T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KÜMELEMEYE DAYALI OTOMATİK DİZİN SEÇİM ARACI GERÇEKLEŞTİRİMİ

MEHMET AKİF YANATMA

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
DR. ÖĞR. ÜYESİ MUSTAFA UTKU KALAY

İSTANBUL, 2019

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

KÜMELEMEYE DAYALI OTOMATİK DİZİN SEÇİM ARACI GERÇEKLEŞTİRİMİ

Mehmet Akif YANATMA tarafından hazırlanan tez çalışması 20.03.2019 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Dr. Öğr. Üyesi Mustafa Utku KALAY

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Dr. Öğr. Üyesi Mustafa Utku KALAY

Yıldız Teknik Üniversitesi _____

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Doç. Dr. Atakan KURT

İstanbul Üniversitesi _____

ÖNSÖZ

Bu tez çalışması boyunca yardım ve emeğini hiç esirgemeyen, görüş ve önerilerileriyle bana her zaman destek olan değerli hocam Dr. Öğr. Üyesi Mustafa Utku KALAY'a çok teşekkür ederim.

Tüm eğitim hayatım boyunca, özveri ve desteği ile hep yanımda olan anne ve babama, bu tez çalışması boyunca da verdikleri maddi ve manevi destekten ötürü teşekkürü bir borç bilirim.

Son olarak bu tez çalışmamı bitirmem için moral ve motivasyonumu üst düzeyde tutmamı sağlayan, tez çalışmamı her zaman kendi isteklerinin önüne koyan eşim Aynur'a teşekkür ederim.

Mart, 2019

Mehmet Akif YANATMA

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

B	Depolama kısıtı
B_1	Product işleminde soldaki tablonun blok sayısı
B_2	Product işleminde sağdaki tablonun blok sayısı
C	Dizin kümesi
d	Uzay boyutu
D	Veri seti
eşik1	Niteliklerin toplam frekans eşiği
eşik2	Niteliklerin seçicilik eşiği
eşik3	Tabloların satır sayısı eşiği
fds	Farklı değer sayısı
l_j	j numaralı dizin
k	Küme sayısı
m_1	Birinci küme merkezi
m_2	İkinci küme merkezi
n	Veri setinin eleman sayısı
o_{random}	Rastgele seçilmiş nesne
Q_j	j numaralı sorgu
W	İş yükü

KISALTMA LİSTESİ

1NF	First Normal Form
2NF	Second Normal Form
3NF	Third Normal Form
AISIO	Automated Index Selection Integrated into Optimizer
BCNF	Boyce Codd Normal Form
CLARA	Clustering Large Applications
CLARANS	Clustering Large Applications Based Upon Randomized Search
COLT	Continuous On-Line Tuning
ER	Entity Relationship
JDBC	Java Database Connectivity
NP	Nondeterministic Polynomial Time
OSDL	Open Source Development Lab
PAM	Partitioning Around Medoids
SQL	Structured Query Language
TPC	Transaction Processing Performance Council
UML	Unified Modelling Language
VT	Veri Tabanı
VTYS	Veri Tabanı Yönetim Sistemi

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

KÜMELEMEYE DAYALI OTOMATİK DİZİN SEÇİM ARACI GERÇEKLEŞTİRİMİ

Mehmet Akif YANATMA

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Dr. Öğr. Üyesi Mustafa Utku KALAY

Günümüzde sayısal veri miktarının artmasıyla birlikte veri tabanı sistemlerinin kullanımı da artmıştır. Bu kadar çok veri arasından istenilen veriye hızlı bir şekilde ulaşmada ise veri tabanlarının fiziksel tasarımı önemli rol oynamaktadır. Fiziksel veri tabanı tasarımının önemli bir kısmını dizin seçimi oluşturmaktadır. Dizinler, tüm tablonun taranmasına gerek duyulmadan belirli seçim şartlarına uyan satırlara ulaşmayı sağlayan yardımcı yapılardır. Yüzlerce tablo ve binlerce sütun içeren ilişkisel bir veri tabanı için milyarlarca dizin oluşturabilme ihtimali vardır. Ancak bir veri tabanı için dizin sayısı sınırsız bir şekilde arttırılamaz. Çünkü veri tabanında yapılan bir güncellemeden sonra yeni veri tabanı ile tutarlı kalmak için dizinlerin de güncellenmesi gerekir, dolayısıyla dizinlerin bir bakım maliyeti vardır. Diğer taraftan dizinler de birer dosya olduklarından depolama maliyetleri vardır. Bu esaslar göz önünde bulundurulduğunda dizin seçim problemi, NP-tam bir problem olarak karşımıza çıkmaktadır. Böyle bir problem karşısında veri tabanı yöneticisinin bir veri tabanı için dizin seçimi yapması çok zordur.

Bu tez kapsamında dizin seçim problemini otomatik olarak çözmek için dizin seçim aracı geliştirilmiştir. Dizin seçim aracı geliştirilirken kümeleme tekniklerinden yararlanılmıştır. Geliştirilen dizin seçim aracı ile bir iş yükü için uygun dizin kümesi seçerek, veri tabanı yöneticisi üzerindeki iş yükünü hafifletmek ve iş yükünü oluşturan sorguların cevaplanması için gereken toplam zamanın ve toplam disk erişim sayısının azaltılması hedeflenmiştir.

Dizin seçim aracının çalışma prensibi, sorgulardan oluşan bir iş yükünü, birbirine benzer sorgular bir araya gelecek şekilde kümelere ayırmaya çalışmaktır. Benzer sorguları biraraya getirmedeki amaç ise birbirine benzer sorguların ortak nitelikleri üzerinde dizin oluşturma önerisinde bulunmaktır. Böylece oluşturulacak dizinlerin birden çok sorgu için faydalı olması beklenmektedir.

Yapılan performans değerlendirmesi testlerinde iş yükünü oluşturan sorgular ardışık ve eş zamanlı olmak üzere 2 farklı şekilde çalıştırılmıştır. Testler ile iş yükünün işlenmesi için gereken toplam disk erişimi sayısı ve toplam cevap süresi ölçülmüştür. İş yükünün işlenmesi için gereken toplam disk erişimi sayısı ve toplam cevap süresinde, kümeleme yöntemi ile elde edilen dizin önerilerinin kullanılması ile büyük oranda iyileşme olduğu görülmüştür.

Anahtar Kelimeler: Fiziksel veri tabanı tasarımı, dizin, dizin seçimi, kümeleme, SimpleDB veri tabanı sistemi

ABSTRACT

AUTOMATIC INDEX SELECTION TOOL IMPLEMENTATION BASED ON CLUSTERING

Mehmet Akif YANATMA

Department of Computer Engineering

MSc. Thesis

Adviser: Assist. Prof. Dr. Mustafa Utku KALAY

Nowadays, with the increase in the number of digital data, the use of database systems has increased. The physical design of the databases plays an important role in reaching the requested data quickly among such a large number of data. An important part of the physical database design is the index selection. Indexes are auxiliary structures that allow access to rows that match specific selection conditions without having to scan the entire table. There is a possibility of creating billions of indexes for a relational database containing hundreds of tables and thousands of columns. However, the number of indexes for a database cannot be increased limitlessly. Because after an update to the database, the indexes need to be updated to remain consistent with the new database, so the indexes have a maintenance cost. On the other hand, indexes are stored in ordinary files like other data files, so they have storage cost. When these criteria are taken into consideration, the problem of index selection is an NP-complete problem. In the face of such a problem, it is very difficult for the database administrator to select the appropriate indexes for a database.

In this thesis, an index selection tool has been developed to automatically solve the index selection problem. Clustering techniques were used to develop the index selection tool. By selecting the appropriate index set for a workload with the developed index

selection tool, it is aimed to lighten of responsibility of database administrator and to reduce the total time and total disk access count required to answer the queries that make up the workload.

The working principle of the index selection tool is to divide a workload into clusters such that similar queries come together. The purpose of grouping similar queries is to suggest indexing on the common attributes of similar queries. By this way, the indexes to be created are expected to be useful for multiple queries.

In the performance evaluation tests, the queries that constitute the workload were executed in two different ways as consecutive and concurrent. The total number of disk access and total response time required to process the workload were measured with these tests. The total number of disk access count and the total response time required to process the workload has been greatly reduced by the use of index recommendations obtained by the clustering method.

Anahtar Kelimeler: Physical database design, index, index selection, clustering, SimpleDB database system

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Dizinler, tüm tabloyu taramaya gerek duymadan belirli seçim ifadelerine uyan satırlara ulaşmayı sağlayan yardımcı dosyalardır [1]. Dizinler, tablonun sütun veya sütun dizilimleri üzerinde oluşturulabilen ve sorgu işleme performansını büyük ölçüde arttıran erişim yapılarıdır [2]. Birer dosya olduklarından alan kaplarlar. Ayrıca veri tabanını güncellemek, yeni veri tabanı durumuyla tutarlı kalmak için dizinlerin de güncellenmesini gerektirir. Bu bakımdan bir dizin, veriye erişimi hızlandırırken veri tabanı bakımını yavaşlatır [1]. Bu esaslara göre dizinler depolama maliyetinin yanında güncelleme, ekleme ve silme sırasında bakım maliyetine sahip olduklarından, bir veri tabanı tablosundaki dizin sayısı sınırsız bir şekilde arttırılamaz. Bu durumda bir veri tabanı için uygun dizinlerin seçilmesi karmaşık bir problem olarak karşımıza çıkmaktadır. Dizin seçim problemi, ilişkisel bir veri tabanı sistemi üzerinde yürütülen bir iş yükünün işlenmesi için gereken maliyeti en aza indirmek amacıyla belirli bir depolama alanı kısıtını da dikkate alarak her bir tablo için uygun dizinleri seçilmesidir [3].

Yüz tane tablo ve bin tane sütun içeren bir ilişkisel veri ambarında, bir tablo için sütunlar ve sütunlardan oluşan alt kümeler üzerinde oluşturulabilecek dizinler dikkate alınırsa, milyarlarca dizin oluşturmak mümkündür. Bu yüzden dizin seçimi, fiziksel tasarımın merkezinde yer alan, klasik ve çok zor bir problemdir. Çok az sayıda ya da yanlış seçimden oluşan dizinlerin varlığında, sorgular veri tabanının büyük bir bölümünün taranmasına sebep olurken, çok fazla dizin ise yüksek güncelleme maliyetine sebep olur. Öngörülemeyen anlık sorgular ise performans sorununu daha da kötüleştirebilir [4].

Dizin seçim problemi, NP-tam(belirsiz çokterimli tam) bir problemdir [2]. Problemin NP-tam olduğunun ispatı, "2.3.5 Fiziksel Tasarım Probleminin Karmaşıklığı" başlığı altında yapılmıştır.

Ameri vd. tarafından [5] yapılan çalışmada, dizin seçim probleminin zorluğu sayısal bir örnek üzerinden gösterilmeye çalışılmıştır. Bunun için bir veri tabanı sisteminin B-tree, hash, bitmap vb. gibi s tane farklı dizin türü oluşturmaya imkan verdiği varsayılmıştır. Hesaplamalarda n tane niteliğe sahip olduğu varsayılan T tablosu kullanılmıştır. Dizinler, bir veya birden fazla nitelik üzerinde oluşturulabilirler ancak bu örnekte anlaşılırlığı artırmak için sadece bir nitelik üzerinde oluşturulan tekli sütun dizinlerinin kullanıldığı, çoklu sütun dizinlerinin kullanılmadığı varsayılmıştır.

T tablosunun niteliklerinden sadece bir tanesi üzerinde dizin oluşturulacak olursa öncelikle bunun hangi nitelik olacağına karar verilmelidir. Çünkü T tablosu, n tane nitelik içeriyorsa, n farklı seçim yapılabilir. Eğer ayrı ayrı iki nitelik üzerinde tekli sütun dizinleri oluşturulmak istenirse n x (n-1) tane farklı seçim yapılabilir. k ≤ n olmak üzere, k, kaç tane nitelik üzerinde tekli sütun dizini oluşturulacağını göstermektedir. Bu durumda, k tane nitelik için oluşturulacak dizin sayısı n x (n − 1) x (n − 2) x ... x (n − k + 1) adet veya diğer bir gösterim şekli ile n!/(n − k)! adet olmaktadır. Ayrıca s tane farklı dizin türü de dikkate alınırsa k tane nitelik için oluşturulacak farklı dizin sayısı s^k x n!/(n − k)! adet olmaktadır. Bu durumda tüm k değerleri için oluşturulabilecek farklı dizin sayısını hesaplayabilmek için (1.1)'deki şu formül ortaya çıkmaktadır:

$$\sum_{k=1}^{n} \frac{(s^k)\, n!}{(n-k)!} \qquad (1.1)$$

Mesela, 4 farklı dizin tipinin oluşturulabildiği bir veri tabanı sisteminde 5 nitelik içeren T(f1, f2, f3, f4, f5) tablosu için 157780 tane birbirinden farklı tekli sütun dizini oluşturulabilmektedir. Bu sonuç göstermektedir ki daha fazla dizin tipini oluşturmaya imkan tanıyan bir VTYS ve çok sayıda tablo içeren bir veri tabanında, çoklu sütun dizinlerinin de hesaba katılmasıyla oluşturulabilecek dizin sayısı çok daha büyük olur.

Veri tabanları için en iyi dizin kümesi seçim çalışmaları, yetmişli yıllardan beri devam etmektedir [6]. Veri tabanlarının geniş çaplı kullanımı, artan veri büyüklüğü ve hızlı veri erişimi talebi göz önünde bulundurulduğunda, fiziksel veritabanı tasarımı ve veritabanı

performans ayarlama sürecinin yönetimini otomatikleştirmek önem kazanmıştır [7]. Bundan dolayı geçmişten günümüze dizin seçim problemini otomatik olarak çözmek için çeşitli çalışmalar yapılmıştır.

Aouchie vd. tarafından [6] dizin seçim problemini çözmek için yapılan çalışmada veri madenciliği tekniği kullanılmıştır. Bu çalışmada aday dizin kümesi önermek için iş yükünü oluşturan sorgulardan yararlanan bir araç tasarlanmıştır. Temel fikir olarak bir dizinin kullanılmasından elde edilen fayda ile iş yükünde sıklıkla yer alan nitelikler arasında güçlü bir bağlantı olduğu esas alınmakta ve bu yüzden sıklıkla kullanılan nitelikler dizin seçiminde kullanılmak için araştırılmaktadır. Bu düşünce ile hareket günlüğünden elde edilen sorgulardan oluşan bir iş yükü, geliştirilen araca girdi olmakta ve SQL sorgu analizi ile üzerinde dizin oluşturulabilecek tüm nitelikler belirlenmektedir. Daha sonra satırları sorgulardan, sütunları ise üzerinde dizin oluşturulabilecek niteliklerden oluşan sorgu-nitelik matrisi oluşturulmaktadır. Bu matris, frekans öğe kümeleri için çıkarılan içeriği temsil etmektedir. Frekans öğe kümelerini hesaplamak için close algoritması kullanılarak aday dizin kümesi elde edilmiştir. En son olarak aday dizin kümesinden kullanılma frekansı en yüksek olan dizinler seçilmektedir. Bir test veri tabanı üzerinde yapılan deneyler sonucunda, geliştirilen araç tarafından oluşturulan dizinler sayesinde %15 ile %25 arasında performans artışı olduğu görülmüştür.

Zaman vd. tarafından [3] yapılan diğer bir çalışmada ise otomatik dizin seçimi için kümeleme tekniği kullanılmıştır. Bu çalışmaya göre, benzer sorgulardan oluşan bir grupta sık sık ortaya çıkan niteliklerin üzerinde dizin oluşturmanın yararlı olacağı sezgisi yürütülmektedir. Ayrıca bu çalışmada dizin seçim probleminin farklı araştırmacılar tarafından farklı şekillerde ele alındığı belirtilmektedir. Buna göre, dizin seçim problemini çözmek için geliştirilen araçları ikiye ayırmışlardır. Bunlardan ilki, lineer programlama, veri madenciliği ve diğer maliyeti en aza indirgeme yöntemlerini kullanan harici araçlardır. İkinci kategori ise çeşitli dizin yapılandırmaları için maliyet tahminleri vermek ve en düşük maliyet tahminiyle bir yapılandırma önermek için sorgu iyileştiriciyi kullanan araçlardır. İlk kategoride belirtilen araçların iyileştirici ile bağlantılarının kesilmiş olması, bir dezavantaj olarak belirtilmiştir. Sebep olarak da bir iş yükü işlenirken, araç tarafından önerilen ama iyileştirici tarafından kullanılmayan dolayısıyla veri tabanı yönetim sistemi üzerinde gereksiz yüke sebep olacak dizinlerin var olma ihtimali

gösterilmiştir. Diğer bir dezavantaj olarak ise bu tür bir aracın iyileştirici tarafından kullanılan mevcut strateji bilgisine sahip olduğu ve iyileştici üzerinde yapılacak herhangi bir değişiklik durumunda güncelliğini kaybetme ihtimalinin bulunduğu belirtilmiştir. İkinci kategorideki araçlarda ise tüm dizinlerin iyileştirici tarafından seçilmesinden dolayı seçilen dizinlerin, iş yükünü işlerken iyileştirici tarafından direkt kullanılması avantajının bulunduğu belirtilmiştir. Ancak bu yaklaşımda birçok olası aday dizin kümesinin veri tabanı sisteminin iyileştirici modülü tarafından değerlendirilmesi gerekmektedir. Dolayısıyla bu yaklaşımın bir dezavantaj olarak pek çok iyileştirici çağrısı gerektireceği belirtilmiştir. Bu da dizin önerme süresinin çok uzun olmasına ve önerme esnasında veri tabanı yönetim sistemi üzerindeki işlerin yavaşlamasına sebep olacağı bilgisi verilmiştir. Zaman vd. [3] yaptıkları çalışmada ise yukarıda belirtilen iki yaklaşım birleştirilerek dizin seçim probleminin büyük bir kısmını harici olarak ve dizin seçiminin ise iyileştirici tarafından yapılmasını sağlayacak bir araç geliştirilmiştir. Böylece, olası dizin kümesi belirlendikten sonra, son dizin kümesinin seçimi, iş yükündeki her bir sorgu için iyileştiricinin sadece bir kez çağrılması ile sağlanmaktadır. Yapılan deneyler sonucunda, bu çalışmada geliştirilen araç ile Microsoft SQL Server dizin seçim aracından daha iyi performans elde edildiği ve daha kısa sürede dizin önerildiği belirtilmiştir.

Pedrozo ve Vaz tarafından [7] otomatik dizin seçimi yapmak üzere yapılan başka bir çalışmada da dizin seçimi yapmak için iyileştiriciden yararlanan AISIO isminde bir araç geliştirilmiştir. Bu çalışmada, dizin seçim sürecine rehberlik etmesi için VTYS ile ilişkinlendirilecek bir sezgisel yöntem önerilmektedir. Önerilen sezgisel yöntem ile VTYS arasındaki köprü, AISIO aracılığıyla yapılmaktadır. AISIO ile en iyi dizin kümesi elde edilmeye çalışılmaktadır. Bu çalışmada aday dizinlerin belirlenmesi, sorgu ağacının optimizasyon sürecinden önce başlamaktadır. AISIO, veri tabanı yönetim sisteminden istatistiki bilgileri almakta ve sezgisel tabanlı işlemler yaparak, kullanılması muhtemel dizinleri numaralandırmaktadır. AISIO, kullanılan sezgisel yönteme göre, önce tek sütunlu aday dizinleri belirler, daha sonra bu dizinleri birleştirerek çoklu sütun içeren aday dizinleri belirler ve en sonunda tekli ve çoklu sütunlardan oluşan aday dizin kümesini elde eder. AISIO, daha sonra VTYS'nin kataloğunda varsayımsal olarak aday dizinleri oluşturur. Bir SQL ifadesi işlendiğinde iyileştirici yürütme planını kurgular ve eğer varsayımsal dizinler yürütme planında bulunursa, bu dizinlerin oluşturulması veri

tabanı yöneticisine tavsiye edilir. Ayrıca bu dizinlerin kullanılması durumundaki performans tahminleri de sunulur ve varsayımsal dizinlerin oluşturulup oluşturulmayacağı konusundaki karar veri tabanı yöneticisine bırakılır.

AISIO, PostgreSQL VTYS'ye entegre edilmiş ve AISIO aracının kodları PostgreSQL VTYS ile birlikte derlenmiştir. Bunda, AISIO aracının ve PostgreSQL sisteminin C programlama dili ile geliştirilmesi etkili olmuştur. AISO aracının dizin seçiminde kullanılması sonucunda ortaya çıkan performans kazanımlarını ölçmek için bir durum çalışması yapılmıştır. Transaction Processing Performance Council (TPC) tarafından üretilen sorgulardan oluşan bir iş yükü PostgreSQL sistemine gönderilmiştir. Değerlendirmede simüle edilen ortam, bir toptan satış şirketi için hareket işleme süreçlerinden oluşmaktadır. Test senaryosuna göre, şirket depoları coğrafik olarak dağınıktır ve satışlar bu depolardan yapılmaktadır. AISIO'nun sonuçlarını karşılaştırmak için Open Source Development Lab(OSDL) tarafından oluşturulan DBT-2 isimli araç kullanılmıştır. Dakikada tamamlanan hareket sayısı performans ölçütü olarak ele alınmıştır. Sistem, dizinlerin hiç kullanılmadığı, DBT-2 aracının ürettiği dizinlerin kullanıldığı ve AISIO'nun ürettiği dizinlerin kullanıldığı üç senaryo ile test edilmiştir. Beklendiği gibi dizinlerin hiç kullanılmadığı senaryoda performans açısından en kötü sonuçların elde edildiği belirtilmiştir. AISIO aracının ürettiği dizinler ile ise en iyi performansın elde edildiği belirtilmiştir.

Schnaitter vd. tarafından [8] yapılan bir çalışmada ise statik bir iş yükü için dizin oluşturmaya eleştirel olarak bakılmakta ve değişen iş yükleri karşısında çevrim dışı bir yaklaşımın üreteceği sonuçların yetersiz olacağı belirtilmektedir. Bunun için bu çalışmada, COLT (Continuous On-Line Tuning) isminde dizinlerin otomatik seçimini destekleyen çevrimiçi ayar aracı geliştirilmiştir. COLT, sorgulardan oluşan bir akış için depolama alanı kısıtına uyan dizinleri dinamik olarak seçmekte ve iş yükü için sorgu işlemenin performansını iyileştirmektedir. COLT, sürekli olarak iş yüklerini izlemekte ve performansta en iyi kazanımı sağlamak için belirli aralıklarla dizin kümesi üzerinde ayarlamalar yapmaktadır.

1.2 **Tezin Amacı**

Sisteme yeni dizinlerin seçilerek eklenmesi ve mevcut dizinlerin sistemden kaldırılması veri tabanı yöneticisinin sorumluluğundadır. Veri tabanı yöneticisi bu işlemleri yaparken iş yükü, veri tabanı tablo yapısı ve VTYS'nin sorgu iyileştiricisinin kullandığı yaklaşım hakkında bilgiye sahip olmalıdır. Ayrıca, NP-tam olan dizin seçim problemi karşısında veri tabanı yöneticisinin, performansı artıracak en iyi dizin kümesini seçmesinin ve performansı muhafaza etmek için gerekli ayarlamaları yapmasının zor olduğu aşikârdır.

Bu tez kapsamında, sorgulardan oluşan bir iş yükü karşısında, performansı artırmaya yönelik, otomatik olarak dizin seçimi yapmaya yarayan bir araç geliştirilmiştir. Geliştirilen otomatik dizin seçim aracı, bir iş yükü için uygun dizin setini, kümeleme teknikleri ile seçen, sistemin iyileştiricisinden bağımsız harici bir araçtır. Böylece, veri tabanı yöneticisinin sorumluluğunun azaltılması, bir iş yükünü oluşturan sorguların cevaplanması için gereken toplam zamanın ve toplam disk erişim sayısının azaltılması hedeflenmektedir.

1.3 **Hipotez**

Bu çalışmada sorgulardan oluşan bir iş yükü işlenirken gereken toplam disk erişim sayısı ve toplam cevap süresini en aza indirgemek amacıyla en uygun dizinler seçilmeye çalışılacaktır. Dizin seçimi yapılırken de kümeleme tekniğinden yararlanılacaktır. Dizin seçiminde izlenecek temel fikir ise iş yükünde yer alan sorguları benzerliklerine göre gruplamak ve her bir grupta yer alan sorgularda ortak olarak ve sıklıkla bulunan nitelikleri belirlemektir. Çünkü bu niteliklerin üzerinde dizin oluşturulması durumunda, oluşturulan dizinlerin birçok sorguya katkı sağlayacağı düşünülmektedir. Böylece bir iş yükü için uygun bir dizin kümesi ve dolayısıyla performans artışı elde edilmeye çalışılmaktadır.

Sorgulardan oluşan bir iş yükü için uygun dizin önerileri elde etmek için kmeans ve kmedoids algoritmaları kullanılmaktadır. Elde edilen dizin önerileri gerçeklendikten sonra, iş yükünü oluşturan sorguların ardışık olarak ve eş zamanlı olarak çalıştırılmasında kullanılmakta ve ortaya çıkan performans değerleri izlenmektedir. Böylece, küme

sayısının ve kümeleme algoritmasının uygun dizin seçimi üzerindeki karşılaştırmaları yapılabilmektedir.

Önerilen hipotez kurgulanırken Zaman vd. tarafından [3] kümeleme tekniği kullanılarak yapılan otomatik dizin seçimi çalışmasından ilham alınmıştır. Ancak bu tez çalışmasında bazı farklılıklar bulunmaktadır. Bu farklılıklar, hipotezin detaylı anlatımı ve küçük bir örnek üzerinden yürütülmesi "Bölüm 4 Dizin Seçim Tekniği" başlığı altında verilmektedir.

