsayan dikdörtgenler (veya kutucuklar) ve bu dikdörtgenleri kapsayan başka dikdörtgenlerden bahsedilir. Bu dikdörtgenler 'minimum kapsayıcı dikdörtgenler (MBR)' olarak adlandırılır. Ağacın yaprak düğümlerinde veri nesnelerine yönelik işaretçiler (pointer) içeren kayıtlar bulunur [7]. Şekil 1'de gösterilen örnek R-Tree modelinde düğümler maksimum beş kayıt alabilmektedir. R1, R2 ve R3 ile temsil edilen MBR'lar minimum ölü alana sebebiyet verecek şekilde düğüm kapasitelerini doldurmaya çalışmışlardır.



Şekil 1. Örnek bir R-Tree modeli



Şekil 2. Quad-Tree veri yapısını baz alan bir uzay-bölümleme (*space-partitioning*) modeli

2.1.2. SP-GiST (Space Partitioned Generalized Search Trees)

SP-GiST terimi, uzay-bölümleme tabanlı arama ağaçları sınıfını ifade eder [8]. PostgreSQL'deki SP-GiST uygulaması ilk olarak Teodor Sigaev ve Oleg Bartunov tarafından geliştirildi [9].

Space-partitioning yöntemi verileri veri yapılarına yerleştirirken, nesneleri kendi aralarında gruplamaktan ziyade, nesnelerin bulunduğu uzayı parçalara ayırır. Şekil 2'de gördüğümüz örnekte; algoritma her düğümde en fazla bir nesne kalacak şekilde uzayı her seferinde dörde bölmüştür. Bu bölünmelerin oluşturduğu uzay parçalarının içerdiği nesneleri gruplayarak veri yapılarına yerleştirir. Quad-Tree [11] bu yaklaşıma verilebilecek en iyi örneklerden biridir. SP-GiST indeks, kendisi için uygun olan kriterler barındıran sorgularda çok hızlı sonuçlar döndürebilir.

2.1.3. BRIN (Block Range Index)

BRIN, ilk defa Alvaro Herrera tarafından 2013 yılında 'minmax indexes' [16] adıyla ortaya atılmıştır. PostgreSQL'e ise 9.5 sürümünden sonra dahil edildi. BRIN, büyük ölçekli tablolarda performansı arttırmak için ortaya çıkmıştır. Buradaki büyük ölçek boyuttan ziyade satır sayısını ifade eder.

BRIN indeksin ortaya çıkışının altında, B-Tree'lerde indekslenen verilerde her satırın tablodaki yerinin kaydedilmesinden ötürü doğurduğu disk alanı ihtiyacının büyük ölçekli verilerin tutulmasında problem teşkil etmesi yatıyor. Bunun yanı sıra B-Tree oluşturulurken ve kullanılırken disk erişimi (I/O) ihtiyacı da büyük verilerin B-Tree'de indekslenmesini bir soru işareti haline getiriyordu. BRIN'in B-Tree'den farklı olarak yaptığı ise verinin tutulduğu sayfaları gruplaması ve her grupta minimum ve maksimum değerleri tutmasıdır. BRIN uzamsal veriler üzerinde uygulandığında verilerin sayısal değişim aralığı yerine geometrik dağılan nesnelerin MBR'ları saklanır. Bu sebeple BRIN indeksinin boyutu çok küçüktür.

2014 yılında Mark Wong tarafından yapılan tabloya toplu veri yükleme deneyinde [10], 10 GB büyüklüğünde oluşturulan bir veri setinin indeks barındırmayan, B-Tree indeks barındıran ve BRIN indeks barındıran tabloya yüklenmesi ile ilgili deneyde, BRIN indeks barındıran tablonun yüklenme süresi, hiçbir indeks barındırmayan tablo ile neredeyse aynı iken B-Tree indekse sahip tablo ile arasında büyük bir fark oluştuğu gözlemlenmiştir.

2.2. Yararlanılan Yardımcı Yöntemler

2.2.1. Geohash

Geohash, 2008 yılında Gustavo Niemeyer tarafından bulunmuş olsa da bu fikrin temelleri G. M. Morton tarafından 1966 yılında ortaya atılmıştır. Geohash, uzay doldurma eğrilerini (*space filling curves*) [12] kullanarak uzayı bölümler ve bu bölümleri sıraya sokar. Bu sıralamayı enlem ve boylam kodları olarak adlandırdığımız bitlerden oluşan kodlar ile tutar.

Özellikle BRIN indeksi kullanırken, eğer nesneleri geohash'e göre sıralayıp indeksin içine gönderirsek, tüm nesneler space filling curve adı verilen eğrilere göre sıralanmış biçimde olacağı için indekste bulunan blokların içerisinde uzayda ki yakınlıkları rastgele olan nesneler değil de, daima uzayda birbirine yakın olan nesneler bulunacaktır. Bu sayede her blok için tutulan minimum ve maksimum değerler daha anlamlı olacaktır. Bu yöntemin indeksin çalışmasına nasıl katkıda bulunduğunu deneyler kısmında göreceğiz.

3. Deney Ortamı

Deneyler için PostgreSQL veri tabanı yönetim sistemi [14] seçilmiştir. PostgreSQL, kurumsal sınıf bir açık kaynak veri tabanı yönetim sistemidir. Gelişmiş veri türlerini ve gelişmiş performans optimizasyonunu destekler. Bu tarz özellikler, yalnızca Oracle Database ve Microsoft SQL Server gibi ticari veri tabanlarında bulunan özelliklerdir. Aynı zamanda PostgreSQL içerisinde PostGIS uzantısını barındırır. PostGIS, PostgreSQL veri tabanı için uzamsal bir veri tabanı genişletmesidir. Konumsal sorguların SQL'de çalıştırılmasına izin vererek coğrafi nesneler için destek sağlar [4]. PostGIS uzantısı kullanıcıya binden fazla coğrafi fonksiyonu sağlar ve tüm iki boyutlu ve üç boyutlu uzamsal veri türlerini içerir. Bu fonksiyonların tamamını barındıran liste ve açıklamaları için [13]'e bakılabilir.

Deneylerde PostgreSQL'in 13.2 versiyonu kullanılmıştır. Bunun yanı sıra deneyler PgAdmin 4 v5 platformu kullanılarak yapılmıştır. Verileri görselleştirmek için QGIS 3.16.13 sürümü kullanılmıştır. Tablolar ve grafikler ise Google E-Tablolar aracılığı ile oluşturulmuştur. Deneyler, aşağıda belirtilen donanım koşullarında yapılmıştır;

- Intel Core i5-8300H 2.30 GHz,
- 8 GB RAM,
- 1000 GB HDD,
- Windows 10 Home

3.1. Deney Metodolojisi

Deneyler PgAdmin platformunda yazılan SQL komutları ile gerçekleştirilmiştir. Sorgu sürelerinin ölçümü için PostgreSQL'in sağladığı 'explain analyze' komutu kullanılmıştır. Bu komutun kullanımının ayrıntıları için [13]'de 9. kısma bakılabilir. Sorgu süreleri bilgisayarın anlık işlem durumuna göre farklılık gösterebildiği için, her sorgu tutarlı sonuçlar elde edilene kadar birkaç kez tekrarlanmıştır. Sonrasında her sonuç tablolarda ve grafiklerde kullanılmak üzere tek tek kaydedilmiştir. Deneylerde kullanılacak olan indeks yapılarını oluşturmak için yazılan SQL komutları tablo 1'de görülebilir.