BÖLÜM 2

FİZİKSEL VERİ TABANI TASARIMI VE AYARI

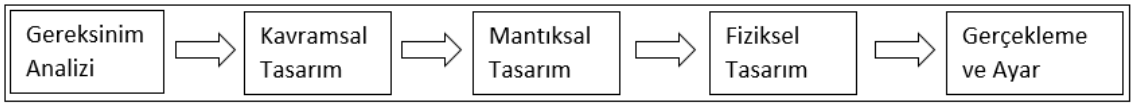
Veri tabanı teknolojisi bağlamında veri, depolanan bilgilerdir. Veri, bir telefon numarası, bir kişinin ismi, adres, kimlik numarası olabilir. Bilgisayar ortamında düzenli bir şekilde saklanan, gerektiğinde erişilebilen, yönetilebilen ve güncelenebilen veriler topluluğuna ise veri tabanı denir. Veri tabanları, veri tabanı yönetim sistemi (VTYS) adı verilen yazılımlar ile oluşturulur ve devamlılığı sağlanır. Bir VTYS, kullanıcılara verilerin nasıl depolandığına veya işlemlerin nasıl gerçekleştirildiğine dair ayrıntıların çoğunu soyutlayan kavramsal bir arayüz sunar. Bu kavramsal temsili sağlamak için kullanılan veri soyutlamasına veri modeli denir. Bir veri modeli çoğu kullanıcı için bilgisayar depolama kavramlarından daha kolay anlaşılabilen nesneler, nesnelerin özellikleri, birbirileriyle olan ilişkileri gibi mantıksal kavramları kullanır. Bu nedenle veri modeli, çoğu uzman olmayan veri tabanı kullanıcıları için gerekli olmayan depolama ve uygulama ayrıntılarını gizler [9].

Günümüzde çoğu VTYS'de, ilişkisel veri modeli kullanılmaktadır. İlişkisel model, iki boyutlu tablo olarak adlandırılan ilişkileri esas alır. İlişkinin sütunları nitelik olarak adlandırılır. İlişkinin ismi ve nitelik kümesi ilişkinin şemasıdır. İlişkisel modelin yanında literatürde geçen diğer veri modelleri sıradüzensel model, ağ modeli, nesne-ilişkisel model ve son yıllarda kullanımı artan ilişkisel olmayan NoSQL veri modelidir [10].

2.1 **Veri Tabanı Tasarım Aşamaları**

Veri tabanı tasarımının amaçlarından bazıları kullanıcı veya uygulamaların gereksinimlerini belirlemek, bilginin doğal ve anlaşılır şekilde yapılandırılmasını

sağlamak, işlem süresi ve depolama alanı gibi performans hedeflerini belirlemektir. Genel veri tabanı tasarımı ve gerçekleştirimi aşamaları için 5 adımdan bahsedilebilir. Bu aşamalarda veri tabanının veri içeriği, yapısı ve kısıtları belirlenir. Şekil 2.1'de bu aşamalar gösterilmiştir. Bu aşamalar, tam olarak ardışık bir sırada ilerlemez, birçok durumda önceki bir aşamadan itibaren tasarımı değiştirmek zorunda kalınabilir. Bu geri besleme aşamaları, bir aşamanın kendi içinde veya aşamalar arasında olabilir [9].



Şekil 2. 1 Veri tabanı tasarım aşamaları

Gereksinim analizi: Bu aşamada, veriyi üreten ve kullanan kişilerle görüşülüp bir şartname hazırlayarak veri tabanı gereksinimleri belirlenir. Bu şartname, işlenecek olan veriyi, veri ilişkilerini ve veri tabanı gerçeklemesinin yapılacağı yazılım platformunu içerir [11].

Kavramsal tasarım: Bu aşamanın amacı VTYS'den bağımsız olarak kavramsal bir şema oluşturmaktır. Bu şema, veri ve veri ilişkileri gösterir ve bu aşamada genellikle varlık-ilişki modeli (ER) veya birleşik modelleme dili (UML) kullanılır [12].

Mantıksal tasarım: Bu aşamada kavramsal model, seçilmiş olan VTYS' nin kullanabileceği veri modeline dönüştürülür. Bu aşamaya VTYS seçildikden sonra başlanır. Kavramsal model normalize edilmiş ilişkilere ve tablolara dönüştürülmüş olur [9].

Fiziksel tasarım: Bu aşama, veri tabloları tanımlandıktan ve normalize edildikten sonra başlar. Verimli bir şekilde veri depolama ve veriye erişim mekanizmalarının oluşturulması üzerine odaklanılır. Fiziksel tasarımın amacı veri tabanının performansını maksimize etmektir. Bu aşama, genellikle veri tabanı yöneticisi pozisyonunda çalışan kişilerin çalışma alanıdır. Fiziksel dosya depolama yapılarının, kayıt yerleşiminin, dizin seçiminin tasarımı bu aşamada yapılır [9], [11].

Gerçekleme ve ayar: Bu aşamada veri tabanı ve uygulama programları gerçeklenir, test edilir ve sonunda hizmet için dağıtıma açılır. Çeşitli işlemler ve uygulamalar tek tek veya birbirleriyle bağlantılı olarak test edilir. Böylece, genellikle fiziksel tasarım değişiklikleri, dizinleme, yeniden düzenleme ve farklı veri yerleşimi fırsatları tasarımcıya sunulmuş olur. Bu faaliyet, veri tabanını ayarlama olarak adlandırılır. Ayarlama işlemi, performans

sorunları tespit edildiğinde bir veritabanının yaşam döngüsü için devam eden sistem bakımının bir parçasıdır [9].

Bu tez çalışması boyunca sadece fiziksel tasarım aşamasına odaklanılacaktır.

2.2 **Dizinler**

Sorgu verimliliğini etkileyen faktörlerden birisi kayıtların dosya içinde nasıl düzenlendiğidir. Kayıtları dosya içinde düzenlemek için birçok yöntem vardır. Örneğin bunlardan bazıları sıralama ve adrese dayalı sıralama olarak adlandırılan yöntemlerdir. Bu yöntemlerin her birinin farklı uygulanışları ve karakteristikleri vardır fakat hepsinin ortak özelliği aranılan kayıtlara, bütün dosyayı taramadan hızlı erişme imkanı tanımalarıdır [13].

Veri tabanı tasarımcısı, çoğunlukla dosyanın nasıl düzenlendiğiyle ilgili değildir daha çok dosyanın düzenlenip düzenlenmediği ve düzenlendiyse hangi nitelikler üzerinde düzenlendiği ile ilgilidir. Örneğin, 6.1.2' de verilen üniversite veri tabanında yer alan student tablosunun niteliklerinden birisi, öğrenci numarasını belirten sid niteliğidir. Student tablosu, sid değerine göre sıralanmışsa, sid değeri verilen bir öğrencinin ismi kolaylıkla bulunabilir. Bu durumda, sid değeri biliniyorsa, student tablosu kullanışlı olur. Ancak, aynı tablodan öğrenci ismi niteliği olan sname değeri bilinen bir öğrencinin sid değeri öğrenilmek istenirse, tüm tablodaki kayıtların taranması gerekir. Çünkü genel bir ilke olarak, bir tablo düzenlendiğinde ancak bir nitelik veya nitelik grubuna bağlı olarak sadece bir yöntemle düzenlenebilir. Eğer hem sid ve hem de sname niteliklerinin değerlerine hızlı bir şekilde ulaşılmak istenirse, student tablosunun farklı niteliklere göre sıralanmış kopyalarına ihtiyaç duyulur. Bu yaklaşım, bir veri tabanı tablosuna uygulanabilir. Bunun için farklı niteliklere göre sıralanmış kopya dosyalar oluşturulmalıdır. Ancak bu yöntem beraberinde bazı problemler getirir. Bu problemler:

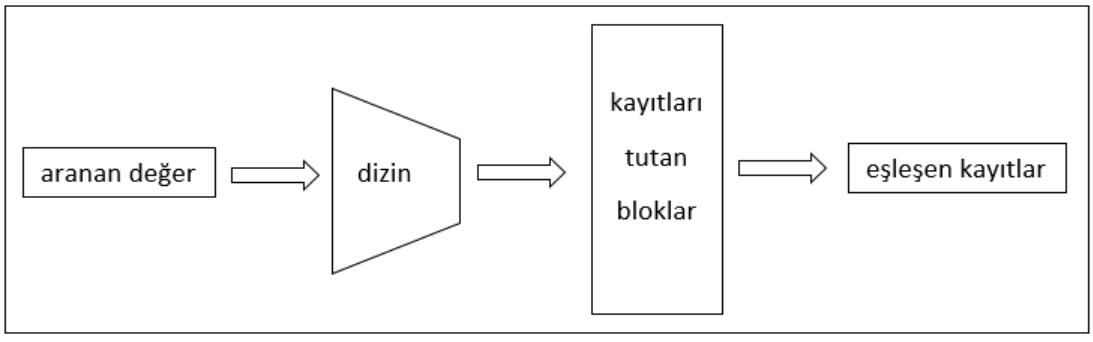
- Aşırı alan kullanımı: Her bir kopya dosya, ana dosya ile aynı boyuttadır dolayısıyla çok fazla alan kullanımına sebep olur,
- Senkronizasyon problemi: Kullanıcı bir tabloda değişiklik yaptığında tüm kopya tablolarda aynı değişikliğin yapılması gerekir, bu da zaman kaybına yol açar,

- Tutarsızlık tehlikesi: Kopya tabloların senkronizasyonunun sağlanması sırasında çıkacak bir hata, yapılan sorgularda yanlış sonuçlar elde edilmesine neden olur.

Bu problemlerin çözümü, kopya dosya oluşturmak yerine dizin kullanılmasıdır. Dizinlerin kapladığı alan, kopya dosyalara göre daha küçük olur, dizinlerin bakımını ve yedekliliğini yönetmek daha kolaydır [13].

Veri tabanları, kayıt dosyaları şeklinde manyetik diskler üzerinde fiziksel olarak depolanırlar. Depolanan kayıtlara verimli bir şekilde ulaşmak için kullanılan çeşitli algoritmalar vardır. Bu algoritmaların bazıları için dizin olarak adlandırılan yardımcı veri yapıları gerekir [9]. Dizin bir tablonun A niteliği üzerindeki, A niteliğinin belli bir değere sahip olduğu kayıtları bulmaya yarayan veri yapısıdır [14]. Bir tablonun sadece bir niteliği üzerinde dizin oluşturulabildiği gibi aynı zamanda aynı tablo üzerinde farklı nitelikler üzerinde de dizin oluşturulabilir. Buna göre bir tablo üzerinde birden fazla dizin oluşturulabilir. Ayrıca birden çok nitelik üzerinde de bir dizin oluşturulabilir. Çeşitli dizinler oluşturmak mümkündür, bunlardan her biri aramayı hızlandırmak için bir veri yapısı kullanır [9].

Dizinler tablo üzerinde arama koşullarına uyan kayıtlara hızlı bir şekilde ulaşılmasını sağlar [10]. Bir sorguda aranan kayıt veya kayıtları bulmak için yüklem veya karşılaştırma belirtimi ifadesinde yer alan ve üzerinde dizin oluşturulmuş niteliklerin değerleri dizinde aranır. Daha sonra dizinin işaret ettiği dosya bloklarında yer alan kayıtlara erişilir [9]. Şekil 2.2'de dizinlerin çalışma mekanizması gösterilmiştir.



Şekil 2. 2 Dizinlerin çalışma mekanizması

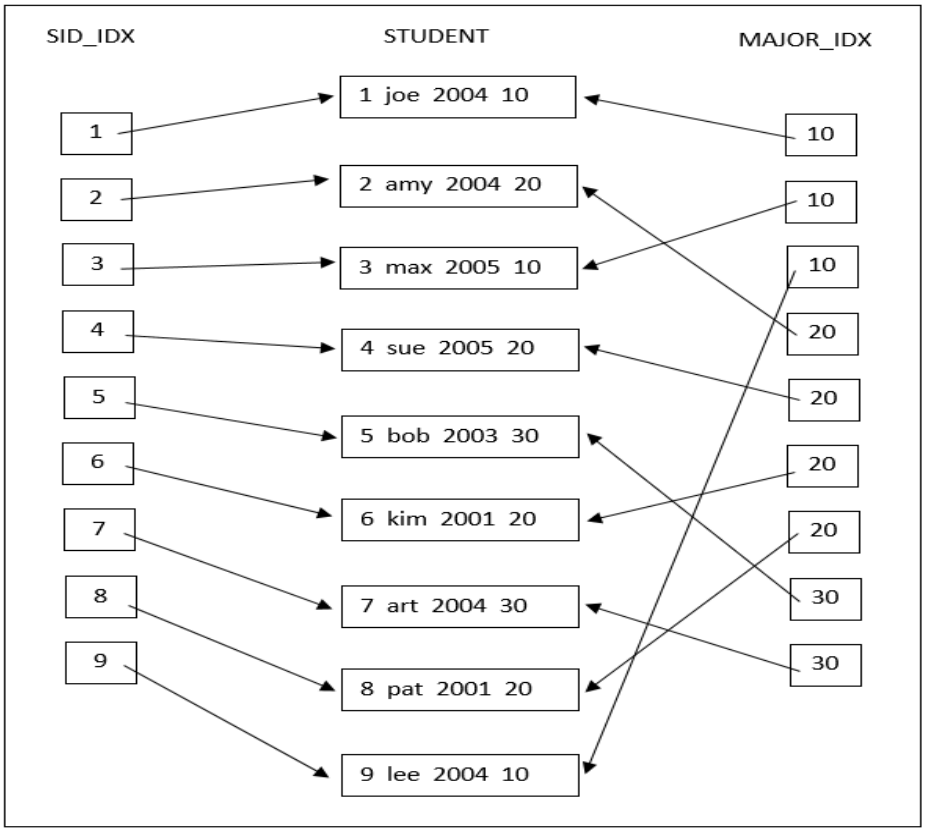
Dizinlerin oluşturulması SQL standartının bir parçası değildir. Ancak çoğu ticari veri tabanı sistemleri tarafından genel kabul gören şekli ile bir tablonun belli nitelikleri üzerinde dizin oluşturmak için genel bir sentaks vardır. Şekil 2.3'de, 6.1.2'de verilen

üniversite veri tabanında yer alan student tablosunun majorid ve sid nitelikleri üzerinde dizin oluşturulmasının örneği verilmiştir. sid niteliği üzerinde dizin oluştururken kullanılan "unique" anahtar kelimesi, sid niteliğinin student tablosunun anahtarı olduğunu ifade eder. Şekil 2.3'deki dizinler tek bir nitelik üzerinde oluşturulmaktadır. Bunun yanında istenirse birden çok nitelik üzerinde de dizin oluşturulabilir [13]. Üzerinde dizin oluşturulan nitelik veya nitelikler arama anahtarı olarak adlandırılır [14].

```
create index MAJOR_IDX on STUDENT(majorid)
create unique index SID _IDX on STUDENT(sid)
```

Şekil 2. 3 Dizin oluşturma komutları örneği

Kavramsal olarak bir dizin, aslında dizin kayıtlarından oluşan bir dosyadır. Her bir dizin kaydı, üzerinde dizin oluşturulmuş niteliğin bir değerini veya niteliklerin değerlerini içermektedir. Bu değer arama anahtarıdır. Bunun yanında her bir dizin kaydı, ana dosya kayıtlarına işaret eden bir kayıt işaretçisi değeri içermektedir. Şekil 2.4'de dizinlerin student tablosu üzerindeki kayıtlara nasıl eriştiğini tasvir eden bir örnek gösterilmiştir [13].



Şekil 2. 4 Student tablosu için SID_IDX ve MAJOR_IDX dizinleri

Şekil 2.4'de SID_IDX dizininin her bir kaydı sid ve kayıt işaretçisi çiftini ve benzer şekilde MAJOR_IDX dizininin de her bir kaydı da majorid ve kayıt işaretçisi değerlerini içerirler. Bir dizin kaydında yer alan kayıt işaretçisi değeri, student tablosunda karşılık düşen kayda işaret eder, böylece ana dosyaya göre daha küçük olan dizinde hızlı bir arama ile direkt olarak student tablosunun kayıtlarına erişilmiş olur [13].

Veri tabanı tablosu çok büyük olduğunda tabloda aranan şartlara uyan kayıtları bulmak için bütün satırları taramanın maliyeti çok büyük olabilir [14]. Örneğin üniversite veri tabanında yer alan student tablosunun 800 bloğu kapsayan 40000 kaydı bulunmaktadır. Bu tablo üzerinde, öğrenci numarası bilinen bir öğrencinin bölümünü öğrenmeye yarayan sorgu, Şekil 2.5'deki gibidir [13].

```
Select majorid from STUDENT where sid = 8
```

Şekil 2. 5 Öğrenci numarası 8 olan öğrencinin bölüm numarasını sorgulama

Veri tabanı sistemi, Şekil 2.5'deki sorguyu, Şekil 2.6'daki gibi ele alıp hiç dizin kullanmadan bütün student tablosunu tarayarak çalıştırabilir ve istediği sonuca erişir. Student tablosunun verileri 800 adet blok içinde yer almaktadır. Bu durumda, en kötü durum senaryosuna göre dizin kullanılmadığı durumda 800 adet disk bloğu erişimi gerekmektedir [13].

```
STUDENT tablosundaki her bir kayıt için:
    Eğer bu kaydın sid değeri 8 ise:
        Bu kaydın majorid değerini döndür
```

Şekil 2. 6 Bir sorgunun dizin kullanmadan yürütülmesi

Veri tabanı sistemi, yukarıdaki yönteme alternatif olarak, Şekil 2.5'deki sorguyu yürütmek için Şekil 2.7'deki gibi SID_IDX dizinini kullanabilir. SID_IDX dizini üzerinde arama anahtarı değerini bulmak için 2 disk bloğu ve dizinin işaret ettiği ana dosya bloğu için de 1 disk bloğu erişimi gerekir. Böylece toplam disk erişim maliyeti 3 olur ve dizin kullanılmayan senaryoya göre 267 kat daha az disk erişim sayısı ile istenilen sonuç elde edilmiş olur [13].

1. SID_IDX dizinini kullanarak sid değeri 8 olan dizin kaydını bul
2. Bulunan dizin kaydındaki kayıt işaretçisi değerini elde et
3. Direkt olarak kayıt işaretçisinin işaret ettiği student kaydına git
4. Gidilen kaydın majorid değerini döndür

Şekil 2. 7 Bir sorgunun dizin kullanılarak yürütülmesi

2.2.1 Dizin Türleri

Her bir dizin kaydı, veri dosyasının bir veya birden fazla kaydına erişmek için arama anahtarıyla birlikte yeterince bilgi içerir. Dizin dosyasındaki kayıtları depolamak için 3 alternatif yöntem vardır [10].

Birinci yöntem: Dizin kaydı, arama anahtarı ile birlikte veri tablosundaki gerçek veri kayıtlarını içerir.

İkinci yöntem: Dizin kaydı, arama anahtarı ve kayıt tanımlayıcısı çiftlerini (k, kayıt işaretçisi) içerir.

Üçüncü yöntem: Dizin kaydı, arama anahtarı ve kayıt tanımlayıcı listesi çiftlerini (k, kayıt işaretçisi listesi) içerir.

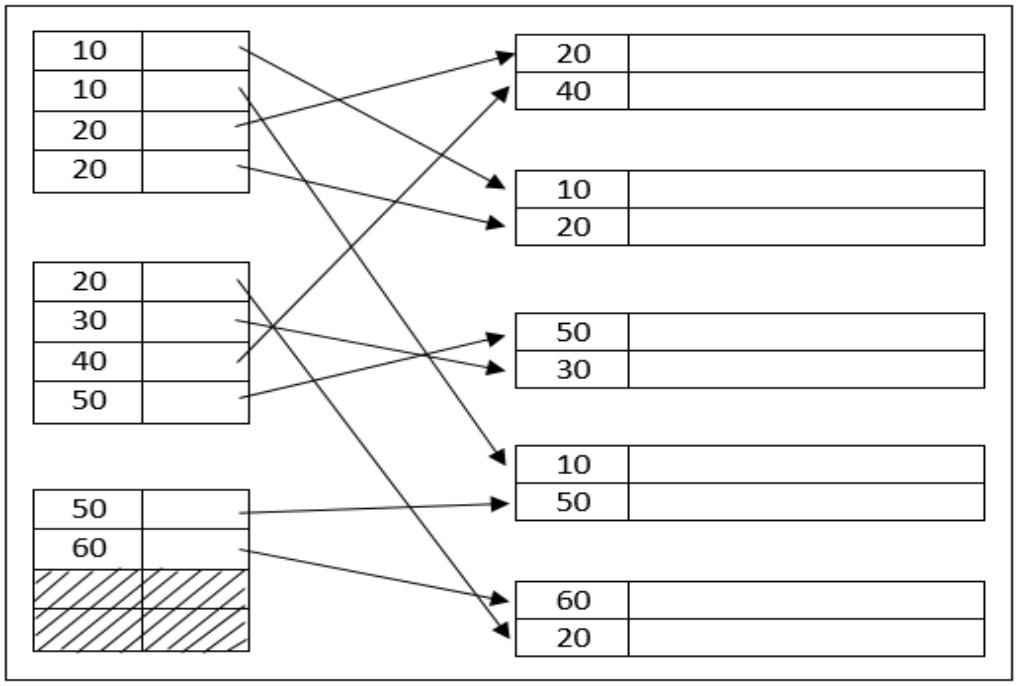
Dizin üretmek için birinci yöntem kullanılırsa, ayrıca veri kayıtlarını depolamaya gerek yoktur. Bu tür bir dizin yapısı sıralanmış bir dosya veya kümelenmiş dizin dosyası (clustered index) gibi özel bir dosya düzenlemesi olarak düşünülebilir. İkinci ve üçüncü yöntemlerde dizin kayıtları, üzerinde dizin oluşturulan dosyanın kayıtlarına işaret ederler ve dizin dosya yapısı, üzerinde dizin oluşturulan ana dosyadan bağımsızdır. Bir dosya üzerinde birden fazla dizin oluşturulmak isteniyorsa, örneğin 6.1.2'de verilen üniversite veri tabanında yer alan student tablosunun sid ve majorId nitelikleri üzerinde dizinler oluşturulmak istenirse, yalnızca birisi için birinci yöntem kullanılabilir. Çünkü aynı dosyanın birden fazla kez depolanmasından kaçınılmaktadır [10].

2.2.1.1 Birincil ve İkincil Dizinler

Biricik değerler alan nitelikler üzerinde oluşturulan dizin birincil dizin olarak adlandırılır. Bir dizin birincil dizin değilse ikincil dizin olarak adlandırılır. Bir dizin dosyasında yer alan

iki dizin kaydının arama anahtarlarının değeri aynı ise, dizin kayıtlarında yinelenme olduğu söylenebilir. Birincil dizin, yinelenme içermemeyi garanti eder çünkü ana dosyanın birincil anahtar değeri her kayıt için farklıdır. Ancak diğer nitelikler üzerinde oluşturulan dizinlerde yinelenme bulunabilir. İkincil dizinler, genelde yinelenme içerirler [15].

Veri dosyasındaki veriler, arama anahtarına göre sıralı olmamasına rağmen dizin kayıtları, arama anahtarına göre sıralıdır. İkincil dizinlerde bir dizin bloğunda yer alan işaretçiler, bir veya birden fazla ardışık veri bloğunu işaret etmek yerine birçok farklı konumdaki veri dosyası bloğunu işaret etmektedir. Bu yüzden bir veri dosyasında ikincil dizin birincil dizine göre çok daha fazla rastgele disk bloğu erişimine sebep olur. Bu da disk erişimi performansı noktasından dezavantaja sebep olur. Şekil 2.8'de ikincil dizin görselleştirilmeye çalışılmıştır. Bu şekilde, veri dosyası blokları iki kayıt içermekte, dizin blokları ise dört tane kayıt içermektedir [14].



Şekil 2. 8 İkincil dizin örneği

2.2.1.2 **Kümelenmiş ve Kümelenmemiş Dizinler**

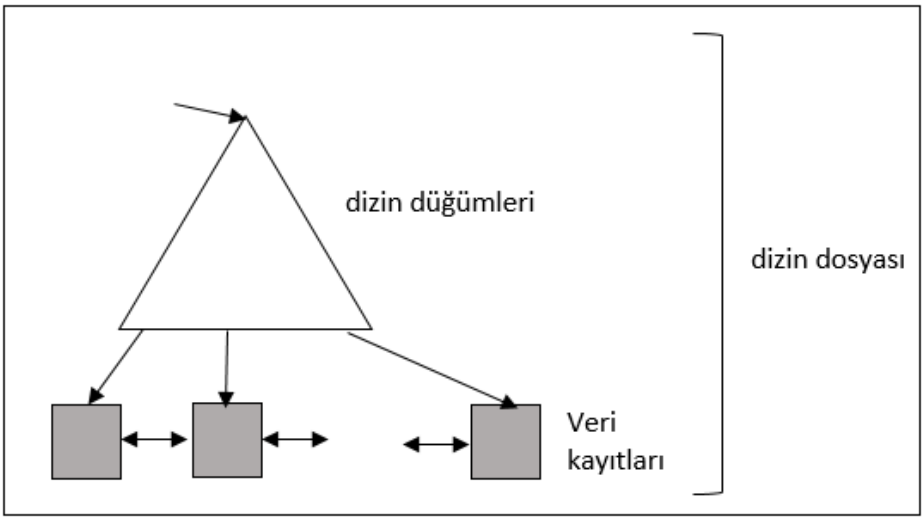
Bir tablodaki kayıtlar, bir niteliğe göre sıralanmışsa, bu nitelik, kümelenmiş nitelik ve tablo da kümelenmiş dosya olarak adlandırılır. Bu, genel olarak kümeli düzenleme tanımıdır. Kümelenmiş bir dizin (clustered index) ise, dizin yapraklarında veri kayıtlarını

bir niteliğe göre sıralı olarak içermektedir. Böylece kümelenmiş bir dizin, aranan kayıtlara daha hızlı bir şekilde ulaşmaya imkan sağlar. Bir dizinin kayıtları, arama anahtarı ile birlikte ana dosyadaki gerçek veri kayıtlarını içeriyorsa, bu dizin kümelenmiş dizindir [15].