Tablo 1. İndeks oluşturmak için kullanılan SQL komutları

GiST	`CREATE INDEX turkey_point_time_geom_gistidx`
	`ON turkey_point_time`
	`USING GIST (geom);`

GiST (3D)	`CREATE INDEX turkey_point_time_idx ON turkey_point_time` `USING GIST (geom gist_geometry_ops_nd);`

SP-GiST	`CREATE INDEX turkey_point_time_geom_spgistidx`
	`ON turkey_point_time`
	`USING SP-GIST (geom);`

BRIN	`CREATE INDEX turkey_point_time_geom_brinidx`
	`ON turkey_point_time`
	`USING BRIN (geom)`
	`WITH (pages_per_range = 1);`

CLUSTER	`CLUSTER turkey_point_time` `USING turkey_point_time_geom_brinidx;`

4. Deneyler

Bu bölümde toplu yükleme (bulk loading) ve pencere sorgusu (window query) deneyleri ve bu deneylerin sonuçları paylaşılacaktır.

4.1. Bulk Loading

Uzamsal veriler genelde büyük boyutlarda olabileceği için bu verilerin veri tabanlarına tek tek değil de, toplu olarak bir arada yüklenmesi indeks oluşturma verimini önemli ölçüde arttıran bir yaklaşımdır.

Bu deneyde, verilerin boş bir tabloya ve çeşitli indeks bulunduran tablolara toplu yüklenme sürelerine bakılarak, indekslerin yüklenme sürelerinde oluşturduğu geciktirmeleri görmek ve hangi indeksin verileri daha çabuk yüklediğini karşılaştırmak amaçlanmıştır. Toplu yükleme için birkaç yöntem bulunmaktadır. Bu yöntemlerden en iyi performans veren ikisi 'pg_bulkload tool' ve 'COPY' [15] metodudur. 'pg_bulkload' şu anda yalnızca linux çekirdeği için kullanılabilir olduğundan, bu deneyde 'COPY' metodu kullanılacaktır. Tablo 2'de toplu yükleme için yazılan SQL komutları görülebilir. Bu komutlar ile verileri öncelikle .csv uzantılı bir dosyaya daha sonra da üzerinde indeks bulunduran ve bulundurmayan tablolara aktaracağız.

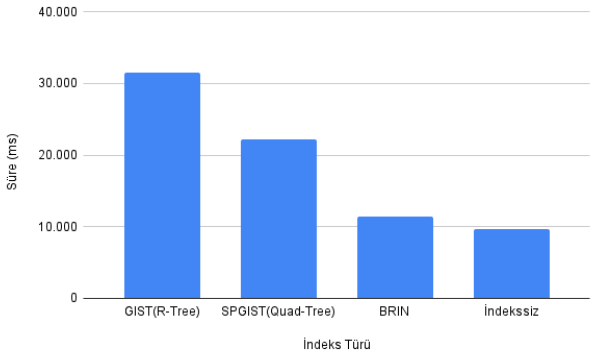
Tablo 2. Tablodan CSV dosyasına ve CSV dosyasından tabloya veri setini toplu yüklemek için kullanılan SQL komutları

Tablodan CSV dosyasına	`COPY gis_osm_buildings_a_free_1` `TO 'C:\Users\HP\Desktop\buildings.csv' (FORMAT csv);`

CSV dosyasından tabloya	`COPY buildings(gid, osm_id, code, fclass, name, type, geom)`
	`FROM 'C:\Users\HP\Desktop\buildings.csv'`
	`DELIMITER ','`
	`CSV HEADER;`

4.1.1. Bulk Loading Test 1

Bu test 1.1 milyon adet 114 megabayt büyüklüğündeki nokta verisinin indeks bulundurmayan, GiST indeks bulunduran, SP-GiST indeks tanımlanan ve BRIN indeks tanımlanan tablolara toplu yüklenmesi sonucu oluşan sürelerin ölçülmesi üzerinedir.

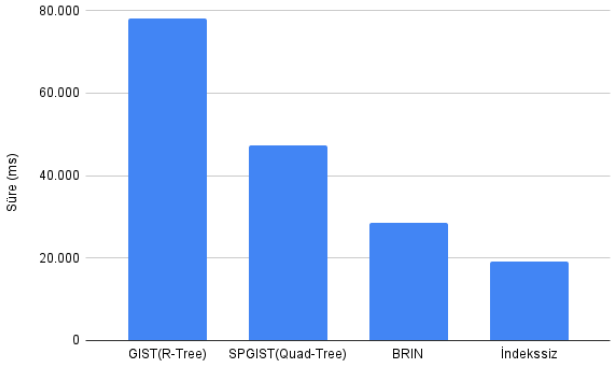


Şekil 3. 114 megabayt büyüklüğündeki nokta verisi için Bulk Loading deneyi süreleri

Şekil 3'te görüldüğü üzere indeks bulundurmayan tablo verileri beklenildiği gibi hepsinden hızlı yüklemiştir. BRIN indeksin ise diğer indeks türlerine göre önemli bir üstünlüğü görünmektedir. Uzay-bölümleme tabanlı SP-GiST indeks de, GiST indekse göre daha iyi bir yükleme performansı göstermiştir. GiST indeks özelinde bakıldığında, R-Tree gibi nesneleri gruplamaya dayalı veri yapılarının diğer indekslere nazaran verileri daha yavaş yüklediğini görmemiz mümkün.

4.1.2. Bulk Loading Test 2

Bu deneyde ise veri miktarını arttırarak tekrar bir karşılaştırma yapacağız. Aynı zamanda önceki deneyden farklı olarak nokta verisi yerine iki boyutlu poligon verisi kullanacağız. Burada 2.3 milyon adet poligondan oluşan bir veri seti kullanıyoruz. Veri setinin büyüklüğü ise 624 megabayt'tan oluşmaktadır.



Şekil 4. 624 megabayt büyüklüğündeki iki boyutlu poligon verisi için toplu yükleme deneyi süreleri

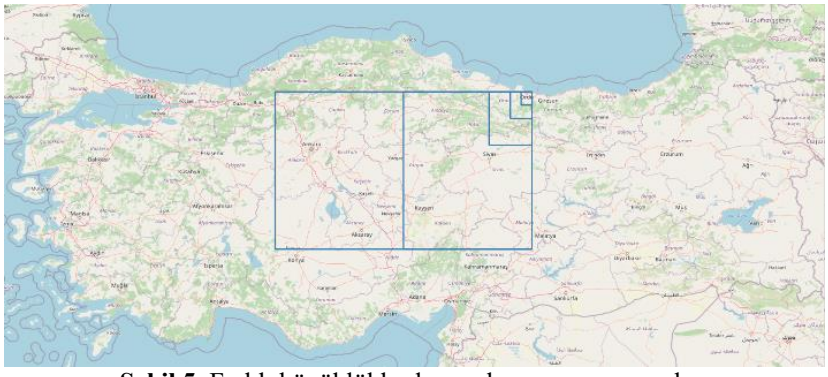
Bu deneyde bir önceki deneyden yaklaşık 5.5 kat daha büyük bir veri seti kullandık. Şekil 3 ve şekil 4'ü karşılaştırdığımızda, veri miktarındaki büyüme; verinin, indekse sahip tablolara yüklenme süresi ile indeks bulundurmayan tabloya yüklenme süresi arasındaki farkı daha da açtı. Bunun dışında oran olarak GiST indeks'in yükleme performansının diğer indekslere nazaran biraz daha kötüleştiğini söylemek mümkün. Verinin türünün (uzamsal boyut olarak) değişmesi ise performansa indekslerin birbirlerine karşı olan performansa herhangi bir etki yapmadı. BRIN indeksin ise diğer indekslere karşı yükleme sürelerinde çok daha avantajlı olduğu görülüyor.