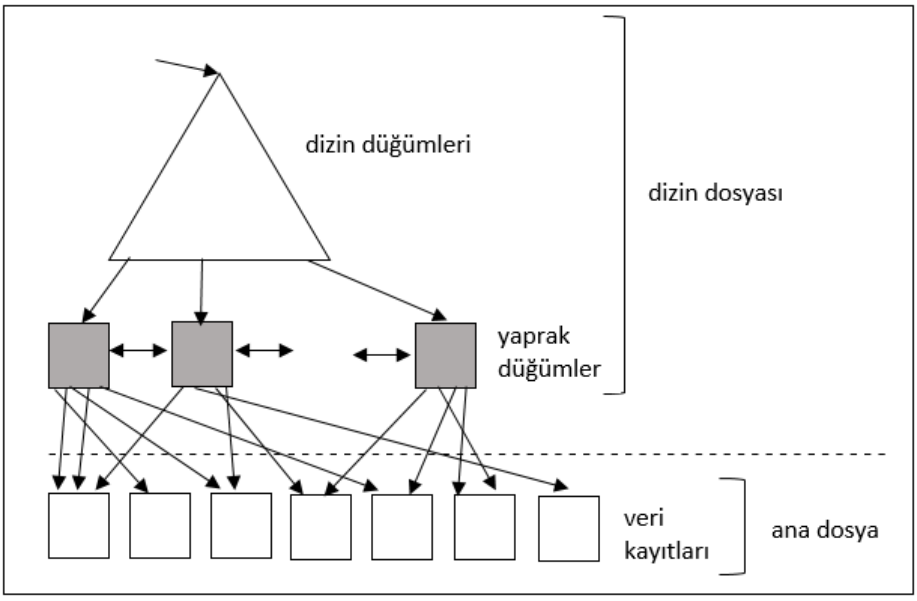
Bir veri dosyası, en fazla bir arama anahtarı üzerinden kümelenebilir, bu da bir veri dosyasında en fazla bir kümelenmiş dizine sahip olunabileceği anlamına gelir. Bir dizin kümelenmemişse, bu dizin kümelenmemiş dizin (unclustered index) olarak adlandırılır. Veri dosyası üzerinde birden çok kümelenmemiş dizin olabilir [10].

Kümelenmiş dizin kayıtları arama anahtarına göre sıralıdır ve kayıtlar ağaç yapısına göre düzenlenmiştir [10]. Yaprak düğümlerinde gerçek veri kayıtları bulunur. Bu yüzden kümelenmiş dizin, ana dosyayı tam olarak temsil etmektedir [2].

Kümelenmiş ve kümelenmemiş dizinler sırasıyla Şekil 2.9 ve Şekil 2.10'da görselleştirilmiştir [15].



Şekil 2. 9 Kümelenmiş dizin



Şekil 2. 10 Kümelenmemiş dizin

Bir aralık sorgusunu cevaplamanın maliyeti kümelenmiş dizin kullanılıp kullanılmadığına göre büyük ölçüde değişebilir. Çünkü kümelenmiş dizin varsa Şekil 2.9'daki gibi yaprak düğümleri arasında sıralı disk erişimi ile dolaşılabilir ve istenilen sonuç birkaç blok erişimi ile elde edilebilir [15]. Çünkü kümelenmiş dizinlerde veri tablosu arama anahtarı niteliğine göre sıralanmıştır ve dizinin yaprak düğümlerinde gerçek veri kayıtları bulunur [2]. Dizin kümelenmemişse Şekil 2.10'daki gibi dizin kayıtlarında yer alan kayıt işaretçileri farklı bloklara işaret edebilirler ve sonuç olarak çok sayıda rastgele disk bloğu erişimi yapılmış olur [15].

2.2.1.3 **Çoklu Sütun İçeren Dizinler**

Birçok sorgu ve güncelleme işlemleri birden çok nitelik içerir. Bir tablonun belirli bir nitelik kombinasyonuna sıklıkla erişiliyorsa, bu niteliklerin bir kombinasyonu olan bir arama anahtarına verimli erişim sağlayan bir erişim yapısı oluşturmak, performans açısından avantajlı olur [9]. Buna göre, bir dizin için arama anahtarı birden çok nitelik içerebilir, bu nitelikler kompozit arama anahtarı olarak adlandırılır [15].

Örneğin, 6.1.2' de verilen üniversite veri tabanında yer alan student tablosu ele alınsın. Student tablosunda sid(öğrenci numarası) anahtar niteliği, sname(öğrenci ismi), gradyear(mezuniyet yılı), majorid(bölüm numarası) nitelikleri bulunmaktadır. Student tablosunun sid niteliğine göre sıralanmış olduğu varsayılsın. Bu tablo üzerinde bölüm

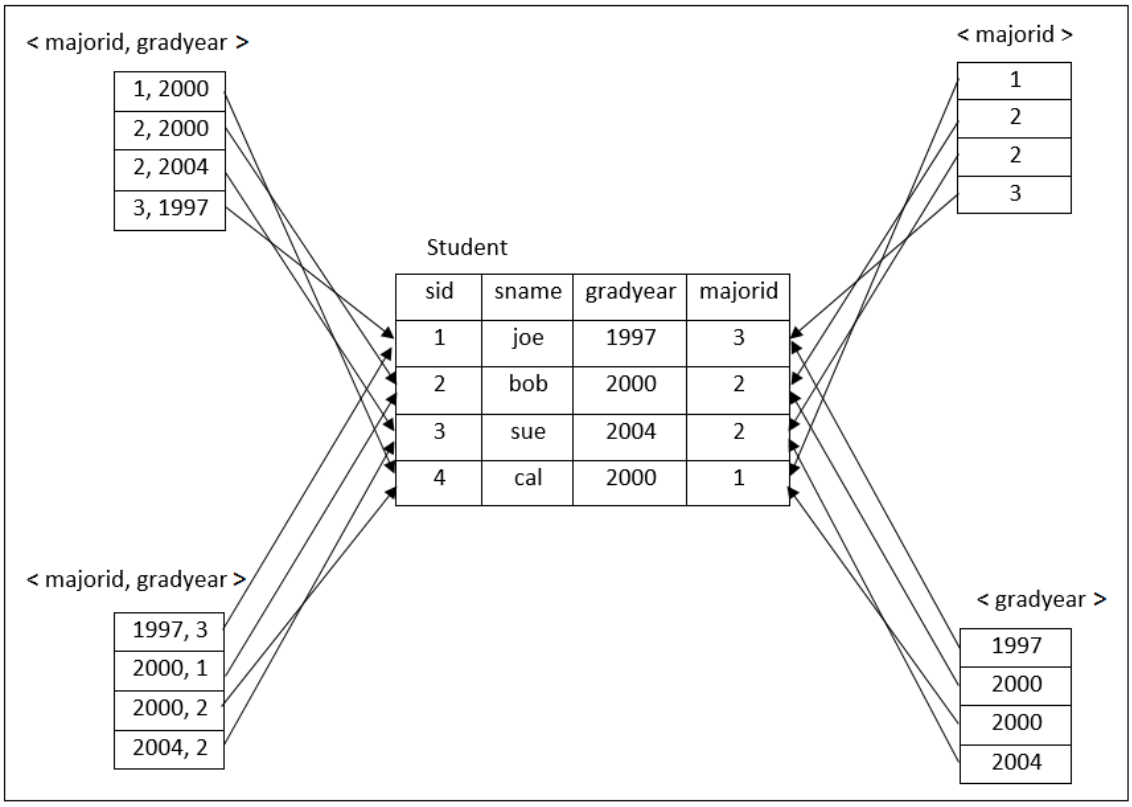
numarası 2 olan ve 2000 yılında mezun olan öğrencileri sorgulayan bir nokta sorgusu aşağıdaki gibi olsun.

SELECT sname FROM STUDENT WHERE majorid=2 AND gradyear=2000

Bu sorguda geçen iki şartı sağlayan birçok student kaydı olduğu varsayılsın. Aranan kayıtlara verimli bir şekilde erişmek için olası yöntemler:

- Gradyear niteliği üzerinde bir dizin olduğu varsayılırsa, gradyear değeri 2000 olan kayıtlara hızlı bir şekilde erişip daha sonra ortaya çıkan kayıtlardan majorid niteliğinin değeri 2 olanlar seçilebilir,
- Majorid niteliği üzerinde bir dizin olduğu varsayılırsa, majorid değeri 2 olan kayıtlara hızlı bir şekilde erişip daha sonra ortaya çıkan kayıtlardan gradyear niteliğinin değeri 2000 olanlar seçilebilir,
- Hem gradyear hem de majorid niteliği üzerinde dizin olduğu varsayılırsa, iki dizini de ayrı ayrı kullanıp ortaya çıkan sonuçların kesişim kümesini alarak istenilen sonuca ulaşılabilir.

Bu ihtimallerin hepsi aynı ve doğru sonuca götürür. Ancak her bir koşulu da sağlayan kayıtlar çok fazlaysa ve kesişim kümesinin eleman sayısı çok az ise yukarıdaki yöntemler verimsiz olur [9]. Çözüm olarak çoklu nitelik içeren <gradyear, majorid> veya <majorid, gradyear> kompozit nitelikleri üzerinde dizin oluşturulabilir. Bu ihtimaller Şekil 2.11'de görselleştirilmiştir.



Şekil 2. 11 Çoklu sütun içeren temsili dizinler

Kompozit arama anahtarı içeren bir dizinde arama anahtarını oluşturan niteliklerin sıralaması bazen büyük farklara sebep olabilir. Örneğin aşağıdaki gibi bir sorgu olsun.

SELECT sname

FROM STUDENT

WHERE majorid=2 AND gradyear BETWEEN 2010 AND 2018

Bu sorgu için {majorid, gradyear} nitelikleri üzerinde oluşturulmuş kompozit, kümelenmiş bir B+ tree dizini verimli bir şekilde sonuç elde etmeye yarar. Çünkü böyle bir dizin ile kayıtlar önce majorid niteliğine göre sıralanır, ardından ise gradyear niteliğine göre sıralanır. Böylece majorid=2 değerine sahip olan tüm kayıtlar biraraya toplanır. Diğer taraftan {gradyear, majorid} nitelikleri üzerinde oluşturulmuş kompozit, kümelenmiş bir B+ tree dizini aynı performansı göstermez. Çünkü böyle bir dizinde kayıtlar önce gradyear niteliğine göre ardından ise majorid niteliğene göre sıralanır. Bu durumda, majorid=2 değerine sahip birden fazla kayıt olabilir ve bu kayıtlar birbirinden çok uzakta olabilir. Böyle bir dizin, gradyear için olan aralık değerlerini kolayca bulmayı sağlarken majorid niteliklerini bulmada yardımcı olmaz [10].

2.2.1.4 **Kaplayan Dizin**

Kaplayan dizin, bir sorgu için gerekli olan tüm nitelikleri içerir. Bir sorgu için T tablosunun dizinlenebilir nitelikler kümesi I_T olsun ve R_T de dizinlenemeyen nitelikler kümesi olsun. K, $I \subseteq I_T$ ifadesinin bir permütasyonu, $R_1 \subseteq (I_T - I)$ ve $R_2 \in \{\emptyset, R_T\}$ olmak üzere T tablosu için aday dizinler kümesi $T(K|R_1 \cup R_2)$ olur. Açıklamak gerekirse, dizinlenebilir nitelik kümesi I_T' nin her bir alt kümesinin tüm permütasyonları arama anahtarı olarak düşünülür. Arama anahtarı olarak kullanılmayan dizinlenebilir niteliklerin alt kümeleri de arama anahtarı olmayacak şekilde dizine eklenebilir. Opsiyonel olarak, T tablosunun dizinlenemeyen nitelikleri olan R_T' nin elemanları da arama anahtarı olmayacak şekilde dizine eklenebilir [2].

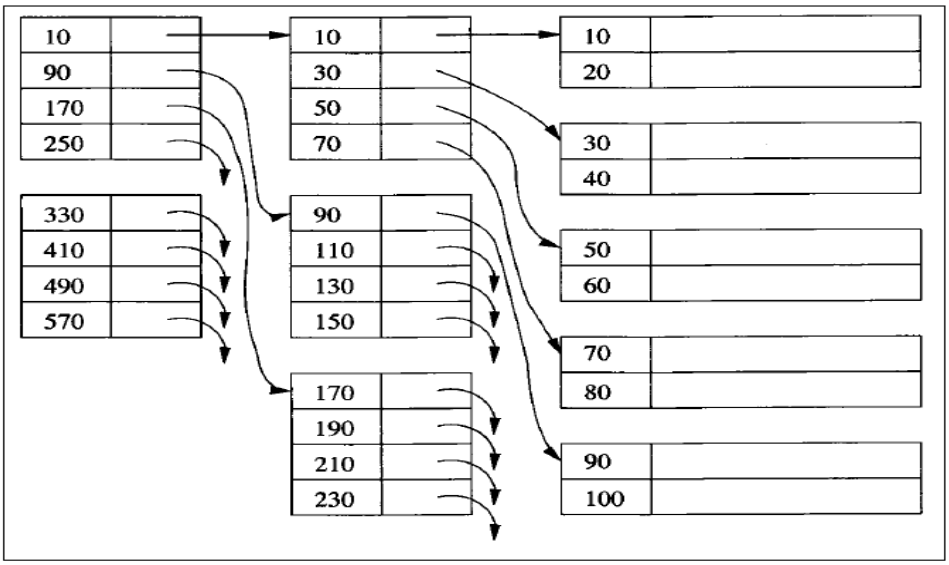
Örneğin, bir T(a, b, c, d) tablosu için aşağıdaki sorgu ele alınırsa, bu sorgu için I(a,c|b) şeklinde bir kaplayan dizin oluşturulabilir. Bu dizinde a ve c nitelikleri birlikte arama anahtarıdır.

SELECT a, b FROM T WHERE a=3 ORDER BY c

Kaplayan dizinler performansı artırmak için kullanışlıdırlar ancak bu tür dizinler büyük olabilir ve sadece birkaç sorgu için kullanışlı olabilirler. Eğer bir sorgu kritik ve sık sık yürütülüyorsa bu sorgu için kaplayan dizin oluşturmak avantajlıdır [2].

2.2.1.5 **Çok Seviyeli Dizin**

Bir dizin, birçok bloktan oluşabilir. Bir dizin kaydı arandığında ikili arama bile kullanılsa, diske çok sayıda erişim gerekebilir. Bir dizin üzerinde başka bir dizin oluşturularak birinci seviyedeki dizin kullanımı daha verimli hale getirilebilir. Aynı fikir ikinci seviye dizin üzerinde de üçüncü seviye dizin ve sürekli bir önceki dizin üzerinde yeni dizin oluşturulacak şekilde ilerletilebilir. Şekil 2.12'de, iki seviyeli dizin örneği gösterilmiştir [14].



Şekil 2. 12 Çok seviye dizin örneği

2.2.2 Dizin Veri Yapıları

Dizin, bir sorgu sonucunda elde edilecek kayıtları, veri tabanı sisteminin bütün kayıtlara bakmadan elde etmesini sağlar. Dizinler 2 esas üzerine gerçekleştirilir: ağaç yapıları, adrese dayalı dizinleme.

2.2.2.1 B-Tree Dizin Yapısı

Sorguların hızlandırılmasında bir ya da iki seviyeye sahip dizinler genellikle çok yardımcı olurken, ticari sistemlerde yaygın olarak kullanılan ve B-tree olarak adlandırılan daha genel bir yapı vardır [14]. B-tree dengeli bir ağaç yapısıdır dolayısıyla kökten yaprak düğümlere giden bütün yollar eşit uzaklıktadır. Genel olarak bir B-tree ağacı kök, ara katman ve yapraklar olmak üzere 3 katmandan oluşur ancak ara katman sayısı değişebilir [14].

Bu yapı, bir veri dosyasındaki kayıtları aramak için dinamik çok seviyeli dizin olarak kullanılır [9]. Bu dizin yapısında kayıtlar arama anahtarına göre sıralıdır [13].

Ara katman, sıfırdan başlayan ve B-tree ağacı içinde köke doğru artan numaralara sahip seviyelerden oluşur. Ara katman kayıtları, arama anahtarı değeri ve blok numarasından oluşur. Ara katmanda yer alan ilk seviye olan seviye 0, her bir dizin bloğunun ilk kaydının arama anahtarı değerini ve o kaydın blok numarasını içerir. Ara katmanda yer alan kayıtlar, arama anahtarına göre sıralı olduğundan, bir dizin bloğunda yer alan kayıtların

değer aralığı, rehber dizinin komşu kayıtlarının arama anahtarlarına bakılarak belirlenebilir [13].

B-tree üzerinde yapılacak bir arama için ara katmanın her seviyesinde bir blok ve buna ek olarak yaprak seviyesinde bir dizin bloğuna erişilir. Bu durumda arama maliyeti ara katman seviyesi sayısı artı birdir. Örneğin 5.2.1' de verilen üniverite veri tabanında yer alan student tablosunun "sname" niteliği üzerinde bir B-tree dizini bulunduğu ve bir disk bloğu büyüklüğünün 4 KB olduğu varsayılsın. Her bir dizin kaydı 22 bayt olursa bir bloğa 178 dizin kaydı sığar. Her bir ara katman kaydı 18 bayt olursa bir bloğa 227 adet ara katman kaydı sığar [13]. Bu durumda:

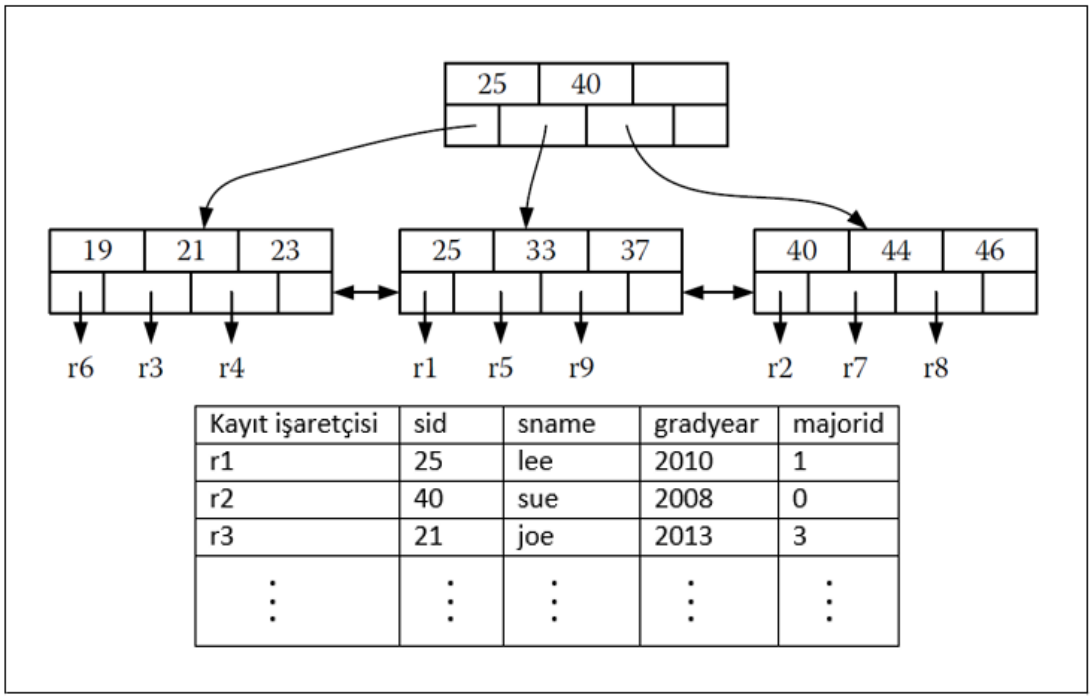
Ara katmanın 0. seviyesi üzerinde 227 x 178 = 40 406 adet dizin kaydı üzerinde 2 blok ile erişimi ile arama yapılabilir.

Ara katmanın 1. seviyesi üzerinde 227 x 227 x 178 = 9 172 162 adet dizin kaydı üzerinde 3 blok ile erişimi ile arama yapılabilir.

Ara katmanın 2. seviyesi üzerinde 227 x 227 x 227 x 178 = 2 082 080 774 adet dizin kaydı üzerinde 4 blok ile erişimi ile arama yapılabilir [13].

Bu örnekten de görüldüğü gibi B-tree dizinleri son derece verimlidir.

Çok seviyeli dizin oluşturma uygulamalarının çoğu B+-tree olarak adlandırılan B-tree yapısının farklı bir türü olan veri yapısını kullanır. Bir B-tree ağacının her seviyesindeki düğümlerinde arama anahtarı ve veri dosyası kayıtlarına işeret eden işaretçiler görülebilir. B+-tree ağacında ise veri dosyası kayıtlarına işeret eden işaretçiler sadece yaprak seviyesinde saklanır. Bu nedenle yaprak düğümlerin yapısı iç düğümlerin yapısından farklıdır. Ayrıca, yaprak düğümleri birbirine bağlıdır, bu sayede komşu yaprak düğümleri üzerinde dolaşabilmeye imkan tanır [15]. Bu özelliği ile bir B+-tree ağacı, nokta sorgular için sağladığı faydanın yanında aralık sorgular için de çok kullanışlı olur.



Kayıt işaretçisi	sid	sname	gradyear	majorid
r1	25	lee	2010	1
r2	40	sue	2008	0
r3	21	joe	2013	3
⋮	⋮	⋮	⋮	⋮

Şekil 2. 13 B+-tree dizini örneği

Şekil 2.13'de, bölüm 5.2.1' de verilen üniversite veri tabanında yer alan student tablosunun "sid" niteliği üzerinde oluşturulmuş bir B+-tree dizin örneği gösterilmiştir. Bu gösterim örnek amaçlı olduğu için her bir dizin düğümü, yüzlerce kayıt içeren gerçek uygulamaların aksine 3 tane kayıt içermektedir. Kök düğümü, student tablosunun bütün kayıtlarını işaret edecek şekilde kapsar ve student tablosunun kayıtlarını 25'den küçük olanlar, 25 ile 40 arasında olanlar ve 40'dan büyük olanlar olmak üzere 3 alt gruba ayırır. Yaprak seviyesinin en solundaki düğüm, student tablosunun "sid" niteliğinin değeri 19, 21 ve 23 olan kayıtlarına işaret eden işaretçiler bulundurur [2].

2.2.2.2 **Adrese Dayalı Dizin Yapısı**

Adrese dayalı dizin gerçeklemede temel fikir, arama anahtarı değerini kova numaralarına dönüştüren hash fonksiyonu kullanarak aranılan veri kaydına ulaşmaya çalışmaktır. Ağaç temelli dizin gerçeklemelerinin aksine adrese dayalı dizinler aralık sorgularını desteklemez. Bu yüzden birçok ticari veri tabanı sistemi sadece ağaç tabanlı dizin gerçeklemeyi tercih eder. Yine de adrese dayalı dizinler join gibi ilişkisel işlemlerde çok kullanışlıdır. Örneğin iç içe döngü içeren join işlemleri, birçok eşitlik seçimi üreten sorgularda adrese dayalı dizin kullanmak maliyet açısından kazançlı olur [15].

2.3 İlişkisel Veri Tabanlarında Fiziksel Tasarım

Bir veri tabanının kavramsal ve mantıksal şemaları tasarlandıktan sonra performansı artırmaya yönelik olarak fiziksel tasarımı yapılır. Fiziksel tasarım yapılırken, verilerin niteliği ve verilere erişim tarzı göz önünde bulundurulur. Fiziksel tasarım için veri tabanının desteklemesi gereken genel iş yükünü anlamak önemlidir. İş yükü, sorgu ve güncelleme işlemlerinin karışımından oluşur. Kullanıcıların belirli sorguların ve güncellemelerin hızlı çalışmasına veya saniyede yürütülen hareket sayısının yüksek olmasına dair belirli gereksinimleri vardır. İş yükü tanımı ve kullanıcıların performans gereksinimleri, fiziksel veri tabanı tasarımı sırasında bir takım kararların alınması için gereken dayanaktır. Ayrıca, iyi bir fiziksel veritabanı tasarımı oluşturmak ve zamanla gelişen kullanıcı gereksinimlerine göre performansı ayarlamak için tasarımcının VTYS'nin çalışma biçimini, VTYS tarafından desteklenen dizin oluşturma ve sorgulama işlem tekniklerinin işleyişini anlaması gerekir [15].

2.3.1 Veri Tabanı İş Yükü

İyi bir fiziksel tasarım yapabilmek için gelmesi muhtemel iş yüklerini doğru analiz etmek gerekir. Bir iş yükü analizi aşağıdaki parçaları içerir [15]:

- Sorguların bir listesi ve frekansları,
- Güncellemelerin bir listesi ve frekansları,
- Sorgu ve güncellemelerin performans hedefleri,
- Sorgu ve güncellemelerin öncelik durumları.

Ayrıca iş yükü içindeki her bir sorgu için aşağıdakiler belirlenmelidir [15]:

- Hangi tablolara erişiliyor,
- Hangi nitelikler "select" cümlesi içinde yer alıyor,
- Hangi nitelikler "where" koşul ifadesi içinde yer alıyor.

Benzer şekilde, iş yükü içindeki her bir güncelleme için aşağıdakiler belirlenmelidir [15]:

- Hangi niteliklerin "where" koşul ifadesi içinde yer aldığı,
- Güncellemenin türü(insert, delete, update) ve güncellenen tablo,

- “Update” komutları için güncellenen nitelikler.

2.3.2 Fiziksel Tasarımı Etkileyen Faktörler

Fiziksel tasarım, verilerin sadece depolama alanında uygun bir şekilde yapılandırılmasını değil, aynı zamanda iyi bir performansı garanti edecek şekilde yapılandırılmasını hedefleyen bir aktivitedir. Bir kavramsal şema için çok fazla sayıda fiziksel tasarım alternatifi vardır. Veri tabanı üzerinde yürütülecek iş yükü analiz edilene kadar, anlamlı fiziksel tasarım kararları ve performans analizleri yapılamaz. Veri tabanı yöneticileri veya tasarımcıları iş yükünde yer alan sorguları, sorguların beklenen çağrı sıklıklarını, yürütme hızındaki zamanlama kısıtlarını, beklenen güncelleme işlemlerinin sıklığını ve varsa nitelikler üzerindeki herhangi bir özel kısıtlamayı analiz etmelidir [9].

Fiziksel tasarımı etkileyen faktörler şu şekildedir:

Veri tabanı sorgularının ve hareketlerinin analiz edilmesi: Fiziksel tasarıma başlamadan önce veri tabanında çalışması beklenen sorgular ve hareketler analiz edilerek veri tabanının amaçlanan kullanımı hakkında bir fikre sahip olunmalıdır. Her bir sorgu için aşağıdaki bilgilere ihtiyaç duyulur [9]:

- Sorgunun eriştiği dosyalar,
- Seçim koşulunda yer alan nitelikler,
- Seçim koşulunun eşitlik, eşitsizlik veya aralık koşulu içerip içermediğinin belirlenmesi,
- “join” koşulu içinde yer alan nitelikler veya birden çok tablo ile bağlantı kurmaya sebep olan koşullar içinde yer alan niteliklerin belirlenmesi,
- Sorgu sonucunda ulaşılan niteliklerin belirlenmesi.