4.2. Pencere Sorgusu

Pencere (window) sorguları, iki boyutlu uzamda uygulandığında, her iki eksende belirli bir aralık içerisindeki uzamsal nesnelerin bulunmasıdır. Ayrıca bu sorgu türü en sık kullanılan uzamsal sorgulardan birisidir.

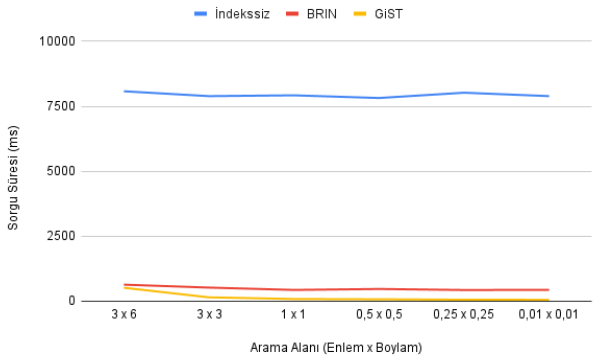
4.2.1. Pencere Sorgusu Test 1

İndeks bulundurmayan tablo, BRIN indekse sahip tablo ve GiST (R-Tree) indekse sahip tablo üzerinde farklı arama alanlarını kapsayan pencere sorguları çalıştırılıp, sorgu sürelerini ölçeceğiz. Burada kullanacağımız veri seti yaklaşık 2.3 milyon adet iki boyutlu poligon nesnesi içeriyor. Deneyde postgis'in 'st_within()' fonksiyonundan yaralanacağız. Bu fonksiyon; kendisine parametre olarak verilen iki geometriden birincisinin içinde bulunan geometrileri – eğer indeks varsa- indeks yardımıyla bulur [13].



Şekil 5. Farklı büyüklüklerde yapılan pencere sorguları

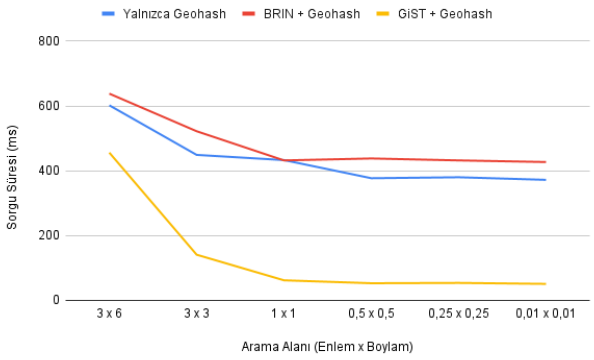
Şekil 6'da görüldüğü üzere indeks kullanılmayan tablo çok küçük ve çok büyük arama alanlarında kötü bir performans verirken orta büyüklükteki bir arama alanı için en iyi performansını göstermiştir. GiST indeks ise verilerin iki boyutlu olmasından ve R-Tree'nin iki boyutlu veriler üzerinde etkili bir veri yapısı olmasından dolayı diğerlerine nazaran oldukça iyi bir performans göstermiş ve tüm arama alanlarında yok denecek kadar kadar az bir sürede sonucu getirmiştir. BRIN indeks indekssiz tabloya göre gayet iyi bir performans göstermesine rağmen yine de GiST indeksin arkasında kalmıştır.



Şekil 6. 3 farklı indeks yapısına sahip tablolar üzerinde yapılan pencere sorgusu performansı

4.2.2. Pencere Sorgusu Test 2

Bu deneyde verileri tablolara aktarırken 'geohash' esasına göre sıralayarak aktaracağız. Bu sayede geohash yönteminin sorgular ve indeks yapıları üzerindeki performansını ölçümlemiş olacağız.