Yukarıdaki maddelerden hepsi erişim yapılarını belirlemede etkendir, 2. ve 4. maddede yer alan nitelikler ise erişim yapıları için adaydırlar. Her bir güncelleme işlemi için de aşağıdaki bilgilere ihtiyaç duyulur [9]:

- İşlem sonucunda güncellenecek dosyalar,
- Her bir dosya üzerindeki işlem türünün(insert, delete, update) belirlenmesi,

- Silme ve güncelleme işlemleri için seçim koşulunda yer alan niteliklerin belirlenmesi,
- Güncelleme işlemi sonucunda değişecek olan niteliklerin belirlenmesi.

Yukarıdaki 3. maddede belirtilen nitelikler erişim yapıları için adaydırlar, çünkü bu nitelikler silme veya güncelleme işlemine uğrayacak kayıtları bulmak için istenen koşul şartı içinde yer alırlar. 4. maddede yer alan nitelikler ise üzerinde erişim yapısı oluşturmaktan kaçınılması gereken niteliklerdir, çünkü bu nitelikler güncellemeye uğrayınca erişim yapısında da güncelleme yapılması gerekmektedir [9].

Güncellemeler bir sorgu bileşenine sahiptir ve bu bileşen güncelleme işlemi için hedeflenen satırları bulmaya yarar. Bu bileşen, iyi bir fiziksel tasarımdan ve dizinlerden yararlanabilir. Diğer taraftan güncellemeler, değiştirdikleri nitelikler üzerinde eğer mevcut dizinler varsa, bunların da güncelliğini koruyabilmek için ek çalışma gerektirirler. Bu yüzden dizinler, sorgular için fayda sağlarken güncellemeleri hızlandırabilir veya yavaşlatabilir. Tasarımcılar, dizin oluştururken bu kar zarar dengesine dikkat etmelidir [15].

Sorgu ve hareketlerin beklenen yürütme frekanslarının analiz edilmesi: Beklenen sorgu ve güncelleme işlemlerinin karakteristiklerinin yanı sıra yürütme sıklığı da dikkate alınmalıdır. Bu frekans bilgisi, nitelik bilgisi ile birlikte her sorgu ve güncelleme işlemi için biriktirilir. Böylece her bir niteliğin seçim veya birleşim ifadesinde geçme sıklığı da elde edilmiş olur [9].

Sorgu ve hareketlerin zaman kısıtlarının analiz edilmesi: Bazı sorgu ve hareketler, katı zaman kısıtlamalarına sahip olabilir. Bu zamanlama kısıtlamalarına sahip sorgularda geçen nitelikler üzerinde erişim yolu oluşturulması açısından daha fazla öncelik tanınabilir [9].

Güncelleme işlemlerinin beklenen frekanslarının analiz edilmesi: Sık sık güncellenen bir dosya için minimum sayıda erişim yolu belirtilmelidir, çünkü erişim yollarının güncellenmesi güncelleme işlemlerini yavaşlatır [9].

Niteliklerin biricik olma kısıtlarının analiz edilmesi: Bir dosyanın birincil anahtarı veya biricik olan diğer nitelikler veya nitelikler üzerinde dizin oluşturulurulabilir. Ayrıca bir niteliğin biricik olup olmadığı sadece dizin üzerinde arama yapılmasıyla anlaşılabilir.

Çünkü niteliğin sahip olduğu tüm değerler dizin yapraklarında yer alır. Örneğin bir kayıt eklenmek istendiğinde biricik olan bir niteliğin değeri zaten dizinde yer alıyorsa, ekleme işlemi nitelik üzerinde biriciklik kısıtını bozacağı için ana dosyaya erişmeye gerek kalmadan reddedilir [9].

2.3.3 Fiziksel Tasarım Ve Ayar Kararları

Fiziksel veri tabanı tasarımı ve veri tabanı ayarı yapılırken alınan önemli kararlar şu şekildedir [15]:

- Hangi dizinler oluşturulmalı?
    - Hangi tablo üzerinde dizin oluşturulmalı ve hangi nitelik veya nitelikler kombinasyonu dizinin arama anahtarı olarak seçilmeli?
    - Oluşturulan dizin kümelenmiş mi veya kümelenmemiş mi olmalı?
- Performansı artırmak için kavramsal şema üzerinde bir değişiklik yapılıp yapılmayacağına karar verilmeli.
    - Alternatif normal formlar: Performans kriterleri göz önünde bulundurularak bir şema birden fazla normal forma dönüştürülebilir.
    - Denormalizasyon: Kavramsal şema tasarım sürecinde yapılan normalizasyon işlemleri, birden fazla ayrışmış tablodan nitelik içeren sorguların performansını artırmak üzere yeniden ele alınabilir.
- Sıklıkla yürütülen sorgu ve hareketlerin daha hızlı çalışması için yeniden yazılıp yazılmayacağına karar verilmeli.

2.3.4 Dizin Oluşturma Kararları

İlişkisel veri tabanı sistemleri fiziksel veri bağımsızlığı sağlar. Bu ifadeden bir veri tabanının fiziksel tasarımını hangi dizin veya dizinler kümesi oluşturursa oluştursun veri tabanında üzerinde yürütülen sorguların aynı sonucu döndüreceği anlaşılır. Bunun yanında farklı fiziksel tasarımların performansı da elbette farklı olacaktır [2].

Belirli bir iş yükü bağlamında düşünülmezse bir veri tabanı için uygun fiziksel tasarım diye bir şey yoktur. Bunun nedeni dizinler sorguları hızlandırırken güncellemeleri

yavaşlatmaması durumunda kullanışlı olmasıdır. Bazı iş yükleri için kullanışlı olan dizinler, bazıları için tamamen gereksiz veya hatta zararlı olabilir. Ayrıca dizinler disk depolama alanını tüketen yedekli yapılardır. Mevcut depolama alanı, her zaman sınırlı olduğundan, bir veritabanının fiziksel tasarımını yaparken hangi dizinlerin oluşturulacağına karar vermek zor bir problem haline gelir [2]. Problemin zorluğuna, 2.3.5 başlığı altında değinilmektedir.

Eşitlik veya aralık koşulları içeren seçim ifadesinde yer alan nitelikler ve birleşim işlemine giren niteliklere verimli bir şekilde erişmek için dizinler gibi erişim yolları gereklidir. Seçim ve birleşim ifadesi içeren sorguların performansı büyük oranda hangi dizinlerin mevcut olduğuna bağlıdır. Diğer taraftan ekleme, silme ve güncelleme işlemleri sırasında dizinlerin varlığı ek yük getirebilir [9].

Bir iş yükü için iyi bir dizin kümesi seçilmesi, kullanılabilir dizin oluşturma tekniklerini ve sorgu iyileştiricisinin çalışma düzenini anlamayı gerektirir [15]. Dizin oluştururken uygulanması gereken fiziksel tasarım kararları şu şekildedir:

Bir nitelik üzerinde dizin oluşturma kararı: Üzerinde dizin oluşturulacak olan nitelik, anahtar bir nitelik olabilir. “where” cümleciğinde belirtilen nitelikler, dizin oluşturmak için adaydırlar. Diğer bir deyişle, eşitlik veya aralık koşulu içeren bir seçim ifadesinde geçen nitelik veya birleşim ifadesinde geçen bir nitelik, üzerinde dizin oluşturmak için uygundur. Çoğu birleşim ifadesi birincil ve yabancı anahtar nitelikleri üzerinde gerçekleştiğinden, bu nitelikler üzerinde dizin oluşturmak performansı artırır. Bir dizin, bir nitelik üzerinde veya birden çok nitelik üzerinde de oluşturulabilir. Eğer mümkünse birden fazla sorguyu hızlandıran dizinler oluşturmak üzere nitelikler seçilmelidir [2], [9], [15].

Çoklu nitelik içeren arama anahtarı seçimi: “where” cümleciği, bir tablonun birden fazla niteliği için koşul ifadesi içeriyorsa çoklu nitelik içeren dizin oluşturulabilir. Aralık sorguları bekleniyorsa, çoklu nitelik içeren dizin oluştururken, sorguları eşleştirmek için arama anahtarındaki niteliklerin sıralanmasına dikkat edilmelidir [15]. Arama anahtarı çoklu nitelik içeren dizin içindeki niteliklerin sıralaması, sorgulardaki niteliklerin sırasına denk düşmelidir [9].

Kümelenmiş dizin oluşturma kararı: Bir veri dosyası üzerinde en fazla bir tane dizin kümelenmiş olabilir, çünkü kümelenme neticesinde dosya fiziksel olarak ilgili niteliğe göre sıralanmış olur. Kümelenmiş dizinler performansı büyük oranda etkiler bu yüzden hangi nitelik üzerinde kümelenmiş dizin oluşturulacağı önemlidir. Aralık sorguları, kümelenmiş dizinlerden çok faydalanır. Bir tablo üzerinde birden fazla nitelik üzerinde aralık sorgusu varsa, hangi nitelik üzerinde kümelenmiş dizin oluşturulacağına karar verilirken sorguların seçiciliği ve iş yükündeki sıklığı göz önünde bulundurulur. Eğer bir sorgu, dosya kayıtlarına ulaşmaya gerek duymadan sadece dizin üzerinde yapılacak arama ile cevaplanıyorsa, kullanılan dizinin kümelenmiş olmasına gerek yoktur. Çünkü kümelenmiş dizinin esas faydası dosya kayıtlarına ulaşıldığı zaman görülür [15].

Adrese dayalı dizin veya ağaç temelli dizin oluşturma kararı: B+-tree dizinleri genellikle çok tercih edilir çünkü eşitlik sorgularının yanı sıra aralık sorgularını da destekler. Adrese dayalı dizinler ise sadece eşitlik koşulu içeren sorgular için uygundur, özellikle birleşim ifadesini sağlayan kayıtları bulmak için kullanışlıdırlar [9].

Yedekli dizin oluşturmaktan kaçınma: Aynı sütunlar üzerinde dizin oluşturmaktan kaçınılmalıdır. Çünkü tek bir dizin oluşturmaya göre ek olarak başka katkı sağlamazlar, ayrıca ek alan kullanırlar ve ek güncelleme maliyeti getirirler [2].

Küçük tablolar üzerinde dizin oluşturmaktan kaçınma: Bir veya iki sayfaya sığan tablolar üzerinde oluşturulan dizinler genellikle ek performans avantajları kazandırmazlar, aksine bakımı yapılacak ve yönetilecek yapıların sayısını artırırlar [2].

Dizin bakım maliyetinin dengelenmesi: Dizinler oluşturulmaya karar verildikten sonra, iş yükünde yer alan güncelleme işlemlerinin her bir dizin üzerindeki etkisini düşünmek gerekir. Bir tablo üzerinde sık sık meydana gelen güncelleme işlemleri, tablo üzerinde mevcut olan bir dizinin bakımı sebebiyle yavaşlıyorsa, ilgili dizini düşürmek gerekebilir. Bununla birlikte, bir dizinin varlığının bazı güncelleme işlemlerini hızlandırabileceği de göz önünde bulundurulmalıdır. Örneğin 6.1.2' de verilen üniversite veri tabanında yer alan student tablosunun sid niteliği üzerinde oluşturulmuş bir dizin olduğu varsayılsın. Bu durumda, belli bir sid değerine sahip öğrencinin majorid değeri güncelleneceği zaman, sid üzerinde mevcut olan dizin, veri dosyasının majorid niteliği üzerinde yapılacak güncelleme işlemini hızlandırır [15].

2.3.5 Fiziksel Tasarım Probleminin Karmaşıklığı

Fiziksel tasarımı etkileyen faktörler ve dizin oluşturma kararları da göz önünde bulundurulduğunda, fiziksel tasarım probleminin zor bir problem olduğu gözükmektedir. Problem çok hızlı büyüdüğü için otomatik bir çözüm üretmeden veri tabanı kullanıcısı tarafından çözülmesi mümkün olmayan bir problemdir. Netice olarak fiziksel tasarım problemi, NP-tam bir problem olarak nitelendirilmektedir.

Çokterimli zamanda çözülebilen problemler P ile sembolize edilirler. Ancak her problem çokterimli zamanda çözülemez. Etkili bir çözüm yönteminin bilinmediği bazı problemler vardır. Bu problemlerin bir alt kümesi olan belirsiz çokterimli tam problemler, NP-tam olarak adlandırılmaktadır. Bugüne kadar herhangi bir NP-tam problemini çözen bir çokterimli zaman algoritması keşfedilmemiştir. NP-tam kümesinde yer alan problemlerden herhangi birisi için etkili bir çözüm yöntemi varsa bu hepsi için de etkili bir çözümün olacağına işaret eder. Hamiltonian yolu problemi, gezgin satıcı problemi ve 0-1 sırt çantası karar problemi NP-complete sınıfında yer alan problem örnekleridir [18], [19].

Fiziksel tasarım probleminin NP-tam olduğunu göstermek için NP-tam olan 0-1 sırt çantası probleminden (0-1 knapsack problem) yararlanılacaktır. Bunun için sırt çantası problemi ile fiziksel tasarım probleminin durumları eşitlenecek ve NP-tam olduğu bilinen sırt çantası problemi, dizin seçim problemine indirgenecektir [2].

0-1 sırt çantası problemi girdi olarak bir tamsayı değeri olan C kapasite değerini ve nesneler kümesi olan $O=\{o_1, \ldots, o_n\}$ kümesini alır. O kümesi içindeki her bir o_i nesnesi için bir hacim değeri c_i ve bu nesnenin sağladığı fayda olan b_i değeri vardır. Problemi çözmek için O kümesi içinden C kapasitesini aşmayacak şekilde bir alt küme seçilir ve bu alt küme seçilirken seçilen elemanlardan elde edilecek olan faydanın maksimum olması istenir [2].

İndirgeme işlemi yapmak için B kapasite ve $O=\{o_1, \ldots, o_n\}$ nesneler kümesini içeren keyfi bir sırt çantası problemi ele alınır. Bu problemde her bir o_i nesnesi T_i tablosu ile ilişkilendirilir. Daha sonra bir fiziksel tasarım problemi örneği kurgulanır. T_i tablosunun col isminde ilk sütununun olduğu ve bu sütun üzerinde kümelenmemiş türde Idx isminde bir dizin oluşturulduğu varsayılsın. Idx dizininin kapladığı blok sayısı c_i ve T_i tablosunun

kapladığı blok sayısı $b_i + 1$ olsun. Kümelenmemiş Idx dizinin kapladığı blok sayısı c_i, T_i tablosunun kapladığı blok sayısından büyük olamaz dolayısıyla ortaya $b_i + 1 \geq c_i$ şeklinde bir eşitsizlik çıkar. Orijinal sırt çantası probleminde b_i ve c_i arasında böyle bir ilişki yoktur ancak bu durum problemin çözümünü değiştirmez [2].

T_i tablosunun satırları, col sütunu üzerindeki değerlerden bir tanesi 0 kalanları 1 olacak şekilde verilerle doldurulduğu varsayılsın. Daha sonra bir $W=\{Q_1, \ldots, Q_n\}$ şeklinde bir iş yükünün olduğu ve aşağıdaki gibi bir sorgunun çalıştırılmak istendiği düşünülsün [2].

Q_i = SELECT col

FROM T_i

WHERE col = 0

Depolama kapasitesi kısıtı, B bir tamsayı olmak üzere B disk bloğu olarak kabul edilsin. Hesaplamaları basitleştirmek ve anlaşılırlığı artırmak için yukarıdaki sorgu çalıştırılırken sadece kümelenmemiş dizinlerin seçelebileceği varsayılsın. Her bir tablo üzerinde I_i dizini mevcuttur ve bu dizinler birbirileri ile etkileşime girmezler. Q_i sorgusu ele alınsın. Eğer I_i dizini mevcut ise ve iyileştirici bu dizini yürütme planına dahil ederse col=0 olan satırları aramak için sadece 1 tane disk erişimi gerekir. Diğer türlü I_i mevcut değilse ya da kullanılmazsa col=0 olan satırları bulmak için tüm tablo taranır dolayısıyla b_i+1 tane disk erişimi gerekir. Bu yüzden I_i dizininin varlığı disk erişim sayısında b_i+1-1 = b_i olmak üzere b_i kadar fayda sağlar. Böylece, fiziksel tasarım problemi her biri c_i blok büyüklüğünde ve disk erişim maliyetinde b_i kadar iyileşmeye sebep olan I_i dizinlerinden oluşan en iyi alt kümenin seçilmesi problemine indirgenir. Buradan açıkça görüldüğü gibi fiziksel tasarım probleminin bu çözümü, doğrudan 0-1 sırt çantası problemi için T_i tablolarıyla ilişkili o_i nesnelerinin seçilmesine dönüşmüştür [2]. Sonuç olarak fiziksel tasarım probleminin NP-tam olduğu kanıtlanmış olmaktadır.

BÖLÜM 3

OTOMATİK FİZİKSEL TASARIM ve VERİ TABANI AYARI

3.1 **Otomatik Fiziksel Tasarım**

Bir iş yükü W={Q_1, . . . , Q_n} için her bir Q_i, SQL sorgusu veya güncellemesi olsun. B de depolama sınırı olsun. Bu durumda dizin kümesi C = {I_1, . . . , I_k}' yi elde etmek için (3.1) denkleminin sağlanması ve (3.2)'deki toplam ifadesinin sonucunun minimum olması gerekmektedir. Bu denklemlerden (3.1)'deki size(I) bir dizinin depolama maliyetini, (3.2)'deki cost(Q, C) de bir Q sorgusu için iyileştirici tarafından C kümesinde yer alan dizinlerin kullanılmasıyla kurgulanan en iyi planın maliyetidir. Bu kısıtlara uyacak şekilde C dizin kümesini bulma problemi, fiziksel tasarım problemidir [2].

$$\sum_j \mathrm{size}(I_j) \leq B \qquad (3.1)$$

$$\sum_j \mathrm{cost}(O_j, C) \qquad (3.2)$$

"2.3.4 Dizin Oluşturma Kuralları" başlığı altında değinilen kurallar mantığa uygundur ancak bu kuralların doğrulukları sadece ideal koşullarda gösterilebilir. Özellikle birçok karmaşık sorgu veya depolama kısıtlamaları olduğunda, bu kuralların tam olarak ne zaman ve nasıl birleştirileceğini bilmek zordur [2].

Fiziksel tasarım probleminin NP–tam bir problem olması dikkate alındığında belirli bir iş yükü için en iyi fiziksel veri tabanı tasarımınının kullanıcı tarafından bulunması çok zordur, bu yüzden otomatik araçlar bir gerekliliktir. Ayrıca, problemin NP-tam olması gerçeği, en uygun fiziksel tasarımın bulunmasının, otomatik araçlar için bile küçük problemler haricinde çok mümkün olmadığını gösterir [2].

Fiziksel tasarım problemi üç bileşen ile analiz ve tasvir edilebilir: Dizin yapılandırmaları için arama uzayı, dizin yapılandırmalarını değerlendirmek için maliyet modeli ve arama uzayını dolaşmak için numaralandırma stratejisi [2].

3.1.1 Arama Uzayı

Fiziksel tasarım problemi bağlamında arama uzayı, belirli bir veri tabanı ve iş yükü için aday olarak düşünülebilecek dizin kümesidir. Sorgu işlemedeki ilerlemeler, giderek artan uygulama karmaşıklığı ile birleştiğinde, tek sütunlu dizin yaklaşımını yetersiz hale getirmiştir. Mevcut karar destek uygulamaları, genellikle ayrıntılı erişim yolu seçimi gerektiren ve tek sütunlu dizinlerin genellikle performansı iyileştirmede yetersiz olduğu karmaşık sorgular üretir. Fiziksel tasarım problemi için yapılan son çalışmalarda arama uzayındaki çoklu sütun dizinler de dikkate alınmaktadır [2].

Dizin arama uzayının genel bir karakterizasyonuna karar verdikten sonra bir iş yükü için arama uzayının ne olacağına karar vermek gerekir. Belirli bir iş yükü için arama uzayını tanımlamanın geleneksel yolu iki adımdan oluşur. İlk adım, iş yükündeki her bir sorgu için aday dizin kümesini tanımlamaktır. Bu adımı gerçekleştirmek için sorgu cümlesini sözdizimsel olarak ele almaktan sorgu iyileştiricisinin kullanıldığı daha karmaşık yöntemlere kadar değişik teknikler vardır. Her sorgu için aday dizin kümesini karakterize ettikten sonra, ikinci adım iş yükü için arama uzayını tanımlamaktır [2].

3.1.1.1 Tek Bir “Select” Sorgusu için Aday Dizinler

İş yükü tek bir “select” sorgusundan oluştuğunda ve depolama alanı sınırı bulunmadığında, fiziksel tasarım problemi çok basit hale gelir. Bu yapılandırmaya dizin eklemek performansa zarar vermez çünkü dizinler güncellenmez ve depolama sınırı olmadığından yapılandırmayı geçersiz kılmaz. Tek bir SELECT sorgusu için en uygun yapılandırmayı içeren küçük bir aday dizin seti bulmak için çeşitli yaklaşımlar vardır [2].

Ayrıştırma tabanlı yaklaşım: Belirli bir sorgu için aday dizin kümesini elde etmenin en kolay yolu, sorgu cümlesinin sözdizimsel olarak analizini gerçekleştirmektir. Buna göre, ilk olarak sorgu cümlesi ayrıştırılır ve ayrıştırma ağacından üzerinde dizin oluştulabilecek nitelik kümesi belirlenir. Bir c niteliği, sorgu içinde “where”, “order-by” ve “group-by”

cümleciği içinde yer alıyorsa veya “update” ifadesinin koşulunda yer alıyorsa üzerinde dizin oluşturulabilir. Örneğin aşağıdaki sorgu göz önünde bulundurulursa, üzerinde dizin oluşturulabilecek sütunlar R.a, R.x ve S.y sütunlarıdır. Üzerinde dizin oluşturulabilecek sütunlar belirlendikten sonra üzerinde dizin oluşturulmayacak sütunlar belirlenir. Aşağıdaki sorgunun işlenmesinde gerekli olmasına rağmen R.b sütunu üzerinde dizin oluşturmaya gerek yoktur [2].

SELECT R.a, R.b FROM R, S WHERE R.x = S.y ORDER BY R.a

Sorguda geçen tüm niteliklerin üzerlerinde dizin oluşturulup oluşturulamayacağı belirlendikten sonra aday dizin kümesi belirlenmeye geçilebilir. Aday dizin kümesini, sorgu motorunun özelliklerine bağlı olarak tanımlamanın farklı yolları vardır. Örneğin sorgu moturu yalnızca tek sütunlu dizinlerden yararlanıyorsa, sorgu cümlesinden elde edilen dizinlenebilir her bir nitelik için bir kümelenmiş dizin ve bir kümelenmemiş dizin oluşturulabilir. Bu durumda yukarıdaki sorguda R.a, R.x ve S.y nitelikleri için bir kümelenmiş ve bir kümelenmemiş olmak üzere toplam 6 tane aday dizin belirlenirdi [2].

Ayrıca, sorgu motorları, çoklu sütun içeren dizinleri de işleyebilir. Örneğin, yukarıdaki sorgu için kaplayan dizin aday kümesi şu şekildedir: {R(x), R(x|b), R(x|a), R(x|a,b), R(a), R(a|b), R(a|x), R(a|b,x), R(a,x), R(a,x|b), R(x,a), R(x,a|b), S(y)}. Ayrıca, bu aday kümesindeki her bir kümelenmemiş dizin için aynı arama anahtarı sütunlarına sahip bir kümelenmiş dizin de aday olabilir [2].

Yürütme planı tabanlı yaklaşım: Bu yaklaşımda, ayrıştırma ağacından ziyade iyileştirici tarafından üretilen yürütme planı kullanılır. Aday dizinleri elde etmek için, sorgu cümlesi yerine yürütme planı işlenerek, iyileştiricinin kendisinden faydalanılınır. Bu yaklaşım sayesinde iyileştirici ile senkronize olunur. İyileştiriciye yeni bir basitleştirme stratejisi dahil edilirse, aday seçim modülü tarafından direkt kullanılabilir [2].

Aday dizin kümesinin azaltılması: Aday dizin kümesi elde edildikten sonra bu kümenin büyüklüğünü azaltmak için bazı fırsatlar olabilir. Dizinlerin kümesini azaltmak için yaygın bir mekanizma, sorgu iyileştiricisini kullanmaktır. Mesela, tüm aday dizinlerin oluşturulduğu ve sorgunun iyileştirildiği varsayılsın. Bu durumda, iyileştirici dizinleri kullanarak en iyi planı elde edebilir. Daha sonra iyileştirici tarafından üretilen plan

incelenip planda kullanılan dizinleri içerecek şekilde aday dizin kümesi sadeleştirilebilir [2].

Bu yaklaşıma göre, bir sorguyu iyileştirmek ve sadeleştirilmiş aday dizin seti elde etmek için önce büyük bir dizin kümesi oluşturmak gerekmektedir. Dizin oluşturmanın maliyeti yüksektir bu yüzden maliyet bu yaklaşımın büyük bir dezavantajı olarak görülebilir. Bu duruma çözüm olarak, dizinleri oluşturmadan giriş sorgusunun iyileştirilmesi simüle edilebilir [2]. Ayrıca, bu yaklaşımın diğer bir problemi de başlangıç aday dizin kümesinin bir tablo için birden fazla kümelenmiş dizin içermesi durumudur. Çünkü böyle bir yapılanma geçersiz olur ve gerçekleştirilemez. Genel olarak, iyileştirmeyi garantilemek için, kümelenmiş dizinlerin her bir kombinasyonu için iyileştirme adımı gerçekleştirilir. Bazı yaklaşımlar, zaman alan bu adım yerine az sayıda alternatif belirleyip birkaç iyileştirme çağrısı ile en iyi planın dizinlerini belirlemeye çalışır [2].

3.1.1.2 **Bir İş Yükü İçin Aday Dizin Kümesi**

Tek bir "select" sorgusu için aday dizin kümesi belirlemenin yanında birçok sorgu ve güncellemeden oluşan bir iş yükü için aday dizin kümesi belirleme işlemi daha karmaşıktır. Bir iş yükü için aday dizin kümesi elde etmeyi amaçlayan basit bir yaklaşıma göre, iş yükündeki her bir sorgu için aday kümeleri elde edilir ve ortaya çıkan kümeler birleştirilir. Ancak bu yaklaşım, güncelleme ve depolama kısıtları sebebiyle bazı fırsatların kaçmasına sebep olur. Çünkü iş yükündeki herhangi bir sorgunun bireysel aday dizin kümesinde yer almayan bir dizin, iş yükü için belirlenen aday dizin kümesinde yer alabilir. Örneğin aşağıdaki iki tane sorgunun ele alındığı ve bir depolama kısıtı olduğu varsayılsın.

Q1 = SELECT A, B, C FROM R WHERE 10 < A < 20
Q2 = SELECT A, B, D FROM R WHERE 50 < A < 100

Bu durumda Q1 için en iyi dizin I_1=R(A|B, C) ve Q2 için en iyi dizin I_2=R(A|B, D) olur. Eğer depolama kısıtı olmazsa en iyi dizin kümesi {I_1, I_2} olur. Depolama kısıtı olduğu durumda ise I_1 ve I_2 'nin birlikte depolama kısıtını aştığı varsayılsın. Bu durumda bu iki dizinden birisi seçilirse sorgulardan birisi çok iyi performans sergilerken diğer sorgunun performansı çok gelişmeyecektir. Bu iki dizine alternatif olarak I_3=R(A|B,C,D) dizini kullanılırsa, her iki sorgu da A niteliği üzerindeki şartları sağlayan satırlara erişebilir ve bu

dizin her iki sorgu için de gerekli olan nitelikleri kapsar. I_3 dizini, I_1 ve I_2 dizinlerine göre bir tane fazla nitelik barındırdığı için I_3 üzerinde arama yapmanın maliyeti diğer dizinlere göre biraz daha yüksek olacaktır. Ancak I_3 dizini, I_1 ve I_2 dizinlerinin kapladığı toplam alandan daha küçüktür ve depolama kısıtını aşmaz ve her iki sorgunun da performansını artırır. Bu yüzden I_3 dizini, Q1 ve Q2 'nin bireysel aday dizin kümesinde yer almamasına rağmen {Q1, Q2} iş yükünün aday dizin kümesinde yer alması performansı artırır. Bundan dolayı, iş yükü için aday dizinler belirlenirken, başlangıçta her bir sorgunun bireysel aday dizinlerinin birleşimiyle oluşan aday dizinlerden yeni dizinler türetilebilir [2].

3.1.2 Maliyet Modeli

Dizin yapılandırmalarının arama uzayında anlamlı bir şekilde dolaşabilmek için bir C yapılandırmasının büyüklüğünü ve bir iş yükünün C yapılandırması altındaki iyileştirme maliyetini bilmeye ihtiyaç vardır. Bir yapılandırmanın maliyeti, yapılandırmada yer alan her bir dizinin maliyetinin toplamıdır. Bir iş yükünün bir yapılandırma altında maliyetini hesaplamak daha zordur. Bunun için en doğru yöntem, yapılandırmadaki dizinleri oluşturmak, iş yükündeki tüm sorguları optimize etmek ve tüm sorgular için tahmini maliyeti toplamaktır. Sorguları optimize etmek nispeten ucuz olsa da veritabanındaki tabloların tekrar tekrar taranması ve sıralanması gerektiğinden yapılandırmadaki dizinlerin oluşturulması oldukça masraflıdır. Bu yönteme alternatif olarak, iyileştiricinin dışında bir maliyet modeli geliştirerek ve sorgu maliyetlerinin fiziksel tasarımdaki değişikliklere göre nasıl değişeceğini tahmin edilebilir. Bu alternatif yöntemdeki temel sorun ise, dizinlerin sadece iyileştirici tarafından kullanan bir plana dahil edilmesi durumunda faydalı olmalarıdır. Bir dizinin aday olma ihtimali yüksek bile gözükse, iyileştirici bu dizinden yararlanan yürütme planı oluşturmazsa, dizin gereksiz olacaktır [2].

Veritabanı yönetim sisteminde, her aday yapılandırmasının gerçekleştirilmesi ve bu yapılandırmaya göre iş yükü sorgularının optimize edilmesi ve maliyetlerinin hesaplanması pratikte mümkün değildir. Ölçeklenebilir fiziksel tasarım algoritmalarının tasarlanması için önemli bir sorun, bir dizin yapılandırması altında sorgu maliyetlerini verimli ve doğru bir şekilde tahmin etmektir. Bu sorunu çözmek için, DBMS'deki varsayımsal bir yapılandırmayı simüle edebilen ve yeni dizinleri gerçekleştirmeden

sorguları optimize edebilen yeni bir bileşene ihtiyaç vardır. Bu süreç “what-if iyileştirme” sürecidir [2].

What-if iyileştirme sürecinde, iyileştirici dizinlerin kendisi yerine meta veri kullanır. İyileştirici, bir dizinin yürütme planında yararlı olup olmadığına karar vermek için dizinin oluşturulduğu tablo hakkındaki bilgileri, dizinin arama anahtarı olan ve olmayan nitelikleri, dizinin büyüklüğü, bütünlük kısıtları gibi bilgileri kullanır. Yalnızca meta veri bilgisi içeren ama gerçek veriler içermeyen bir dizin varsayımsal dizin olarak adlandırılır. İyileştirici açısından bakıldığında böyle varsayımsal bir dizinin normal bir dizine göre farkı yoktur. İyileştirme süreci normal ve varsayımsal dizinleri aynı anda dikkate alarak ilerleyebilir. En son oluşturulan yürütme planı varsayımsal dizinlere de referans verebilir. Bu tür yürütme planları gerçekte yürütülebilir değildir çünkü varsayımsal dizinler gerçekten mevcut değildir. Ancak oluşturulan yürütme planı ve bir sorgunun maliyeti bütün varsayımsal dizinler gerçeklenerek normal dizinlere dönüştürülse bile değişmez. Bu nedenle sistem kataloglarında ilgili meta verileri oluşturarak bir dizinin varlığı simüle edilebilir. Meta veri küçük ve dizinin büyüklüğünden bağımsız olduğundan bu süreç verimlidir ve normal iyileştirme süreçleri ile karşılaştırıldığında neredeyse hiç ek yük getirmez [2].

3.2 İlişkisel Veri Tabanlarında Veri Tabanı Ayarı

Sistemin ilk tasarımını yaparken doğru ve detaylı iş yükü bilgilerinin gelmesi zor olabilir. Bir veri tabanı kullanıma açıldıktan sonra uygulamaların, hareketlerin, sorguların ve görüntülerin gerçek kullanımı ilk fiziksel tasarım sırasında hesaba katılmamış olabilecek faktörleri ve sorunlu alanları açığa çıkarır. Veritabanı tasarımının ilk aşamasından sonra, veritabanının gerçek kullanımı, ilk tasarımın iyileştirilmesi için kullanılabilecek değerli bir bilgi kaynağı sağlar. Veri tabanının başlangıç tasarımı yapılması ve kullanıma açılmasından sonra en iyi performansı elde etmek için gerçek kullanım verileri ışığında başlangıç tasarımında yapılan iyileştirmeler veri tabanı ayarıdır [9], [15].

Beklenen iş yükü hakkında yapılan birçok varsayım, gerçek sistemin çalıştırılması sonucunda gözlenen verilerle değiştirilebilir. Veri miktarı hakkındaki başlangıç tahminleri sistem kataloglarında yer alan gerçek istatistiki bilgilerle değiştirilebilir. Sorguların

dikkatle izlenmesi beklenmeyen problemlerin farkedilmesini sağlayabilir. Örneğin iyileştirici iyi planlar üretmek için bazı dizinleri kullanmıyor olabilir [15].

Ayar yapmanın amaçları şu şekildedir [9]:

- Uygulamaların daha hızlı çalışmasını sağlamak,
- Sorguların ve hareketlerin yanıt süresini iyileştirmek,
- Verimliliği diğer bir deyişle birim zamanda hizmet verilen hareket sayısını artırmak.

Veri tabanı ayar sürecine bazı istatistiki veriler girer, bu veriler VTYS tarafından toplanır. Toplanan istatistiki veriler [9]:

- Tabloların boyutları,
- Bir sütundaki farklı değerlerin sayısı,
- Belirli bir sorgu veya hareketin belirli bir zaman aralığında gönderilip yürütülmesi sayısı,
- Sorgu ve hareketlerin farklı aşamalarının işlenmesi için gereken süreler.

Veritabanı sistemi faaliyetleri ve süreçleri izlenerek depolama alanı istatistikleri, disk üzerindeki toplam okuma yazma istatistikleri, bir dizin içindeki seviye sayısı gibi dizin istatistikleri de elde edilir [9].

Bir veritabanının ayarlanması sürecinde hareketlerin eş zamanlılığını artırmak için kilit yarışlarının önlenmesi, tampon havuzunun büyüklüğünün ve kullanımının iyileştirilmesi, kaynakların verimli kullanılması gibi problemler çözülmeye çalışılır. Bu problemlerin çoğu, uygun fiziksel VTYS parametrelerini ayarlayarak, cihaz konfigürasyonlarını değiştirerek, işletim sistemi parametrelerini değiştirerek ve diğer benzer faaliyetlerde bulunularak veri tabanı yöneticisi tarafından çözülebilir [9]. Bu bölümde çeşitli fiziksel tasarım kararlarının ayarlanmasına değinilecektir.

3.2.1 Dizinlerin Ayarlanması

İlk dizin kümesi seçimi, çeşitli sebeplerden dolayı yeniden gözden geçirilebilir. Bazı sorguların çalışması bir dizin eksikliğinden dolayı çok uzun sürebilirken bazı dizinler ise hiç kullanılmayabilir. Bazı dizinler de çok fazla güncellenebilir çünkü sık sık değişikliklere

uğrayan bir nitelik üzerinde dizin oluşturulmuş olabilir. Veri tabanı kullanıma açıldıktan sonra gözlenen iş yükü, ilk iş yükü incelemelerinde önemli kabul edilmiş bazı sorgu ve güncellemelerin aslında çok sık yürütülmediğini ortaya koyabilir. Gözlenen iş yükü, önemli olan bazı yeni sorguları ve güncellemeleri de belirleyebilir. Bu yeni bilgilerin ışığında, başlangıçtaki dizin seçimi gözden geçirilir. Mevcut olan bazı dizinler düşürülebilir ve yenileri eklenebilir. Bu ekleme ve çıkarma işlemlerini yaparken de yine, ilk dizin kümesi oluşturulurken kullanılan kurallar geçerli olur [9], [15].

Çoğu VTYS'nin bir sorgunun nasıl yürütüldüğünü -iyileştirici tarafından oluşturulan plan- göstermek için bir komut veya izleme özelliği vardır. Bu özellik sayesinde hangi sırayla hangi işlemlerin yapıldığı ve hangi dizinlerin kullanıldığı görülebilir. Yürütme planları analiz edilerek, performans kaybı sorunlarının nedenlerini teşhis etmek mümkündür. [9].

Ayar sürecinde başlangıç tasarımında yer alan dizinleri gözden geçirmenin yanı sıra, bazı dizinleri periyodik olarak da gözden geçirmek gerekebilir. Örneğin bir B+-tree dizini için bu dizini kullanan uygulama, dizinin silinen sayfalarını birleştirmiyorsa bazı durumlarda alan doluluğu önemli ölçüde azalabilir. Bu da dizinin büyüklüğünü ve yüksekliğini gerekenden daha büyük hale getirir ve dolayısıyla erişim süresini artırabilir. Bu durumda dizinin yeniden oluşturulması düşünülmelidir. Benzer şekilde, çok fazla ekleme yapmak, kümelenmiş bir dizinde performansa etki eden taşmalara neden olabilir. Yine, dizini yeniden oluşturmak faydalı olabilir [9], [15].

3.2.2 Veri Tabanı Mantıksal Tasarımının Yeniden Ayarlanması

Belirli bir fiziksel veritabanı tasarımı beklenen hedefleri karşılamıyorsa, veri tabanı yöneticisi mantıksal veritabanı tasarımına dönerek mantıksal şemada denormalizasyon gibi ayarlamalar yapabilir ve oluşan mantıksal tasarıma denk düşen yeni fiziksel tablolar ve yeni dizin kümesi oluşturabilir. Veri tabanı tasarımı yapılırken veri gereksinimleri kadar işlem gereksinimleri de dikkate alınmalıdır. Eğer işlem gereksinimleri dinamik olarak değişiyorsa, kavramsal şemada değişiklikler yapılmalı ve bu değişiklikler mantıksal şemaya ve fiziksel tasarıma da yansıtılmalıdır.

Örneğin bazı tabloların denormalizasyon işlemine uğraması gerekebilir. Çünkü iki veya daha fazla tabloda yer alan niteliklerin sıklıkla birlikte kullanılmasının gerektiği durumlar

olabilir. Bu durumda normalizasyon seviyesinin BCNF seviyesinden 3NF, 2NF veya 1NF seviyesine düşürülmesi gerekebilir [9].

3.2.3 Sorguların Ayarlanması

Bir sorgunun beklenilenden çok daha yavaş çalıştığı fark edilirse, sorunu bulmak için sorgu dikkatlice incelenmelidir. Bazen sorgunun yeniden yazılması, bazı dizin ayarlarıyla birlikte sorunu çözebilir. Sorgu ayarının gerekli olabileceğini gösteren başlıca iki gösterge vardır. Bunlar [20]:

- Bir sorgu için çok fazla disk erişimi yapılması,
- Sorgu planının gerekli dizinlerin kullanılmadığını göstermesi.

Sorgu ayarında ilk adım olarak iyileştirici tarafından üretilen planın iyi anlaşılması gerekir. Yeni dizinler oluşturma fikrinden önce sorguyu yeniden yazarak mevcut dizinlerle iyileşmenin sağlanıp sağlanmayacağı kontrol edilmelidir. Örneğin 6.1.2'de verilen üniversite veritabanında yürütülmek üzere yazılmış olan aşağıdaki sorguda geçen sname ve gradyear nitelikleri üzerinde dizinler varsa, bu dizinlerin gerekli tablo satırlarına ulaşmak için kullanılabileceği düşünülebilir. Ancak bazı iyileştiriciler, "where" cümleciği içinde yer alan koşulları bir bütün olarak görür. Bu tür bir iyileştirici, sname ve gradyear nitelikleri üzerinde mevcut olan her iki dizini de kullanamaz ve student tablosu üzerinde baştan sona tarama yapmaya yöneltir. Bu iyileştirici için bu sorgu, eğer iki ayrı sorgunun birleşimi(union) olacak şekilde birisi "where sname = 'joe'" olmak üzere ve diğeri de "where gradyear = 2004" şeklinde yeniden yazılırsa sname ve gradyear nitelikleri üzerinde yer alan dizinler kullanılarak verimli bir şekilde cevaplanabilir [15].

SELECT sid FROM student WHERE sname = 'joe' OR gradyear = 2004

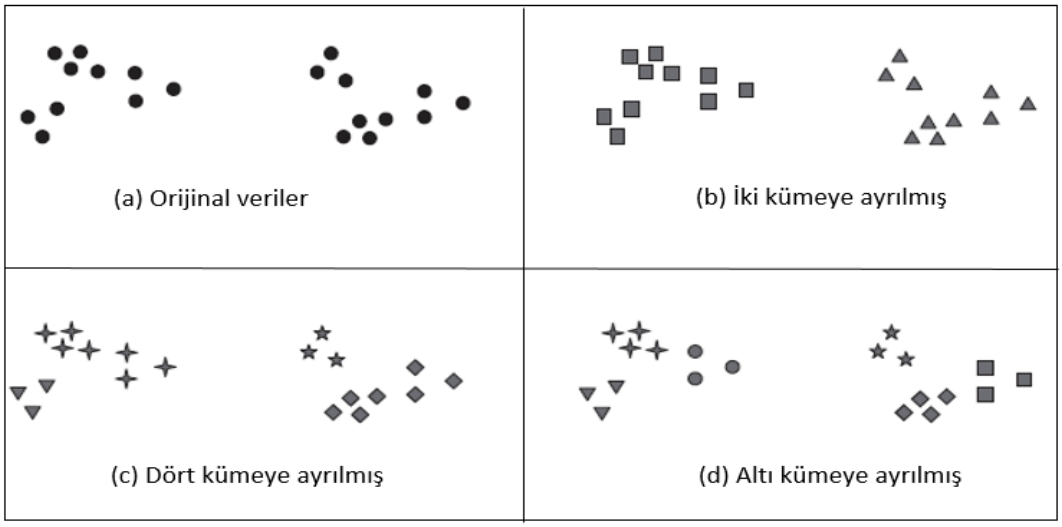
BÖLÜM 4

KÜMELEME ALGORİTMALARI

Veri madenciliği, büyük miktarda veriden otomatik olarak bilgi çıkarma sürecidir [21]. Bu süreçde, çıkan sonuçları kullanarak tahminler yapmak üzere veri içindeki ilişkiler ve örüntüler tespit edilmeye çalışılır [22]. Bu bilgileri elde etmek için sınıflandırma, kümeleme, tahmin gibi teknikler kullanılır.

Veri madenciliği yöntemlerinden kümeleme tekniği, nesneleri, ortak karakteristiklerine bağlı olarak, küme olarak adlandırılan gruplara ayırma işlemidir. Kümeleme işlemi sonucunda, bir küme içindeki nesneler birbirine çok benzer iken kümeler arasında benzerlik mümkün olduğunca az olur. Farklılıklar ve benzerlikler, nesneleri tanımlayan nitelik değerlerine göre değerlendirilir ve genellikle mesafe ölçümleri kullanılarak belirlenir [23].

Kümelemenin amacı benzer nesneleri aynı grupta toplarken farklı olanları ise başka gruplara atamaktır. Bir grup içindeki benzerlik ve gruplar arası farklılık arttığı zaman daha iyi ve daha belirgin bir kümelenme ortaya çıkar. Aynı veri topluluğu üzerinde farklı kümeler oluşturabilir. Örneğin Şekil 3.1'de 20 tane nesneden oluşan veri topluluğu 3 farklı yöntemle kümelenmeye çalışılmıştır. İlk yöntemde iki küme, ikinci yöntemde 4 küme ve üçüncü yöntemde de 6 küme elde edilmiştir. Bu şekil, kümeleme sonucunda elde edilecek kümeler kümesinin kesin olmadığını, elde edilecek sonucun verinin doğasına ve arzulanan sonuca bağlı olduğunu göstermektedir [23].



Şekil 4. 1 Aynı veri kümesinin farklı şekillerde kümelenmesi

Verilere ve istenen küme özelliklerine bağlı olarak bölümlemeli, hiyerarşik, yoğunluk tabanlı, grafik tabanlı ve spektral kümeleme gibi farklı küme oluşturma yöntemleri vardır [24]. Küme analizinin en basit ve en temel versiyonu, bir veri topluluğunu gruplara ayıran bölümlemeli (partitioning) kümeleme yöntemidir [23]. Bu yöntemde, veri kümesindeki nesneler, hesaplanabilir bir küme değerine dayanarak bir kümeden diğerine taşınarak yeniden yerleştirilir. Veriler, bir yer değiştirme metodu ile yinelemeli olarak k tane küme arasında yeniden gruplanır. Ayrıca bu yöntem merkez tabanlı kümeleme olarak da bilinir. Bu yöntemde, küme sayısı kullanıcı tarafından önceden tanımlanmalıdır [25]. Oluşan kümeler birbiri ile kesişmezler dolayısıyla her bir nesne sadece bir kümeye atanır [21].

Bu tez çalışmasında bölümlemeli kümeleme algoritmaları olan k-means ve k-medoids algoritmaları kullanılacaktır.

4.1 **K-Means Algoritması**

Bu algoritma, veri grubunda yer alan elemanların ortalama değerlerinin hesaplanmasıyla ortaya çıkan merkezi noktaya göre ilerleyen bir tekniğe sahiptir. Bunu yaparken de küme içi benzerlikleri maksimum, kümeler arası benzerlikleri de minimum yapmaya çalışır.

K-means algoritması, D veri setinin d boyutlu uzayda n tane nesneye sahip olduğu bir durumda, D veri setini k tane kümeye bölmeye çalışır. n tane elemandan oluşan D veri setinin elemanlarının öklit uzayında olduğu varsayılsın. k tane küme $C_1, \ldots C_k$ olsun. $1 \leq i$ ve $j \leq k$ olmak üzere $C_i \subset D$ ve $C_i \cap C_j = \emptyset$ olur. C_i kümesinin merkez noktası c_i ve $p \in C_i$ olmak üzere p ve c_i arasındaki uzaklık, $dist(p,c_i)$ ile öklit uzaklığı olarak hesaplanır. C_i

kümesinin kalitesi, küme içindeki varyasyona bağlıdır. Bunun için C_i kümesindeki her bir eleman ile c_i arasındaki karesel hata fonksiyonu toplamı (4.1)'deki denklem ile hesaplanır [23].

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist(p, c_i)^2 \quad (4.1)$$

E, veri seti içindeki tüm nesneler için karesel hata toplamıdır [23]. K-means algoritmasının amacı k tane küme üzerindeki karesel hata toplamını en aza indirgemektir [26]. Böylece, küme içi benzerlik maksimum yapılırken kümeler arası benzerlik minimum olmuş olur. K-means, karesel hata toplamını en aza indirgeyen bir kümelenmeyi bulmak için yinelemeli bir yaklaşım kullanır. Kümelemeye başlamak için k küme sayısı, isteğe bağlı bir değer olarak başlangıçta seçilmelidir [24].

K-means algoritmasının çalışma şekli de şu şekildedir [23]:

(1) k tane küme merkezi D veri setinden rastgele olarak seçilir,

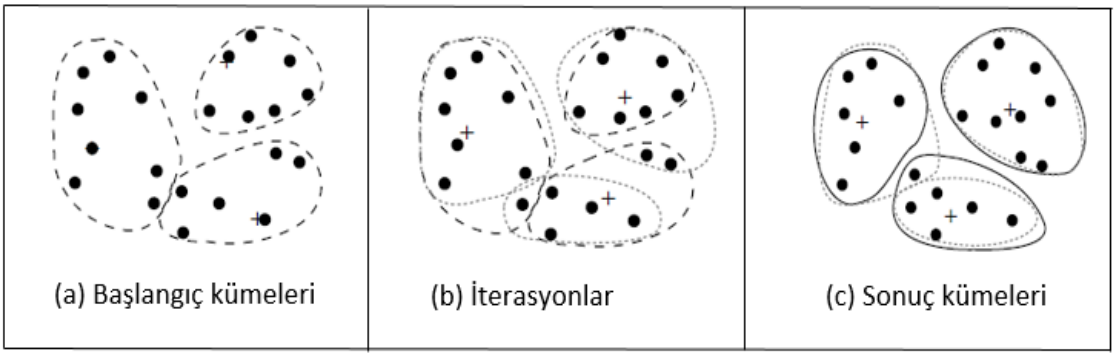
(2) Tekrar,

(3) Her nesne, kümelerdeki nesnelerin ortalama değerine bağlı olarak nesnenin en benzer olduğu kümeye atanır,

(4) Küme merkezleri yeniden hesaplanarak güncellenir,

(5) Kümelerdeki değişim bitene kadar ikinci adımdan devam edilir.

K-means algoritması iterasyonlu olarak ilerleyerek küme içi varyasyonları geliştirir. İterasyonlar kümelerdeki elemanların değişmesi bitene kadar devam eder. Şekil 3.2'de k=3 için k-means algoritmasının ilerlemesi gösterilmiştir [23].



Şekil 4. 2 Bir veri topluluğunun k-means ile kümelere ayrılması aşamaları

4.2 **K-medoids Algoritması**

K-means algoritması aykırı verilere duyarlıdır. Çünkü bu tür nesneler, verilerin çoğunluğundan uzaktadır ve dolayısıyla bir kümeye atandığında kümenin ortalama değerini büyük ölçüde bozabilirler. Bu etki, k-means algoritmasında karesel hata fonksiyonun kullanılması nedeniyle daha da şiddetlenir. Bu durum diğer nesnelerin yanlış kümelere atanmasına sebep olabilir [23].

Aykırı değerlere karşı olan bu hassasiyeti azaltmak için ortaya sürülen diğer bir algoritma k-medoids algoritmasıdır. Bu algoritma, k-means algoritmasından farklı olarak, bir küme içindeki nesnelerin ortalama değerini kümenin referens noktası olarak almaz. Bunun yerine, her bir kümeyi temsil etmek üzere, kümenin en merkezinde yer alan gerçek bir nesneyi kullanmayı önerir. Bu, yöntemin kilit noktasıdır. Temsilci nesneler medoid olarak da adlandırılır. Her bir nesne, en benzer olduğu temsilci nesnenin yer aldığı kümeye atanır. Dolayısıyla bu metod ile kümeleme, her bir nesne ile bu nesneye karşılık gelen referans noktası arasındaki benzersizliklerin toplamını minimize etme prensibine dayanarak gerçekleştirilir. Bunun için mutlak hata kriteri uygulanır. Her bir p nesnesi ile karşılık düştüğü C_i kümesinin o_i temsilci nesnesi arasındaki benzersizlik dist(p, o_i) ile hesaplanır. Bu benzersizliklerin toplamını minimize etmek için mutlak hataların toplamı (4.2)'deki gibi hesaplanır. Böylelikle k-medoids metodu, n tane nesneyi mutlak hatalarının toplamı en az olacak şekilde k tane kümeye ayırır [23], [27].

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \mathrm{dist}(p, o_i) \qquad (4.2)$$

Veri setinin büyüklüğü, elde edilen kümelerin kalitesi ve algoritmanın sonuç üretme süresi kriterlerine bağlı olarak k-medoids algoritmasının PAM (Partitioning Around Medoids), CLARA (Clustering Large Applications), CLARANS (Clustering Large Applications Based Upon Randomized Search) gibi çeşitli isimler altında uygulama yöntemleri vardır [23].

Bunlardan en yaygın olan PAM algoritması, kümeleme işlemi için açgözlü ve yinelemeli bir yaklaşım izler. K-means algoritmasına benzer şekilde k küme sayısı isteğe bağlı bir değer olarak başlangıçta seçilmelidir. PAM algoritmasının adımları şu şekildedir [23]:

k: küme sayısı

D: n tane nesneden oluşan veri seti

(1) D içinden keyfi olarak k tane nesne, başlangıç temsilci nesneleri olarak seçilir,

(2) Tekrar,

(3) Geri kalan nesneler, en yakın oldukları o_j temsilci nesnesinin bulunduğu kümeye atanır,

(4) Rastgele olarak temsilci olmayan o_{random} nesnesi seçilir,

(5) o_j ve o_{random} nesneleri için mutlak hata kriterleri hesaplanır,

(6) Eğer o_{random} için karesel hata fonksiyonu daha düşükse, medoid o_{random} olarak değiştirilir,

(7) Değişiklik olmayana kadar ikinci adımdan devam edilir.