Şekil 7. Verileri geohash'e göre sıralayarak yapılan pencere sorgusu performansı

Şekil 7'de görüldüğü üzere indeks bulundurmayan bir tablodaki verileri herhangi bir veri yapısı kullanmadan yalnızca geohash esasına göre sıralı halde tutarak oldukça iyi bir sorgu performansı yakalamış olduk. Geohash nesneleri uzay doldurma eğrilerine göre sıraladığı için uzayda birbirine yakın bulunan nesnelerin tabloda da birbirine yakın bulunması, rastgele sıralı biçimde bulunan sorgu esnasında nesnelerin çok daha çabuk bulunmasına olanak sağladı.

Bu kısımda dikkat edilmesi gereken diğer bir konu arama alanı arttıkça GiST indeksin performansının kötüleşmesi ve indeks bulundurmayan sorgu performansına yaklaşması. Bunun

nedeni; arama alanı büyüdükçe veri setini ‘budamanın’ anlamsızlaşmaya başlamasıdır. Arama alanı ne kadar küçükse arama ağacı bulunduran ve bulundurmayan tablo arasındaki sorgu performansı o kadar farklılaşır.

Tablo 3. Pencere sorgusu için hazırlanan SQL komutları

Pencere Sorgusu	`SELECT name, geom FROM buildings_gist WHERE ST_Within(geom, ST_MakeEnvelope(37, 40, 38, 41, 4326))`

Tablo 4. Test 1 pencere sorgusu sonuçları (milisaniye cinsinden)

	İndekssiz	**BRIN**	**GiST**
Arama Alanı (Enlem x Boylam)	Süre (ms)	Süre (ms)	Süre (ms)
3 x 6	8077	637	518
3 x 3	7890	524	149
1 x 1	7921	434	85
0,5 x 0,5	7818	471	74
0,25 x 0,25	8022	431	58
0,01 x 0,01	7890	437	55

Tablo 5. Test 2 pencere sorgusu sonuçları (milisaniye cinsinden)

	Yalnızca Geohash	**BRIN + Geohash**	**GiST + Geohash**
Arama Alanı (Enlem x Boylam)	Süre (ms)	Süre (ms)	Süre (ms)
3 x 6	602	638	456
3 x 3	449	522	141
1 x 1	433	432	62
0,5 x 0,5	377	438	53
0,25 x 0,25	380	432	54
0,01 x 0,01	372	427	51

Sonuç

Yapılan çalışmalar sonucunda, verilerin toplu yüklenmesinde BRIN indeksin diğer indeks yapılarına göre yükleme süresi açısından büyük bir avantaj sağladığı görülmüştür. Bunun yanı sıra verinin uzamsal boyutunun (nokta, poligon gibi) değişmesi uzay bölümleme tabanlı indeks yapısının veri gruplama tabanlı indeks yapısına karşı avantajını değiştirmemiştir.

Pencere sorgusunda ise yapılan ilk testte, gittikçe daralan arama alanının belli bir ölçüden sonra GiST indeksin performansını önemli ölçüde arttırdığını gözlemleyebiliyoruz. Fakat İndekssiz tablonun ve BRIN indeks bulunduran tablonun sorgu performansında arama alanının