4.3 **K-means ve K-medoids Algoritmları ile Örnek Uygulama**

Tek boyutlu uzayda yer alan verilerden oluşan veri seti D: {1, 2, 3, 8, 11, 13, 25} olsun. Bu veri seti, k-means ve k-medoids algoritmaları yardımıyla, k=2 olmak üzere 2 kümeye ayrılmaya çalışılsın. Başlangıçta küme merkezleri keyfi olarak m_1=3 ve m_2=8 olarak seçilsin [23].

K-means algoritması kullanıldığında, kümeleme işlemi 3 tane iterasyon ile tamamlanmıştır. Birinci küme {1, 2, 3, 8} ve ikinci küme {11, 13, 25} olur. Oluşan kümeler Çizelge 4.1'de gösterilmiştir.

Çizelge 4. 1 K-means ile iterasyonlar

İterasyon no	1. küme	2. küme	m_1	m_2
1	{1, 2, 3}	{8, 11, 13, 25}	2	14,25
2	{1, 2, 3, 8}	{11, 13, 25}	3,5	16,33
3	{1, 2, 3, 8}	{11, 13, 25}	3,5	16,33

K-medoids algoritması kullanıldığında, kümeleme işlemi 2 tane iterasyon ile tamamlanmıştır. Birinci küme {1, 2, 3} ve ikinci küme {8, 11, 13, 25} olur. Oluşan kümeler Çizelge 4.2'de gösterilmiştir.

Çizelge 4. 2 K-medoids ile iterasyonlar

İterasyon no	1. küme	2. küme	m_1	m_2
1	{1, 2, 3}	{8, 11, 13, 25}	2	11
2	{1, 2, 3}	{8, 11, 13, 25}	2	11

K-means ve k-medoids algoritmalarının ürettiği sonuca bakıldığında farklı oldukları görülür. Bunun sebebi D veri içindeki {25} verisinin diğer verilere göre sayı doğrusunda çok uzakta olmasıdır. K-means algoritması aykırı verilere karşı duyarlıdır. Bu durumda {25} verisi k-means ile yapılan kümelemeyi olumsuz etkilemiştir ve {8} verisi birinci kümeye atanmıştır [23].

BÖLÜM 5

DİZİN SEÇİM TEKNİĞİ

Otomatik dizin seçimi için önerilen hipotez kurgulanırken, Zaman vd. [3] tarafından kümeleme tekniği kullanılarak yapılan otomatik dizin seçimi çalışmasından ilham alınmıştır. Önerilen hipotezin çeşitli aşamaları Şekil 5.1'de gösterilmiştir. Bu aşamalar Zaman vd. [3] tarafından yapılan çalışmada da yer almaktadır. Ancak Şekil 5.1' deki aşamaların uygulanmasında, bu tez çalışması ile Zaman vd. [3] tarafından yapılan çalışma arasında bazı farklılıklar bulunmaktadır. Bu farklılıklar, "5.5 Dizin Seçim Tekniğinin Özgünlüğü" başlığı altında belirtilmiştir.

Şekil 5. 1 Dizin seçim tekniğinin aşamaları

5.1 Sorgu Frekanslarının Belirlenmesi

Dizin seçim aracı, ilk olarak dizin seçimi yapılacak olan iş yükünü girdi olarak alır. Bu iş yükünde yer alan birbirinden farklı sorguları ve bu sorguların frekanslarını belirler. Böylece iş yüküne göre daha az sorgu içeren, birbirinden farklı sorgulardan oluşan bir sorgu kümesi ve bu kümede yer alan her bir sorgunun iş yükü içindeki frekansı elde edilmiş olur. Bundan sonraki aşamalarda, iş yükünün bir alt kümesi olan ve birbirinden farklı sorgulardan oluşan bu sorgu kümesi kullanılacaktır. Gerekli yerlerde ise sorguların frekansı çarpan olarak kullanılacaktır.

5.2 Dizin Oluşturmaya Aday Niteliklerin Belirlenmesi

Bu aşamada, üzerinde dizin oluşturalabilecek nitelikler belirlenir. Bunun için sorgular analiz edilerek satırlarda sorgular sütunlarda ise nitelikleri gösterecek şekilde sorgu-nitelik matrisi elde edilir. Bu matrisin her bir hücresi, bir nitelik bir sorguda bulunuyorsa 1, bulunmuyorsa 0 değerine sahip olmaktadır. Sorguların analiz edilmesiyle elde edilen sorgu-nitelik matrisinin bu ilk ham hali, değeri 1 olan hücrelerin denk düştüğü nitelikler üzerinde dizin oluşturulabileceğini belirtir. Ancak değeri 1 olan hücrelerin denk düştüğü bütün nitelikler üzerinde dizin oluşturulması, sayıca çok fazla, faydasız ve hatta performans kaybına sebep olabilecek dizinlerin oluşturulmasına sebep olabilir. İlerleyen aşamalarda sorgu-nitelik matrisi, bazı işlemlere tabi tutulacak ve böylece en uygun dizin kümesi elde edilmeye çalışılacaktır.

Daha sonra, sorgu-nitelik matrisine benzer şekilde satırlarda sorgular sütunlarda ise nitelikleri gösterecek şekilde sorgu-frekans matrisi elde edilir. Bu matrisin her bir hücresinde bir niteliğin bir sorguda kaç kez yer aldığını belirten frekans bilgisi gösterilir. Sorgu-frekans matrisi oluşturulurken iş yükünü oluşturan sorguların frekansları çarpan olarak kullanılır. Daha sonra sorgu-frekans matrisinin sütunlarında yer alan değerler toplanarak her bir nitelik için toplam frekans değeri elde edilir. Bir niteliğin toplam frekans değeri, o niteliğin üzerinde dizin oluşturulup oluşturulmayacağını belirlemede kullanılır. Eğer bir niteliğin toplam frekans değeri belli bir eşik değerinin altında ise, o nitelik üzerinde dizin oluşturmaya uygun değildir. Bir sorgu, bir iş yükünde birçok kez gözüküyorsa, sorguda yer alan niteliklerin frekansı da yüksek olur ve dolayısıyla sorguda yer alan niteliklerin üzerinde dizin oluşturma ihtimali de yükselir.

Üzerinde dizin oluşturmaya aday nitelikleri belirlemek üzere (5.1), (5.2) ve (5.3) denklemlerinin birleşmesinden oluşan (5.4) numaralı denklem kullanılır. (5.4) denklemine göre bir nitelik, üzerinde dizin oluşturmaya aday olabilmek için (5.1) veya (5.2) denklemlerinden en az birisini sağlamalı ve ayrıca bunlara ek olarak (5.3) numaralı denklem koşulunu da her durumda sağlamalıdır.

$$\text{Frekans} > \text{Eşik1} \tag{5.1}$$

$$\text{fds/T} > \text{Eşik2} \tag{5.2}$$

$$\text{Frekans} * \text{T} > \text{Eşik3} \tag{5.3}$$

$$\big((\text{Frekans} > \text{Eşik1}) \text{ OR } (\text{fds/T} > \text{Eşik2})\big) \text{ AND } (\text{Frekans} * \text{T} > \text{Eşik3}) \tag{5.4}$$

(5.1) ve (5.3) denklemlerinde yer alan frekans değeri, üzerinde dizin oluşturulabilir her bir niteliğin iş yükünde kaç kez geçtiğini belirten değerdir. Denklemlerde yer alan T değeri ise bir tablonun satır sayısıdır. Denklemlerdeki eşik değerleri kullanıcı tarafından belirlenir.

Birinci denklem her bir niteliğin toplam frekans bilgisini kontrol eder. Bir nitelik, üzerinde dizin oluşturmaya aday olabilmek için toplam frekans değerinin belirlenen eşik değerinin üstünde olması gerekir. Buradaki amaç sorgularda sık geçmeyen niteliklerin, dizin oluşturmak için iyi bir aday olmadığı gerçeğinden hareketle toplam frekans değeri belli bir eşik değerinin altında kalan nitelikleri elemektir.

İkinci denklemde ise oluşturulacak olan dizinlerin seçici olması hedeflenmektedir. Bunun için de bir niteliğin farklı değer sayısının içinde bulunduğu tablonun satır sayısına oranı kontrol edilmektedir. Bu oran, ne kadar yüksek olursa oluşturulacak olan dizinin seçiciliği de o derece yüksek olur. Çünkü bir tablonun satırlarında yer alan bir niteliğin değerleri birbirinden ne kadar farklı ise ilgili nitelik için dizin oluşturmak o kadar anlamlı olur. Bir sorgu için de farklı değer sayısı yüksek olan nitelikler içeriyorsa, seçiliği yüksek olduğu söylenebilir.

Her durumda sağlanması beklenen üçüncü denklemde ise tabloların büyüklüğü kontrol edilmektedir. Bir niteliğin yer aldığı bir tablonun satır sayısı büyüklüğü ile niteliğin frekans değeri çarpımı kullanıcı tarafından belirlenmiş olan eşik değerinin üstünde

olması gerekir. Buradaki amaç küçük tablolarda yer alan nitelikler, eğer iş yükünde çok sık yer almıyorlarsa üzerlerinde dizin oluşturmaktan kaçınmaktır.

Denklemlerde kullanılan tabloların satır sayıları ve niteliklerin seçicilik değerleri veri tabanı sisteminin katalog bilgilerinden elde edilir. Geliştirilen dizin seçim aracı veri tabanı sisteminin katalog bilgilerine erişebilmektedir.

(5.4) denklemini sağlamayan niteliklerin yer aldığı sütunlar, sorgu-nitelik ve sorgu-frekans matrislerinden çıkarılır. Böylece geriye üzerinde dizin oluşturmaya aday nitelikler kalır.

Daha sonra, kümelenmemiş çoklu sütun içeren dizinler için arama anahtarını oluşturan niteliklerin sırasını belirlemek için sorgu-frekans matrisinde yer alan nitelikler ağırlıklandırılır. Ağırlıklandırma işleminde sorgu-frekans matrisi kullanılarak aynı satır ve sütun sayısına sahip ağırlıklandırılmış sorgu-frekans matrisi elde edilir. Ağırlıklandırma işlemi sonucunda sorgu-frekans matrisi etkilenmeden kalır. Ağırlıklandırma işlemi yapılırken, sorgu-frekans matrisinin her bir satırında bulunan her bir niteliğin “join” ifadesi içinde yer almasına, sabit bir değere eşit olmasına veya aralık ifadesi içinde yer almasına dikkat edilir. Buna göre, bir niteliğin matris içindeki frekans değeri, “join” ifadesi içinde yer alıyorsa 3, sabit bir değere eşitlik ifadesi içinde yer alıyorsa 2 ve aralık ifadesi yer alıyorsa 1 ile çarpılır. Daha sonra sütunlarda yer alan değerler toplanarak her bir niteliğin toplam ağırlık değeri elde edilir. Bu toplam ağırlık değerleri sorgu-nitelik matrisi içindeki niteliklerin sıralanmasında kullanılır. Buna göre, en büyük toplam ağırlık değerine sahip olan nitelik en solda yer alacak şekilde sorgu-nitelik matrisi yeniden düzenlenir. Elde edilen bu sorgu-nitelik matrisi artık kümeleme işlemlerinde kullanılacaktır.

Çoklu sütun dizinlerin arama anahtarını oluşturan niteliklerin sırasını belirlemek için yapılan bu ağırlıklandırma işlemi, aslında arama anahtarını oluşturan niteliklerden daha fazla yarar sağlayacağı düşünülen niteliğin, daha solda olmasına olanak sağlayan bir varsayımdır. Kullanılan ağırlık değerleri ise kümelenmemiş dizinlerde en fazla faydanın “join” ifadelerinde, en az faydanın ise aralık ifadelerinde elde edileceği düşüncesine göre belirlenmiştir.

Kümelenmiş dizinler için de aday nitelik seçimi bu aşamada yapılır. Kümelenmiş dizinler belirlenmeye çalışılırken de (5.4) denklem şartını sağlayan nitelikler kullanılır. Her bir tablo için tek bir arama anahtarı niteliğine sahip kümelenmiş dizin oluşturma önerisi sunulmaktadır. Kümelenmiş dizin önerisi sunmak için de ağırlıklandırma işlemi yapılır. Bunun için de sorgu-frekans matrisi kullanılarak aynı satır ve sütun sayısına sahip ağırlıklandırılmış sorgu-frekans matrisi elde edilir. Kümelenmiş dizinler aralık sorgusunda çok kullanışlıdırlar. Bunun için sorgu-frekans matrisinin her bir satırında yer alan her bir niteliğin frekans değeri, ilgili nitelik aralık sorgusu içinde yer alıyorsa 3, "join" ifadesi içinde yer alıyorsa 2 ve sabit bir değere eşitlik ifadesi içinde yer alıyorsa 1 ile çarpılır. Böylece ağırlıklandırılmış sorgu-frekans matrisi elde edilir. Daha sonra, her bir niteliğin toplam ağırlık değeri hesaplanır ve her bir tablonun nitelikleri kendi aralarında toplam ağırlık değerlerine göre artan sırada sıralanır. Kümelenmiş dizin seçiminde kullanılan diğer bir ölçüt de niteliklerin seçicilik değerleridir. Daha yüksek seçicilik değerine sahip bir nitelik üzerinde dizin oluşturmak daha yararlı olacağından, seçicilik değeri fazla olan bir nitelik üzerinde kümelenmiş dizin oluşturulmaya çalışılmaktadır. Bunun için her bir tablonun nitelikleri seçicilik değerlerine göre kendi aralarında artan sırada sıralanırlar. Daha sonra her bir tablo için niteliklerin ağırlık sırası ve seçicilik sırası toplanır. Her bir tablo için sıra toplamları en büyük olan nitelikler, üzerlerinde kümelenmiş dizin oluşturmaya aday olarak belirlenirler. Eğer bir tabloda yer alan birden fazla niteliğin ağırlık ve seçicilik sıraları toplamı değerleri aynı ise bu durumda ağırlık sıralaması değeri daha büyük olan nitelik seçilir. Çünkü kümelenmiş dizinler aralık ifadesi içinde sıkça yer alan nitelikler için çok kullanışlıdırlar.

Bir dosya üzerinde en fazla bir tane kümelenmiş dizin oluşturulabilir. Üzerinde kümelenmiş dizin oluşturulacak olan nitelik, genelde birincil anahtardır. Bunun sebebi ise mantıksal dizinlerdir. Mantıksal dizin, ikincil dizinlerin ana dosyaya erişimi birincil dizin üzerinden yapması ile olmaktadır. Diğer bir deyişle, ikincil dizinin kayıtlarında arama anahtarı ve kayıt işaretçisi çifti yerine, arama anahtarı ve birincil anahtar çifti saklanır. Birincil anahtar da kayıt işaretçisi gibi ana dosya kayıtlarının her birisi için biricik olan bir bilgidir. Böylece mantıksal dizin sayesinde ana dosya, dizinlerden bağımsız kılınmış olur. Çünkü ana dosyanın fiziksel yerleşimi değişince, ana dosya kayıtlarına kayıt işaretçisi ile işaret eden ikincil dizinlerin hepsinin güncellenmesi gerekmektedir. Oysa

mantıksal dizinlerde böyle bir gereksinim yoktur ve bu özellik, güncelleme sıklığı fazla olan bir dosya için performans açısından çok önemlidir.

Bu tez çalışmasında ele alınacak iş yükleri ise sadece sorgulardan oluşmakta ve güncelleme işlemi bulunmamaktadır. Bundan dolayı önerilecek kümelenmiş dizinlerin birincil anahtar üzerinde oluşturulması zorunluluğu kalkmış olmaktadır. Bu durumda seçicilik değeri birden küçük olan nitelikler üzerinde de kümelenmiş dizin oluşturulması önerilmektedir.

5.3 Kümeleme

Bu aşamada kümelenmemiş dizin adaylarını belirlemek kümeleme işlemi yapılır. Sorgu-nitelik matrisi kümeleme işleminde girdi olarak kullanılır. Sorgu nitelik matrisinin satırları, birbirine benzer olanları bir araya gelecek şekilde gruplanarak kümeler oluşturulur. Kümeleme işlemi için kmeans ve kmedoids algoritmaları kullanılır. Dizin seçim aracının oluşturacağı küme sayısı kullanıcı tarafından belirlenir.

5.4 Aday Dizinlerin Önerilmesi ve Oluşturulması

Kümeleme işlemi sonucunda elde edilen kümelerde yer alan sorguların ortak olan nitelikleri üzerinde kümelenmemiş dizin oluşturulması önerilir. Eğer bir nitelik üzerinde hem kümelenmiş hem de kümelenmemiş dizin oluşturulması öneriliyorsa, bu nitelik için kümelenmemiş dizin önerisi aday listesinden çıkarılır. Bu aşamada önerilen kümelenmiş ve kümelenmemiş dizinler oluşturularak veri tabanı sisteminin katalog bilgilerine kaydedilir.

5.5 Dizin Seçim Tekniğinin Özgünlüğü

Önerilen dizin seçim tekniği kurgulanırken Zaman vd. [3] tarafından yapılan çalışmadan ilham alınmıştır. Ancak, bu tez çalışmasında önerilen dizin seçim tekniği ile Zaman vd. [3] tarafından yapılan çalışmada kullanılan dizin seçim tekniği arasında bazı farklılıklar bulunmaktadır. Bunlardan ilki, dizin oluşturmaya aday niteliklerin belirlenmesi aşamasında kullanılan denklemlerdir. Zaman vd. [3] tarafından yapılan çalışmada, aday niteliklerin belirlenmesi için niteliklerin iş yükü içindeki frekans değerleri ve tabloların satır sayısı dikkate alınmaktadır. Bu iki kriterin mantıksal VEYA operatörü ile

birleştirilmesiyle ortaya çıkan koşulu sağlayan nitelikler aday olarak seçilebilmektedirler. Bu çalışmada ise dizin oluşturmaya aday niteliklerin belirlenmesi aşamasında kullanılan (5.4) denklemi tamamen farklı şekilde kurgulanmıştır. Buna göre (5.4) denklemi üç koşulun mantıksal operatörler ile birleşmesinden oluşmaktadır. İlk koşul, niteliklerin iş yükü içindeki frekans değerlerinin belli bir eşik değerinin üstünde olmasıdır. İkinci koşula göre ise seçicilik değerleri belli bir eşik değerinin altında olan nitelikler, üzerlerinde dizin oluşturmak için aday olamazlar. Üçüncü koşula göre ise satır sayısı küçük olan tablolarda yer alan niteliklerin frekans değeri yüksek olmadığı müddetçe üzerlerinde dizin oluşturulamaz. (5.4) denklemi, birinci ve ikinci koşullardan en az birisinin sağlanmasını, üçüncü koşulun ise her durumda sağlanmasını bekleyecek şekilde kurgulanmıştır. Ayrıca bu çalışmada farklı eşik değerleri kullanılmaktadır.

İkinci farklılık, bu tez çalışmasında belirtilen makale çalışmasından farklı kümeleme algoritmalarının kullanılmasıdır. Bu tez çalışmasında k-means ve k-medoids algoritmaları kullanılmaktadır.

Üçüncü farklılık ise iş yükünü oluşturan sorguların yürütülme şeklidir. Buna göre, belirtilen makale çalışmasında, performans testleri yapılırken bir iş yükünde yer alan sorgular teker teker ardışık olarak çalıştırılarak toplam cevap süresi ölçülmektedir. Bu tez çalışmasında ise, performans testleri yapılırken, iş yükünde yer alan sorgular hem ardışık hem de eş zamanlı olarak çalıştırılmaktadır. Ayrıca performans ölçümü yaparken, toplam cevap süresinin yanında toplam disk erişim sayısı da ölçülmektedir.

Ayrıca diğer bir fark da bu tez çalışmasında geliştirilmiş olan dizin seçim aracının, eğitimsel amaçlı bir veri tabanı sistemi olan SimpleDb üzerinde harici olarak çalıştırılmasıdır.

5.6 **Örnek Bir İş Yükü İçin Uygun Dizinlerin Seçilmesi**

Bu bölümde, otomatik dizin seçimi için önerilen hipotezi daha iyi anlatabilmek için örnek bir uygulama yapılmaktadır. Bunun için 20 tane kayıta sahip olan T1(A,B) ve 15 tane kayıta sahip olan T2(C, D, E) tabloları üzerinde oluşturulmuş 5 farklı sorgudan oluşan küçük bir iş yükü kullanılacaktır. Kullanılacak olan iş yükü W ve iş yükünü oluşturan sorguların frekans bilgileri şu şekildedir:

W = 2*Q1 + 3*Q2 + 2*Q3 + 5*Q4 + 4*Q5

W iş yükünü oluşturan 5 farklı sorgu ise Çizelge 5.1'deki gibidir.

Çizelge 5. 1 W iş yükünü oluşturan sorgular ve frekansları

Sorgu Adı	Frekans	Sorgular
Q1	2	SELECT B FROM T1, T2 WHERE A BETWEEN 1 AND 10 AND B=C AND C=5
Q2	3	SELECT A FROM T1, T2 WHERE A=D AND B BETWEEN 3 AND 5 AND D=5 AND E=20
Q3	2	SELECT A,D FROM T1, T2 WHERE B BETWEEN 3 AND 5 AND E BETWEEN 10 AND 20 GROUP BY D HAVING SUM(A)>10
Q4	5	SELECT A FROM T1 WHERE B BETWEEN 3 AND 5 AND C BETWEEN 10 AND 20 AND E=15
Q5	4	SELECT C,E FROM T1, T2 WHERE B BETWEEN 3 AND 5 GROUP BY E HAVING SUM(C)>50

Öncelikle üzerinde dizin oluşturulabilecek tüm nitelikleri belirlemek için iş yükündeki sorgular analiz edilir. Yapılan analiz sonucunda bazı matrisler oluşturulur. Bu matrislerden ilki sorgu-nitelik matrisidir. W iş yükü için sorgu nitelik matrisi Çizelge 5.2'deki gibidir.

Çizelge 5. 2 Sorgu-nitelik matrisi

Sorgular	Üzerinde Dizin Oluşturulabilir Nitelikler				
	T1.A	T1.B	T2.C	T2.D	T2.E
Q1	1	1	1	0	0
Q2	1	1	0	1	1
Q3	0	1	0	0	1

Çizelge 5. 2 Sorgu –nitelik matrisi (devamı)

Sorgular	Üzerinde Dizin Oluşturulabilir Nitelikler				
	T1.A	T1.B	T2.C	T2.D	T2.E
Q4	0	1	1	0	1
Q5	0	1	0	0	0

Daha sonra sorgu-frekans matrisi oluşturulur. Sorgu-frekans matrisinde yer alan nitelikler, üzerlerinde dizin oluşturmaya aday olabilmek için (5.4) denklemini sağlamalıdır. Yukarıdaki W iş yükü için sorgu-frekans matrisi Çizelge 5.3'deki gibidir. Sorgu-frekans matrisinde ayrıca niteliklerin toplam frekans değerleri, seçicilik değerleri ve toplam frekans değerleri ile yer aldıkları tablonun satır sayısı çarpımları da bulunmaktadır.

Çizelge 5. 3 Sorgu-frekans matrisi

Sorgular	Üzerinde Dizin Oluşturulabilir Nitelikler				
	T1.A	T1.B	T2.C	T2.D	T2.E
Q1	2	2	4	0	0
Q2	3	3	0	6	3
Q3	0	2	0	0	2
Q4	0	5	5	0	4
Q5	0	4	0	0	0
Frekans	5	16	9	6	13
Farklı değer Sayısı / T	0,1	1	1	0,8	0,93
*Frekans * T*	100	320	135	90	195

W iş yükü için eşik1 değerinin 10 olduğu varsayılsın. T1 tablosonda yer alan niteliklerin farklı değer sayıları sırasıyla 2 ve 20 olsun. T2 tablosunda yer alan niteliklerin farklı değer sayıları da sırasıyla 15, 12 ve 14 olsun. Her bir niteliğin farklı değer sayısının içinde bulunduğu tablonun satır sayısına bölünmesiyle elde edilen seçicilik değerleri Çizelge 5.3'de gösterilmiştir. Eşik2 değerinin 0.8 olduğu varsayılsın. Eşik3 değerinin de 12 satırlık bir tablonun satır sayısı çarpı eşik1 değeri olduğu varsayılırsa, eşik3 değeri 120 olur. Kullanılan eşik değerleri kullanıcı tarafından değiştirilebilir. (5.4) denklemine göre, Çizelge 5.3'deki değerler ve eşik değerleri göz önünde bulundurulursa, A ve D nitelikleri sorgu-nitelik ve sorgu frekans matrisinlerinden çıkarılırlar. Bu çıkarma işlemleri sonucunda sorgu-nitelik matrisi Çizelge 5.4'deki matrise ve sorgu-frekans matrisi de Çizelge 5.5'deki matrise dönüşür.

Çizelge 5. 4 Sadeleştirilmiş sorgu-nitelik matrisi

Sorgular	Aday Nitelikler		
	T1.B	T2.C	T2.E
Q1	1	1	0
Q2	1	0	1
Q3	1	0	1
Q4	1	1	1
Q5	1	0	0

Çizelge 5. 5 Sadeleştirilmiş sorgu-frekans matrisi

Sorgular	Aday Nitelikler		
	T1.B	T2.C	T2.E
Q1	2	4	0
Q2	3	0	3
Q3	2	0	2

Çizelge 5. 5 sadeleştirilmiş sorgu-frekans matrisi (devamı)

Sorgular	Aday Nitelikler		
	T1.B	T2.C	T2.E
Q4	5	5	4
Q5	4	0	0

Çizelge 5.4'deki matrisde yer alan nitelikler, üzerlerinde dizin oluşturulmaya adaydırlar. Daha sonra kümelenmemiş çoklu sütun dizinlerin arama anahtarını oluşturan niteliklerin sırasını belirlemek için ağırlıklandırma işlemi yapılır. Ağırlıklandırma işlemini yapmak için Çizelge 5.5'deki sorgu-frekans matrisinde yer alan nitelikler ağırlıklandırılır ve her bir niteliğin toplam ağırlığı hesaplanır. Ağırlıklandırılmış sorgu-frekans matrisi Çizelge 5.6'daki gibi olur.

Çizelge 5. 6 Kümelenmemiş dizinler için ağırlıklandırılmış sorgu-frekans matrisi

Sorgular	Aday Nitelikler		
	T1.B	T2.C	T2.E
Q1	6	10	0
Q2	3	0	6
Q3	2	0	2
Q4	5	5	8
Q5	4	0	0
Toplam Ağırlık	20	15	16

Daha sonra, sorgu-nitelik matrisinin nitelikleri, Çizelge 5.6'da yer alan ağırlıklandırılmış sorgu-frekans matrisinden elde edilen niteliklerin toplam ağırlıklık değerlerine göre soldan sağa azalacak şekilde sıralanır. Bu işlem sonucunda sorgu-nitelik matrisi Çizelge 5.7'deki gibi olur.

Çizelge 5. 7 Nitelikleri toplam ağırlıklarına göre sıralanmış sorgu-nitelik matrisi

Sorgular	Aday Nitelikler		
	T1.B	T2.E	T2.C
Q1	1	0	1
Q2	1	1	0
Q3	1	1	0
Q4	1	1	1
Q5	1	0	0

Artık, kümelenmemiş dizin önerileri elde etmek için sorgu-nitelik matrisinin son hali, kümeleme işlemi için girdi olarak kullanılacaktır. Çizelge 5.7'de yer alan sorgu-nitelik matrisinin satırları, birbirilerine benzerliklerine göre kümelere ayrılır. Kümeleme için k-means ya da k-medoids algoritmaları kullanılabilir. Bu örnekte k-means algoritması kullanılmıştır.

Kümeleme işlemi sonucunda 3 tane küme oluşturulması istendiği varsayılırsa, k-means algoritmasının çalışması sonucunda elde edilen kümeler şu şekildedir:

1.Küme: Q1, Q5

2.Küme: Q2, Q3

3.Küme: Q4

Kümeleme işleminin sonucunda elde edilen kümelere göre oluşturulması önerilen kümelenmemiş dizin önerileri şu şekildedir:

1. Küme için: {T1(B)}

2. Küme için: { T1(B), T2(E) }

3. Küme için: { T1(B), T2(E,C) }

Yukarıdaki örneğin buraya kadar olan kısmında kümelenmemiş dizin seçimi anlatılmıştır. Kümelenmiş dizinler de özellikle aralık sorguları başta olmak üzere bazı sorgular için çok avantajlı olabilir. Bunun için de her bir tablo için bir tane arama anahtarı niteliğine sahip

kümelenmiş dizin oluşturma önerisi sunulmaktadır. Kümelenmiş dizin öneride bulunurken de yine sadece (5.4) denklemini sağlayan nitelikler dikkate alınmaktadır. Öncelikle, Çizelge 5.5'de yer alan sorgu-frekans matrisine ağırlık ataması yapılır. Atama işlemi sonucunda Çizelge 5.8'deki ağırlıklandırılmış sorgu-frekans matrisi elde edilir.

Çizelge 5. 8 Kümelenmiş dizinler için ağırlıklandırılmış sorgu-frekans matrisi

Sorgular	Aday Nitelikler		
	T1.B	T2.C	T2.E
Q1	4	6	0
Q2	9	0	3
Q3	6	0	6
Q4	15	15	4
Q5	12	0	0
Toplam Ağırlık	46	21	13

İş yükünde yer alan sorgularda geçen her bir tablonun Çizelge 5.8'de yer alan nitelikleri belirlenir ve bu nitelikler her bir tablo için kendi aralarında Çizelge 5.8' de yer alan toplam ağırlık değerlerine göre sıralanır.

Bu durumda T1 ve T2 tabloları için sırasıyla Çizelge 5.9 ve Çizelge 5.10'daki ağırlık sıralaması elde edilir.

Çizelge 5. 9 T1 tablosonun niteliklerinin ağırlıklarına göre sıralanması

T1	T1.B
Ağırlık	46

Çizelge 5. 10 T2 tablosonun niteliklerinin ağırlıklarına göre sıralanması

T2	T2.E	T2.C
Ağırlık	13	21

Kümelenmiş dizin seçiminde göz önünde bulundurulucak diğer bir husus da niteliklerin seçicilik sıralamasıdır. Buna göre her bir tablonun Çizelge 5.4'deki sorgu-nitelik matrisinde yer alan nitelikleri, kendi aralarında seçicilik değerlerine göre artan sırada sıralanır. Bu durumda T1 ve T2 tabloları için sırasıyla Çizelge 5.11 ve Çizelge 5.12'deki seçicilik sıralaması elde edilir.

Çizelge 5. 11 T1 tablosonun niteliklerinin seçicilik değerlerine göre sıralanması

T1	T1.B
Seçicilik	1

Çizelge 5. 12 T2 tablosonun niteliklerinin seçicilik değerlerine göre sıralanması

T2	T2.E	T2.C
Seçicilik	0,93	1

Daha sonra, her bir tablonun niteliklerinin toplam ağırlık değeri sırası ve seçicilik değeri sırası toplanır. Bu toplam değeri, hangi nitelik için en büyükse, o nitelik üzerinde kümelenmiş dizin oluşturulması önerilir. Buna göre, T1 ve T2 tablolarında yer alan niteliklerin toplam sıra değerleri Çizelge 5.13 ve Çizelge 5.14'deki gibi olur.

Çizelge 5. 13 T1 tablosundaki niteliklerin ağırlık ve seçicilik sıraları toplamı

T1	T1.B
Toplam Sıra	2

Çizelge 5. 14 T2 tablosundaki niteliklerin ağırlık ve seçicilik sıraları toplamı

T2	T2.E	T2.C
Toplam Sıra	2	4

Bu durumda T1 tablosu için B ve T2 tablosu için de C niteliği üzerinde kümelenmiş dizin oluşturulması önerilir.

Bir nitelik üzerinde hem kümelenmiş hem de kümelenmemiş dizin oluşturma önerisi varsa bu nitelik üzerinde sadece kümelenmiş dizin oluşturulur. Buna göre en son oluşturulması önerilen kümelenmiş ve kümelenmemiş dizinler şu şekildedir:

Kümelenmemiş dizin önerileri: T2(E), T2(E,C)

Kümelenmiş dizin önerileri: T1(B), T2(C)

Son olarak önerilen dizinler oluşturulurak veri tabanı sisteminin katalog bilgilerine dizinlerle ilgili bilgiler kaydedilir ve iş yükündeki sorgular yürütülürken iyileştirici tarafından kullanılması beklenir.

BÖLÜM 6

DENEYSEL ORTAMIN HAZIRLANMASI VE PERFORMANS DEĞERLENDİRMESİ

Bu bölümde, bir iş yükünün yürütülmesine ve dizinlerin kullanımına olanak sağlayan deneysel bir ortam hazırlanmakta ve bu deneysel ortam üzerinde performans testleri yapılmaktadır. Bu tez kapsamında geliştirilen dizin seçim aracının bir iş yükü için ürettiği dizin önerileri, deneysel ortam aracılığıyla gerçeklenmektedir. İş yükünü oluşturan sorgular da deneysel ortam üzerinde yürütülmektedir. Gerçeklenen dizinler, iş yükünü oluşturan sorguların yürütülmeleri sırasında sorgu planlarına dahil edilmektedirler. Böylece, geliştirilen dizin seçim aracının ürettiği dizin önerilerinin kullanılması ile bir iş yükünün işlenmesi için gereken toplam cevap süresine ve toplam disk erişim sayısına bağlı performans ölçümleri yapılmaktadır. Performans testleri yapılırken iş yükünü oluşturan sorgular ardışık olarak veya eş zamanlı olarak yürütülmektedir.

Deneysel ortam için ilişkisel veri tabanı yönetim sistemi olan SimpleDB kullanılmakta ve SimpleDB aracılığıyla 8 tablodan oluşan "üniversite veri tabanı" isminde bir veri tabanı oluşturulmaktadır. Geliştirilen dizin seçim aracı, SimpleDB' ye harici olarak entegre edilmektedir. Daha sonra sorgulardan oluşan bir iş yükü hazırlanıp, dizin seçim aracının önerdiği dizinler kullanılarak sorgular çalıştırılmaktadır.

6.1 Deneysel Ortamın Hazırlanması

Deneysel ortamı hazırlanırken SimpleDB ilişkisel veri tabanı yönetim sistemi kullanılmaktadır. Veri tabanı olarak, üniversiteki öğrenciler ve öğrencilerin aldıkları

dersler hakkında bilgileri tutan “üniversite veri tabanı” isminde bir veri tabanı oluşturulmaktadır. Ayrıca, performans testlerinde kullanmak üzere, üniversite veri tabanı üzerinde yürütülebilen 50 farklı sorgunun rastgle sayılarda yer almasıyla oluşan 200 sorguluk bir iş yükü oluşturulmaktadır.

6.1.1 SimpleDB Veri Tabanı Yönetim Sistemi

Eğitimsel amaçlı iskelet bir veri tabanı sistemi olan SimpleDB, ticari bir VT sistemindeki bütün katmanları içermektedir. Java ile geliştirilmiş olan SimpleDB, 12 paket ve yaklaşık 5000 satır koddan oluşmaktadır. Sistem eğitimsel amaçlı olduğu için üzerinde eklentiler yapmaya müsaittir [16].

Sistem, 2 katlı istemci-sunucu mimarisine sahiptir. İstemci tarafı, sunucu ile haberleşmesine olanak sağlayan JDBC arayüzlerini ve JDBC sürücüsü gerçeklemesini içerir. Driver, Connection, Statement ve ResultSet gibi JDBC standart ara yüzleri ile programcı, sunucuya bağlandıktan sonra, sorgu gönderip, sorgu sonuçlarını tarayabilmektedir. İstemci-sunucu alt yapısı RMI protokolü ile gerçekleştirilmiştir [16]. Şekil 6.1’de bir SimpleDB istemcisinin JDBC aracalığıyla sunucu ile iletişim kurmasının örneği gösterilmiştir [28]. Ayrıca sistem eş zamanlılığı destekler, böylece birden çok istemci eş zamanlı olarak aynı dosyalara erişim sağlayabilir.

```
String qry = “select sname from student where sid = 3 ”;
Driver d = new SimpleDriver();
Connection c = d.connect(“jdbc:simpledb://localhost”, null);
Statement s = c.createStatement();
ResultSet r = s.executeQuery(qry);
while (r.next())
        System.out.println(r.getString(“sname”));
r.close();
c.commit();
```

Şekil 6. 1 İstemcinin JDBC aracılığıyla sunucuya bağlanması örneği

Sunucu tarafı, bir veri tabanı sistemi için gerekli olan işlevsellikleri tam olarak sağlar ancak verimlilik hususunu göz ardı eder. Bu sebeple, sunucu tarafına verimliliği artırmak üzere eklentiler yapılabilir. Sunucu tarafı 10 tane katman içerir. Bunlar, aşağıdan yukarıya dosya yönetimi, günlük yönetimi, tampon yönetimi, hareket yönetimi, kayıt yönetimi, katalog yönetimi, sorgu işleme, ayrıştırıcı, planlayıcı ve uzaktan erişim katmanlarıdır. Bu katmanlar mümkün olan en basit algoritmalar ile gerçekleştirilmiştir. Her katman altındaki katmanlardan hizmet alır. Sistemde katalog, günlük, erişim yapıları ve veri dosyaları olmak üzere 4 çeşit dosya vardır. Bu dosyalar rastgele erişim dosyalarıdır ve bu dosyalara blok seviyesinde erişilmektedir. Dosya yönetimi bu dosyalara erişimi sağlar. Günlük yönetimi, günlük dosyasına günlük kaydı eklenmesini ve günlük dosyasının taranması işlevini yürütür. Sistem, işletim sisteminden bağımsız olarak kendi tampon havuzuna sahiptir. Verilerin okunması ve güncelleme işlemleri diskten tampon havuzuna getirilen sayfalar üzerinde yapılmaktadır. Hareket yönetimi, eş zamanlılığı kilit protokolleri yardımıyla destekler. Veri tabanı hareketlerinin bütünlüğünü ve sürekliliğini sağlamak için veri kurtama mekanizmaları da bu katman tarafından sağlanır. Kayıt yönetimi, veri kayıtlarının disk bloklarında depolanması için metodlar sunar. Katalog yönetimi aracılığıyla tablolar hakkında istatiksel bilgiler tutulur. Sorgu işleme katmanı, select, project ve product ilişkisel cebir operatorlerinden oluşan sorgu ağacının oluşturulmasını sağlar. Ayrıştırıcı katmanı, sorguda geçen tabloları, nitelikleri ve yüklemleri çıkarır. Planlayıcı katmanı, bir SQL ifadesi için bir yürütme stratejisi oluşturur ve bunu ilişkisel bir cebir planına dönüştürür. Uzaktan erişim katmanı istemcilerle JDBC aracılığıyla haberleşmeyi sağlar [16], [28].

SimpleDB'nin erişim metodları, Btree ve adrese dayalı statik dizin yapılarıdır [16].

SimpleDB' de iki tane planlayıcı bulunmaktadır. Bunlardan ilki basit planlayıcıdır. Basit planlayıcının kullandığı algoritmaya göre "from" cümleciğinde yer alan her bir tablo için plan oluşturulur, daha sonra bu planlar ayrıştırıcıdan gelen verilerdeki tablo sırasına göre product işlemine girer. Ayrıştırıcıdan gelen verilerdeki bu tablo sırası tamamen keyfidir, dolayısıyla farklı bir tablo sırası eşdeğer bir sorgu ağacı elde edilmesine sebep olur, ancak eşdeğer sorgu ağaçları farklı sayılarda disk erişimi gerektirirler. "Product" işlemlerinden sonra "where" cümleciğinde yer alan yüklem ile "select" düğümü oluşturulur. Son olarak da "select" cümleciğinde yer alan nitelikler ile "project" düğümü oluşturulur. Basit

planlayıcı, hiçbir istatistiki bilgi kullanmadan en basit sorgu ağacını oluşturmaktadır. Basit planlayıcı, erişim yapısı olan dizinleri veya sorgu optimizasyonunda yaygın olarak bilinen sezgisel yaklaşımları kullanmamaktadır [13], [16].

SimpleDB'nin kullandığı diğer bir planlayıcı da sezgisel planlayıcıdır. Bu planlayıcının kullanıdığı sezgisel yaklaşımda tabloların "join" sırası artımlı olarak ilerler. Planlayıcı önce bir tane tablo seçer ve daha sonra "join" işlemleri tamamlanıncaya kadar birer birer tablo seçerek "join" sırasına dahil eder [13]. Tablolar join sırasına dahil edilirken sorgu ağacının maliyetini en aza indirgeme niyetiyle, en aza sayıda kayıt üreten tablolar önce seçilir. Sorgu ağacında yer alan düğümler için plan oluşturulması aşamasında dizinlerden ve çoklu tampon ile "product" işleminden yararlanılır. Bir tablo üzerinde select işlemi yaparken veya iki tablo arasındaki "join" işlemi gerçekleştirilirken B+-tree dizinlerinden yararlanılır. Sezgisel planlayıcı sadece B+-tree indekslerini içeren plan ağacı oluşturulmasına imkan vermektedir. Ayrıca sezgisel planlayıcı, iki tablo arasında "product" işlemi gerçekleştirilirken de çoklu tampon ile "product" yapılmasına izin vermektedir.

6.1.1.1 **Çoklu Tampon ile Product İşlemi**

Veri tabanı yönetim sistemlerinin işletim sisteminden bağımsız, kendi tampon organizasyonları bulunmaktadır. Buna göre, fiziksel belleğin belli bir bölümü veri tabanı işlemlerinde kullanılmak üzere veri tabanı sistemine tahsis edilir. Verilerin okunması ve güncellenmesi, diskten tampon havuzuna alınan sayfalar üzerinde yapılır. Tampon havuzunun içerdiği sayfa sayısı sınırlı olduğundan, gerektiğinde havuzdan bir sayfa kurban olarak seçilebilmektedir [16]. Bir istemci, tampon havuzu yöneticisinden istediği bloğa karşılık gelen sayfanın tampon havuzunda bağlanılmasını isteyebilir. Bağlanmış bir tampon, bir istemci tarafından kullanılıyor anlamına gelir. İstemci, bu sayfayı okuma ve yazma amaçlarıyla kullanabilir. İstemci, tampon havuzunda bağlanılmasını istediği sayfa ile işi bitince sayfanın çözülmesini isteyebilir. Çözülmüş tampon, herhangi bir istemci tarafından kullanılmayan tampondur [17]. Bu ön bilgilerden sonra product işlemi için gereken disk erişim sayısını azaltmaya yarayan çoklu tampon ile product yöntemi incelenecektir.

Bir veri tabanında yer alan T1 ve T2 tabloları arasında gerçekleştirilecek product işlemi sırasında, T2 tablosunun blok sayısı çok fazla ise bu blokların tamamı tampon havuzunda yer alamayabilir. Bu durumda yapılabilecek en iyi şey T2 tablosunu eşit parçalara ayırarak aşamalı olarak taramaktır. Önce T2 tablosunun ilk parçasını oluşturan bloklar tampon havuzuna getirilir ve T1 tablosu ile product işlemi yapılır. Ardından T2 tablosunun diğer parçaları tampon havuzuna getirilir ve T1 tablosu ile product işlemi yapılır. Böylece toplamda T1 tablosu, T2 tablosunun parça sayısı kadar okunur, T2 tablosu ise 1 kez okunmuş olur. Bu fikrin genelleştirilmiş hali çoklu tampon ile product işlemi algoritmasıdır. Bu algoritmaya göre, product işlemlerinde T2'nin her bir bölümü için T1 tablosu bir kez okunur. B_1 ve B_2 sırasıyla T1 ve T2 tablolarının blok sayılarını ve k da T2 tablosunun her bir parçasının blok sayısını göstermek üzere, T1 ve T2 tabloları arasındaki product işlemi için gereken disk erişim sayısı (6.1)'deki gibi tahmin edilebilir [13].

$$B_2 + B_1 * B_2/k \qquad (6.1)$$

Literatürde ardışık taşma(sequential flooding) olarak bilinen problemde, dosya sürekli olarak taranmaktadır. Bu problem genellikle tampon havuzundan bir sayfanın kurban seçilmesi sırasında en uzak geçmişteki sayfayı kurban seçen (LRU- least recently used) algoritmanın kullanılmasında ortaya çıkmaktadır. Ardışık taşma problemini SimpleDB havuzunda gözlemleyebilmek için şu şekilde bir test senaryosu hazırlanmıştır. Bölüm 6.1.2'de verilen üniversite veri tabanında yer alan dept tablosunun 50 tane kaydı 1 sayfada, student tablosunun 40000 kaydı ise 800 sayfada yer almaktadır. Üniversite veri tabanı üzerinde Q: <select * from dept, student> sorgusu yürütülmek istenirse, product işleminin en basit gerçekleştirimi kullanıldığı zaman bütün student sayfaları 50 kez taranacaktır. Tampon havuzunun sayfa sayısının 801'den küçük olması halinde bazı sayfalar kurban olarak seçilecektir. Kurban seçim yöntemi olarak LRU kullanıldığında eğer tampon havuzu kapasitesi 801'den küçükse, Q sorgusunun yürütülmesi, 40002 adet disk erişimi ile neticelenmektedir. Bu rakam çoklu tampon ile product yöntemi ile iyileşmektedir. Örneğin, sistemin çoklu tampon ile product yöntemine izin veren sezgisel planlayıcısı kullanılırken tampon havuzu kapasitesi 100, 400 ve 800 olarak ayarlandığında, sırasıyla 2353, 2203 ve 2003 adet disk erişimi gerekmektedir. Çoklu

tampon ile elde edilen bu sonuçlar, product işleminin en basit gerçekleştirimi kullanıdığında elde edilen 40002 adet disk erişimi ile karşılaştırılırsa, çoklu tampon yönteminin sağladığı fayda açık olarak görülecektir [16].

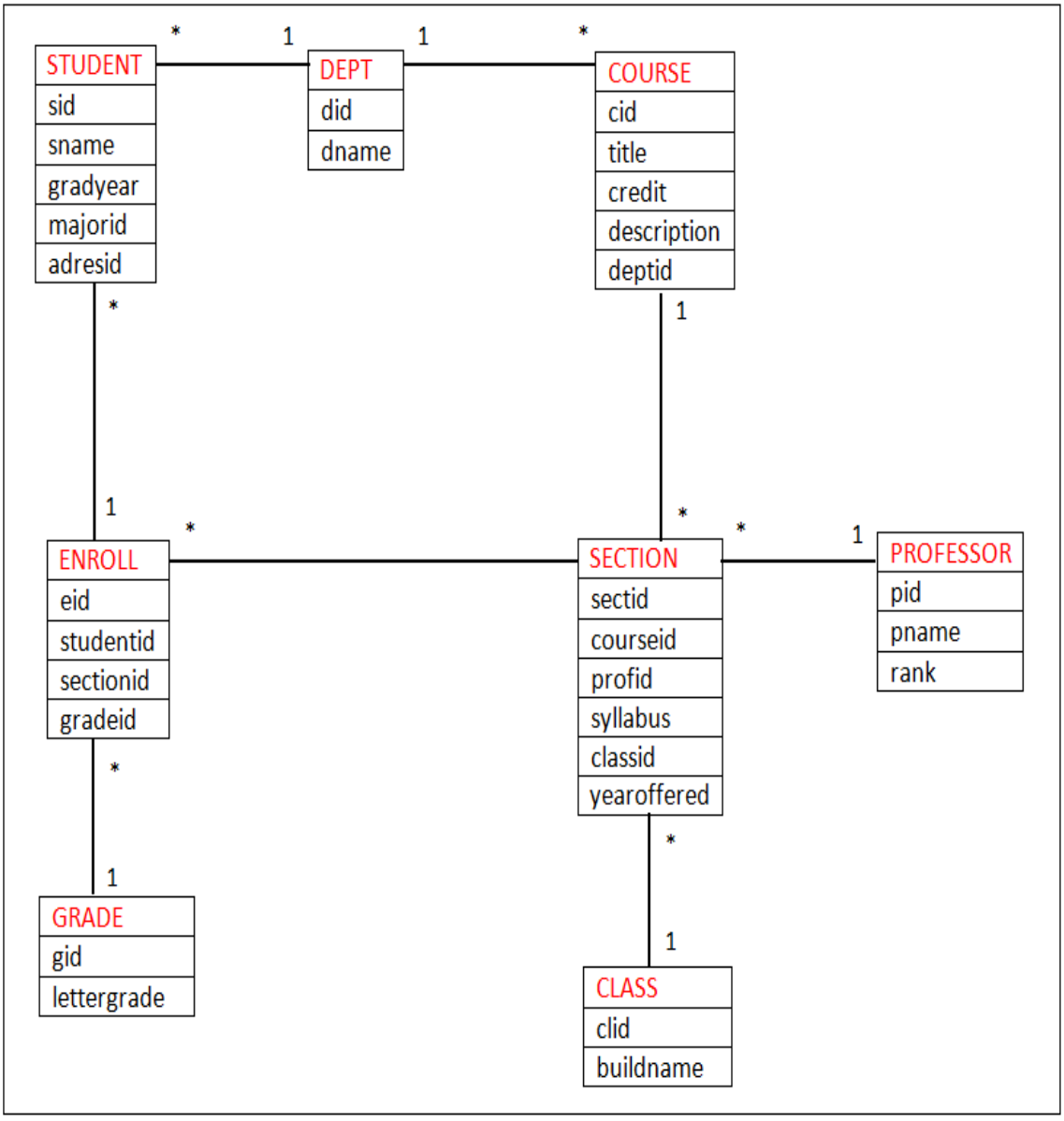
6.1.2 İlişkisel Bir Veri Tabanı: Üniversite Veri Tabanı

Üniversite veri tabanı, 8 tablodan oluşmaktadır. Bu veri tabanı, üniversiteki öğrenciler ve öğrencilerin aldıkları dersler hakkında bilgileri tutar. Dizinlerin kullanımıyla ilgili performans testlerinde kullanılmaktadır. Çizelge 6.1'de üniversite veri tabanının şeması gösterilmiştir [9].

Çizelge 6. 1 Üniversite veri tabanı şeması

STUDENT(sid, sname, gradyear, majorid)
DEPT(did, dname)
COURSE(cid, title, credit, description, deptid)
SECTION(sectid, courseid, profid, syllabus, classid, yearoffered)
PROFESSOR(pid, pname, rank)
CLASSROOM(clid, buildname)
ENROLL(eid, studentid, sectionid, gradeid)
GRADE(gid, lettergrade)

Bu veri tabanı, gerçek bir dünya nesnesi hakkında bilgileri depolamak için birçok küçük kayıtlar kullanır. Örneğin, üniversiteki her bir öğrenci, Student kaydına sahip olmanın yanında aldığı her ders için bir Enroll kaydına sahiptir. Benzer şekilde her bir ders, bir Course kaydına sahip olmanın yanında birçok Section kaydına sahiptir. Ayrıca bir derse ait bir Section kaydı da, bu ders seansının kim tarafından verildiğine dair Professor kaydı ve nerede verildiğine dair de Classroom kaydına sahiptir. Her bir bölüm de bir Dept kaydına ve birçok Student ve Course kaydına sahiptir. Şekil 6.2'de veri tabanının tuttuğu varlıkları ve bunların birbiriyle ilişkisini gösteren sınıf diyagramı gösterilmiştir [9].



Şekil 6. 2 Üniversite veri tabanı

Performans testlerinde kullanılmak üzere, deney ortamı hazırlık aşamasında, üniversite veri tabanına veri yüklemesi yapılmaktadır. Veri yüklemesi, Çizelge 6.2'deki istatistiki bilgilere uyacak şekilde yapılmaktadır. Çizelge 6.2'de T, tablo için kullanılmıştır. B(T) tablonun içerdiği blok sayısını ve R(T) de tablodaki kayıt sayısını belirtmek üzere kullanılmıştır. V(T, F) ise her bir niteliğin için farklı değer sayısını belirtmek üzere kullanılmıştır.