daralması ile kayda değer bir iyileşme görülmemiştir. Bu da BRIN indeksin, sağladığı genel avantaj dışında, arama alanı veya veri sayısı ile bağlantılı bir avantaj sağlamadığını göstermiştir. Bu çalışmanın neticesinde; uzamsal sorgularda indeks kullanmanın indeks kullanılmayan durumlara kıyasla sorgu sürelerini büyük ölçüde kısalttığı ve özellikle daha küçük arama alanlarında yapılan pencere sorgularının, R-Tree gibi uzamsal veriler için üretilmiş indeks türleri kullanılarak yapılmasının oldukça avantajlı olduğu anlaşılabilir. Test 2'de ise verileri geohash esasına göre sıralamanın indeks kullanılan ve indeks kullanılmayan durumlardaki etkisi gözlemlenmiştir. Verilerin yalnızca geohash ile sıralanarak yapıldığı sorguda, bir önceki testte yapılan indekssiz sorguya göre çok daha iyi bir sorgu süresi performansı ortaya çıkardığı gözlemlenebilir. Öyle ki; her büyüklükteki arama alanı için, yalnızca BRIN indeks kullanılarak yapılan sorgudan daha iyi performans gösterdiğini de gözlemleyebiliriz. Bunun dışında, geohash'e göre sıralanmış verilerin BRIN ile indekslenerek yapıldığı sorgu, BRIN indeks için gözle görülür bir performans değişikliği göstermemiştir. Veriler geohash esası ile sıralandığında ve GiST indeks ile indekslendiğinde, yalnızca GiST kullanılan deneydekinden biraz daha iyi bir performans göstermiştir. Gelecek çalışmalarda verilerin geohash gibi uzayda birbirine yakın olacakları düzende sıralanarak indekslere eklenmesi üzerinde daha farklı deneyler yapılabilir. Bu tarz çalışmalar, bir veri grubu ve sorgu türü için en optimal sorgu sürelerini veren indekslerin performanslarının daha da arttırılmasına yardımcı olabilir.

Kaynakça

[1] Google Maps uygulamasına ait bazı istatistikler. Erişim: 30 Nisan 2022. https://cloud.google.com/blog/products/maps-platform/9-things-know-about-googles-maps-data-beyond-map

[2] İndeks kullanımının avantajları. Erişim: 17 Nisan 2022. https://www.postgresql.org/docs/current/indexes-intro.html

[3] Kalay, M.U. "Novel approaches on bulk-loading of large scale spatial datasets". Concurrency and Computation: Practice and Experience 2021.

[4] PostgreSql dokümantasyon web sayfası. Erişim: 26 Nisan 2022. https://postgresql.org/docs/

[5] SPGiST dev web sayfası. Erişim: 20 Nisan 2022. http://sai.msu.su/~megera/wiki/spgist_dev

[6] Comer, D. "Ubiquitous B-Tree". ACM Comput. Surv. 1979; 11(2):121–137.

[7] Guttman, A. "R-Trees: A Dynamic Index Structure for Spatial Searching". SIGMOD Rec. 1984; 14(2):47–57.

[8] Aref, Walid G. Ilyas. Ihab F. "SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees". Journal of Intelligent Information Systems 2001; 17:215-240.

[9] SPGiST dev web sayfası. Erişim: 20 Nisan 2022. http://sai.msu.su/~megera/wiki/spgist_dev

[10] Loading Tables and Creating B-Tree and Block Range Indexes. Erişim 29 Nisan 2022. https://www.2ndquadrant.com/en/blog/loading-tables-creating-b-tree-block-range-indexes/

[11] Samet H. Multidimensional Data Structures for Spatial Applications. İçinde: Algorithms and Theory of Computation Handbook, Second Edition, Volume 1 [Internet]. Chapman and Hall/CRC; 2009 [a.yer 17 Haziran 2022]. s. 1-42. (Chapman & Hall/CRC Applied Algorithms and Data Structures series; c. 3). Erişim adresi: http://www.crcnetbase.com/doi/abs/10.1201/9781584888239-c6

[12] Sagan, H. Space-Filling Curves by Hans Sagan.. Springer New York; 1994.

[13] Postgis dokümantasyon web sayfası. Erişim: 1 Mayıs 2022. https://postgis.net/docs/

[14] Ramakrishnan, R, Gehrke, J, Gehrke, J. Database management systems. McGraw-Hill New York; 2003.

[15] Farklı postgresql komutları ile bulk loading deneyi. Erişim: 27 Nisan 2022. https://www.highgo.ca/2020/12/08/bulk-loading-into-postgresql-options-and-comparison/

[16] Alvaro Herrera tarafından yapılan minmax indexes tanımı. Erişim: 15 Nisan 2022. https://www.postgresql.org/message-id/20130614222805.GZ5491@eldon.alvh.no-ip.org