Çizelge 6. 2 Üniversite veri tabanı hakkında istatistikler

T	B(T)	R(T)	V(T, F)	
STUDENT	800	40000	40000	F=sid
			39960	F=sname
			40	F=gradyear
			50	F=majorid
DEPT	1	50	50	F=did, dname
COURSE	250	5000	5000	F=cid, title, description
			10	F=credit
			50	F=deptid
SECTION	1000	20000	20000	F=sectid
			5000	F=courseid
			6000	F=profid
			19800	F=syllabus
			4000	F=classid
			40	F=yearoffered
PROFESSOR	150	6000	6000	F=pid
			5980	F=pname
			20	F=rank
CLASSROOM	100	4000	4000	F=clid
			40	F=buildname
ENROLL	5000	1000000	1000000	F=eid
			20000	F=sectionid
			40000	F=studentid
			20	F=gradeid
GRADE	1	20	20	F=gid
			20	F=lettergrade

Üniversite veri tabanı tablolarında yer alan nitelikler, düzenli değer dağılımına sahiptir. Örneğin 40000 tane kayıta sahip olan Student tablosunda gradyear değerinin farklı değer sayısı 40 olarak verilmiştir. Bu durumda Student tablosunda gradyear niteliğinin her bir farklı değeri, R/V = 40000/40 olmak üzere 1000 adet bulunur.

6.1.3 İş Yükünün Oluşturulması

SimpleDB, eğitimsel amaçlı bir veri tabanı sistemi olduğu için SQL'in sadece küçük bir bölümünü kullanmaya imkan vermektedir. Sistemin ayrıştıcısı, SQL'in temel bir alt kümesini tanır. Bu alt küme, basit yüklemlere sahip "select, project, join ve groupby" sorgularından oluşmaktadır. Boolean operatörü olarak sadece "and" operatörünü kullanmaya imkan tanır. Karşılaştırma operatörü olarak sadece "=" ve mantıksal operatör olarak da "between" kullanılmasına imkan tanır. Toplama fonksiyonları olarak da "min, max, count, sum, avg" fonksiyonlarına kullanılmasına olanak sağlar. Sorgu işlemcisi, sorgu ağacını select, project, product, sort ve group by ilişkisel cebir operatörlerini kullanarak oluşturmaktadır [28].

Sistemin kullanmaya imkan verdiği SQL ifadeleri dikkate alınarak, üniversite veri tabanında üzerinde sorgulamalar yapmak üzere sorgulardan oluşan bir iş yükü oluşturulacaktır. Performans testlerinde kullanılacak olan iş yükü, birbirinden farklı 50 adet sorgunun farklı sayılarda yer almasıyla toplamda 200 adet sorgudan oluşmaktadır. Çizelge A.1'de, iş yükünü oluşturan 50 faklı sorgu ve bu sorguların iş yükü içinde bulunma sayısı olan frekansları verilmiştir.

6.1.4 Testlerde Kullanılan Bilgisayar

Testlerde sunucu ve istemciler aynı bilgisayar üzerinde çalıştırılmıştır. Kullanılan bilgisayarda Intel(R) Core i5-2450M 2.50GHz x 2 işlemci ve 4 GB bellek bulunmaktadır. İşletim sistemi olarak Windows 10 Pro yüklüdür.

6.2 Dizinlerin Seçilmesi

Performans değerlendirmesi amacıyla yapılan testler, hiç dizin kullanmadan, rastgele seçilmiş dizinlerle, k-means ve k-medoids algoritmaları yardımıyla elde edilen dizinlerle

yapılmaktadır. Geliştirilmiş olan dizin seçim aracı ile testlerde kullanılacak olan iş yükü için otomatik olarak dizin önerileri elde edilebilmektedir.

6.2.1 Dizinlerin Rastgele Olarak Seçilmesi

Üniversite veri tabanında 8 tane tablo bulunmaktadır. Bu tablolar için rastgele olarak kümelenmiş ve kümelenmemiş dizinler seçilmektedir. Tabloların sadece bir tane niteliği üzerinde kümelenmiş dizinler oluşturulacağı varsayılırsa bu durumda 28 tane kümelenmiş dizin oluşturulabilir. Kümelenmemiş dizinlerin ise tabloların bir, iki ve en fazla üç tane niteliği üzerinde oluşturulacağı varsayılırsa bu durumda 348 farklı kümelenmemiş dizin oluşturulabilir. Toplamda ise üniversite veri tabanı için 376 farklı B+ -tree dizini oluşturulabilir. Bu 376 dizin adayının arasından 25 tanesi rastgele olarak seçilmekte ve daha sonra seçilen bu dizinler gerçeklenmektedir. Rastgele seçilen dizinler Çizelge 6.3'de gözükmektedir.

Çizelge 6. 3 Rastgele seçilen dizinler

Tablo	Kümelenmiş Dizin	Kümelenmemiş Dizin
Student		{sname}, {sid, gradyear}, {majorid, gradyear, sid}, {gradyear, sid, majorid}
Dept	-	{did, dname}
Course	-	{deptid}, {title, cid}, {deptid, title, cid}, {credit, title, description}, {description, deptid, title}, {deptid, description, cid}
Section	{profid}	{sectid}, {yearoffered, profid}, {yearoffered, classid}, {syllabus, sectid, profid}, {syllabus, sectid, classid}, {courseid, yearoffered, profid}, {yearoffered, profid, syllabus}, {classid, yearoffered, profid}
Professor	-	{rank, pname} {rank, pname, pid}
Classroom	-	-
Enroll	{eid}	{eid, gradeid}
Grade	-	{lettergrade, gid}

6.2.2 Dizinlerin Dizin Seçim Aracı İle Otomatik Seçilmesi

Dizin seçim aracı yardımıyla öncelikle iş yükündeki birbirinden farklı olan sorgular ve bu sorguların frekansları belirlenmektedir. Çizelge A.1'de performans testlerinde kullanılacak olan sorgular ve bu sorguların frekansları verilmiştir. Daha sonra sorgularda geçen ve üzerlerinde dizin oluşturulabilecek nitelikleri belirlemek için sorgu-nitelik matrisi oluşturulmaktadır. Ardından sorgu-frekans matrisi oluşturulur. Sorgu frekans matrisinde, üzerinde dizin oluşturulabilir niteliklerin frekansları yer alır. Sorgu-frekans matrisindeki niteliklerin toplam frekansları hesaplanır ve ardından (5.4) denklemini sağlayan nitelikler belirlenir. Aday nitelikleri belirlemek için öncelikle (5.4) denkleminde kullanılacak olan eşik değerleri belirlenmelidir. Niteliklerin toplam frekans değerini kontrol eden eşik1 değeri, iş yükündeki sorgu sayısının %10' una denk gelecek şekilde 20 olarak ayarlanmıştır. Niteliklerin seçiciliklerini kontrol eden eşik2 değeri 0,01 olarak ayarlanmıştır. Tabloların büyüklüğünü kontrol eden eşik3 değeri de eşik1 değeri ile 250 satırlık bir tablonun büyüklüğü baz alınarak hesaplanmış ve 5000 olarak ayarlanmıştır. Bu değerler kullanıcı tarafından istenirse değiştirilebilir.

İş yükünde yer alan sorgularda bulunan niteliklerin bilgileri ve (5.4) denklemini sağlayan aday nitelikler Çizelge 6.4'de gösterilmektedir. Bundan sonraki aşamalarda Çizelge 6.4'de aday durumu "+" ile gösterilen nitelikler üzerinde kümelenmiş ve kümelenmemiş dizin seçimi yapılmaktadır.

Çizelge 6. 4 İş yükünde yer alan sorgularda bulunan niteliklerin bilgileri

Nitelik	Toplam Frekans	Seçicilik	(Toplam Frekans) * (Tablo Satır Sayısı)	Aday Durumu
Student.sid	22	1	880000	+
Student.sname	18	0,99	720000	+
Student.majorid	22	0,0012	880000	+
Student.gradyear	22	0,001	880000	+

Çizelge 6. 4 İş yükünde yer alan sorgularda bulunan niteliklerin bilgileri (devamı)

Nitelik	Toplam Frekans	Seçicilik	(Toplam Frekans) * (Tablo Satır Sayısı)	Aday Durumu
Dept.did	34	1	1700	-
Dept.dname	2	1	100	-
Course.cid	24	1	120000	+
Course.title	21	1	105000	+
Course.credit	20	0,002	100000	+
Course.description	0	1	0	-
Course.deptid	22	0,01	110000	+
Section.sectid	27	1	540000	+
Section.courseid	29	0,25	580000	+
Section.profid	31	0,3	620000	+
Section.syllabus	0	1	0	-
Section.classid	24	0,2	480000	+
Section.yearoffered	20	0,02	400000	+
Professor.pid	27	1	162000	+
Professor.pname	10	0,99	60000	+
Professor.rank	20	0,003	120000	+
Classroom.clid	24	1	96000	+
Classroom.buildname	20	0,01	80000	+
Enroll.eid	19	1	19000000	+
Enroll.studentid	25	0,04	25000000	+

Çizelge 6. 4 İş yükünde yer alan sorgularda bulunan niteliklerin bilgileri (devamı)

Nitelik	Toplam Frekans	Seçicilik	(Toplam Frekans) * (Tablo Satır Sayısı)	Aday Durumu
Enroll.sectionid	41	0,02	41000000	+
Enroll.gradeid	12	0,00002	12000000	-
Grade.gid	12	1	240	-
Grade.lettergrade	7	1	140	-

6.2.2.1 **Kümelenmiş Dizinlerin Seçilmesi**

Her tablonun bir tane niteliği üzerinde kümelenmiş dizin seçimi yapılmaktadır. Seçim yapılırken ağırlık ataması ve niteliklerin seçiciliklerine dikkat edilmektedir. Buna göre bir tablonun nitelikleri toplam ağırlıklarına ve seçiciliklerine göre artacak şekilde sıralanırlar. Her bir tablonun toplam ağırlık ve seçicilik sırası en fazla olan niteliği üzerinde kümelenmiş dizin oluşturulur. Ağırlık ataması yapmak için sorgu-frekans matrisi kullanılır ve yeni bir ağırlıklandırılmış sorgu frekans matrisi elde edilir. Test amaçlı kullanılan iş yükündeki niteliklere ağırlık ataması yapılması sonucunda elde edilen ağırlık sıralaması Çizelge 6.5'daki gibidir.

Çizelge 6. 5 Kümelenmiş dizinleri belirlemek için niteliklerin ağırlıklandırılması

Tablo	Niteliklerin Toplam Ağırlık Sıralaması
Student	gradyear:22 majorid:34 sid:39 sname:54
Course	credit:20 title:41 deptid:43 cid:51
Section	yearoffered:26 classid:45 sectid:49 courseid:50 profid:54
Professor	pname:10 rank:32 pid:50
Classroom	buildname:20 clid:45
Enroll	studentid:42 eid:43 sectionid:63

Test amaçlı kullanılan iş yükündeki niteliklerin seçicilik sıralaması Çizelge 6.6'deki gibidir.

Çizelge 6. 6 Kümelenmiş dizinleri belirlemek için niteliklerin seçicilik sıralaması

Tablo	Niteliklerin Seçicilik Sıralaması
Student	gradyear: 0,001 majorid: 0,00125 sname: 0,99 sid: 1
Course	credit: 0,002 deptid: 0,01 title: 1 cid: 1
Section	yearoffered: 0,002 classid: 0,2 courseid: 0,25 profid: 0,3 sectid: 1
Professor	rank: 0,003 pname: 0,99 pid: 1
Classroom	buildname: 0,01 clid: 1
Enroll	sectionid: 0,02 studentid: 0,04 eid: 1

Her bir tablonun niteliklerinin ağırlık ve seçicilik sıralarının toplandığında, en büyük toplam değerine sahip niteliğin seçilmesiyle elde edilen kümelenmiş dizin adayları Çizelge 6.7'de gösterilmiştir.

Çizelge 6. 7 Kümelenmiş dizin adayları

Tablo	Kümelenmiş Dizin Adayı
Student	sname
Course	cid
Section	profid
Professor	eid
Classroom	pid
Enroll	clid

6.2.2.2 **Kümelenmemiş Dizinlerin Seçilmesi**

Bu aşamada çoklu nitelik içeren dizinlerdeki niteliklerin sırasını belirlemek için sorgu-frekans matrisi üzerinde ağırlık ataması yapılır ve yeni bir ağırlıklandırılmış sorgu-frekans

matrsi elde edilir. Daha sonra sorgu-nitelik matrisindeki niteliklerin sırası, ağırlıklandırılmış sorgu-frekans matrisindeki niteliklerin toplam ağırlıklarına göre soldan sağa azalacak şekilde yeniden sıralanır. Bu aşamadan sonra, niteliklerinin sırası yeniden düzenlenmiş olan sorgu-nitelik matrisi, kümeleme işlemlerinde kullanılacaktır.

Kümeleme işlemlerinde k-means ve k-medoids algoritmaları kullanılmaktadır. Kümeleme işlemlerinde sonucunda oluşacak olan küme sayısını belirten k değeri için 3, 5, 8, 10, 20, 30, 40 ve 50 değerleri kullanılmaktadır. Elde edilen kümelerde yer alan sorguların ortak olan nitelikleri üzerinde dizin oluşturma önerisi yapılmaktadır.

K-means kümeleme algoritmasının kullanılması ile elde edilen dizin kümeleri Çizelge 6.8'daki gibidir. Elde edilen bir kümedeki dizin önerisinin arama anahtarı için zaten kümelenmiş dizin önerisi yapılmışsa, bu durumda bu nitelik üzerinde sadece kümelenmiş dizin oluşturulacaktır. Çizelge 6.8'daki dizin kümelerine Çizelge 6.7'deki her bir tablo için olan kümelenmiş dizinler de dahil edilmiştir. Diğer bir deyişle Çizelge 6.8'daki bütün dizin önerisi grupları, Çizelge 6.7'deki kümelenmiş dizin önerilerini kapsamaktadır.

Çizelge 6. 8 K-means kullanılması ile elde edilen dizinler

Arama anahtarı nitelikleri	k=3	k=5	k=8	k=10	k=20	k=30	k=40	k=50
{sname} (kümelenmiş)	+	+	+	+	+	+	+	+
{majorid}				+	+	+	+	+
{gradyear}					+	+	+	+
{sid, majorid}						+	+	+
{sid, gradyear}				+		+	+	+
{majorid, sname}						+	+	+
{gradyear, sname}							+	+

Çizelge 6. 8 K-means kullanılması ile elde edilen dizinler (devamı)

Arama anahtarı nitelikleri	k=3	k=5	k=8	k=10	k=20	k=30	k=40	k=50
{sid, majorid, sname}								+
{cid} (kümelenmiş)	+	+	+	+	+	+	+	+
{title}						+	+	+
{credit}		+		+		+	+	+
{deptid}			+		+	+	+	+
{cid, title}					+	+	+	+
{deptid, credit}						+		+
{deptid, title}								+
{cid, deptid, title}							+	+
{sectid}				+		+	+	+
{profid} (kümelenmiş)	+	+	+	+	+	+	+	+
{courseid}	+	+	+	+	+	+	+	+
{classid}			+	+	+	+	+	+
{yearoffered}			+	+	+	+	+	+
{profid, yearoffered}				+		+	+	+
{profid, courseid}					+	+	+	+
{courseid, sectid}							+	+
{courseid, yearoffered}							+	+
{pid} (kümelenmiş)	+	+	+	+	+	+	+	+
{pname}					+		+	+
{rank}					+	+	+	+
{pid, rank}					+	+	+	+
{clid} (kümelenmiş)	+	+	+	+	+	+	+	+

Çizelge 6. 8 K-means kullanılması ile elde edilen dizinler (devamı)

Arama anahtarı nitelikleri	k=3	k=5	k=8	k=10	k=20	k=30	k=40	k=50
{buildname}			+		+	+	+	+
{eid} (kümelenmiş)	+	+	+	+	+	+	+	+
{scid}					+	+	+	+
{stid}				+		+	+	+
{scid, stid}							+	+

K-medoids kümeleme algoritmasının kullanılması ile elde edilen dizin kümeleri Çizelge 6.9'daki gibidir. Çizelge 6.9'daki dizin önerisi kümeleri de Çizelge 6.8'daki dizin önerisi kümelerine benzer şekilde oluşturulmuştur.

Çizelge 6. 9 K-medoids kullanılması ile elde edilen dizinler

Arama anahtarı nitelikleri	k=3	k=5	k=8	k=10	k=20	k=30	k=40	k=50
{sname} (kümelenmiş)	+	+	+	+	+	+	+	+
{majorid}	+		+			+	+	+
{gradyear}					+	+	+	+
{sid, majorid}					+	+	+	+
{sid, gradyear}			+		+	+	+	+
{majorid, sname}		+		+		+	+	+
{gradyear, sname}								+
{sid, majorid, sname}								+
{cid} (kümelenmiş)	+	+	+	+	+	+	+	+
{title}					+	+	+	+
{credit}					+	+	+	+
{deptid}	+	+		+	+	+	+	+

Çizelge 6. 9 K-medoids kullanılması ile elde edilen dizinler (devamı)

Arama anahtarı nitelikleri	k=3	k=5	k=8	k=10	k=20	k=30	k=40	k=50
{cid, title}		+				+	+	+
{deptid, credit}								+
{deptid, title}					+		+	+
{cid, deptid, title}						+	+	+
{sectid}				+	+	+	+	+
{profid} (kümelenmiş)	+	+	+	+	+	+	+	+
{courseid}		+		+	+	+	+	+
{classid}			+	+	+	+	+	+
{yearoffered}					+	+	+	+
{profid, yearoffered}					+		+	+
{profid, courseid}					+		+	+
{courseid, sectid}						+	+	+
{courseid, yearoffered}					+	+	+	+
{pid} (kümelenmiş)	+	+	+	+	+	+	+	+
{pname}			+		+		+	+
{rank}							+	+
{pid, rank}					+		+	+
{clid} (kümelenmiş)	+	+	+	+	+	+	+	+
{buildname}			+	+	+	+	+	+
{eid} (kümelenmiş)	+	+	+	+	+	+	+	+
{scid}				+	+	+	+	+
{stid}			+		+	+	+	+
{scid, stid}						+	+	+

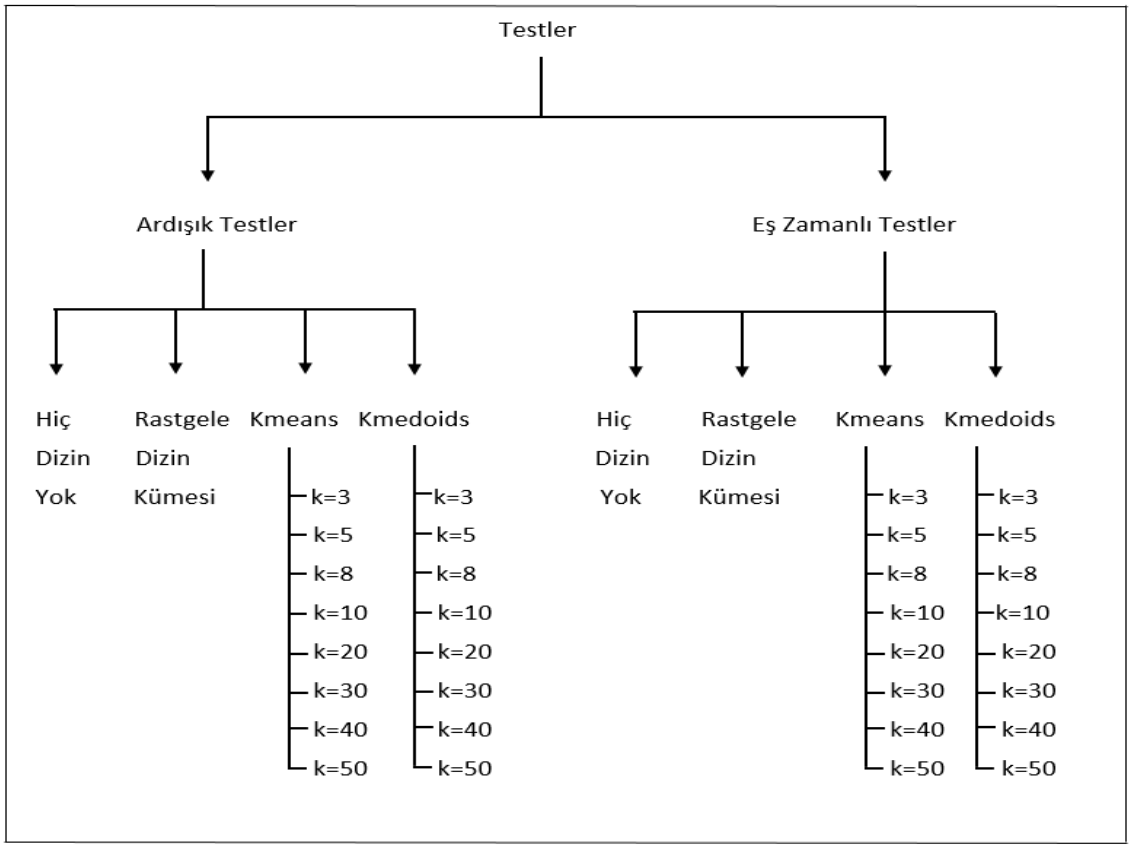
6.3 **Performans Testleri**

Performans testlerine başlamadan önce sunucu hazır hale getirilmektedir. Bunun için sunucunun tampon havuzu büyüklüğü ve planlayıcı türü parametreleri ayarlanmaktadır. Testler yapılırken istemciler sunucuya bağlanmakta ve sunucu üzerinde sorgulama yapılmaktadır. Çizelge A.1'de verilen iş yükünü oluşturan 200 sorgunun her birisi, bir istemcinin sunucuya bağlanması ile sorgulanmaktadır. SimpleDB, eşzamanlılığı desteklediği için istemciler sunucuya ardışık veya eş zamanlı olarak da bağlanabilmektedirler. Testler, iş yükünü oluşturan sorguların ardışık ve eş zamanlı olarak yürütülmesi ile gerçekleştirilmiştir. Yapılan testlerde iş yükü, hiç dizin kullanmadan, rastgele belirlenmiş dizinlerle, k-means ve k-medoids algoritmaları yardımıyla elde edilen dizin önerileriyle işlenmiştir. Testler sonucunda performans değerlendirmesi iş yükünün işlenmesi için gereken toplam disk erişim sayısı ve harcanan toplam zaman üzerinden yapılmıştır.

Testlerde kullanılan iş yükü sadece sorgulardan oluşmasına rağmen sorguların yürütülmesi sırasında yazma işlemleri de meydana gelebilmektedir. Bunların sebebi günlük kayıtlarının yazılması ve dizin kullanılmadığı durumlarda çoklu tampon ile "product" yapılmasıdır.

Sunucu hazır hale getirilirken tampon havuzu büyüklüğü ayarlanmaktadır. Bir sorgunun çalışmasında gerekli disk erişim sayısını etkileyen faktörlerden birisi de tampon havuzu büyüklüğüdür. Tampon havuzu büyüklüğü arttıkça aranılan bir sayfanın havuzda bulunma olasılığı artacak dolayısıyla disk erişimene gerek kalmayacaktır. Bu yüzden tampon havuzu büyüklüğü bir iş yükünün işlenmesi sonucundaki toplam disk erişim sayısına doğrudan etki etmektedir. Yapılan testlerde, iş yükünü oluşturan sorgular ardışık olarak sorgulanırken tampon havuzu büyüklüğü 400, eş zamanlı sorgulamada ise 100000 olarak ayarlanmıştır. Bu sayıların altında tampon havuzu yetersizliği hatası karşılaşılmıştır.

Sunucu hazırlanırken diğer parametre olan planlayıcı türü sezgisel planlayıcı olarak ayarlanmaktadır. Yapılan performans testlerinin şeması Şekil 6.3'deki gibidir. Şekil 6.3'deki k değeri, k-means ve k-medoids algoritmalarının oluşturacağı küme sayısını belirtmektedir.



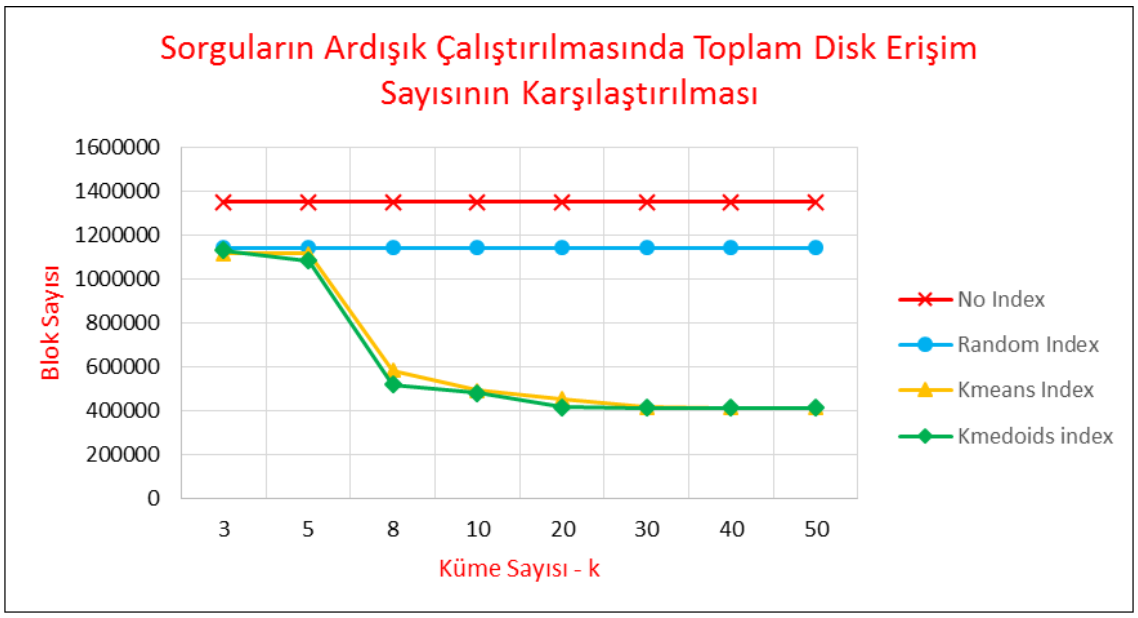
Şekil 6. 3 Performans değerlendirmesi amacıyla yapılan testler

6.3.1 Sorguların Ardışık Çalıştırılması İle Yapılan Testler

Sorguların ardışık çalıştırılmasında, istemciler art arda sunucuya bağlanmaktadır. Buna göre bir istemci, sunucu ile bağlantı kurar ve SQL sorgusunu sunucuya gönderir, sorgulama işlemi sunucu üzerinde tamamlandıktan ve sonuçlar istemciye ulaştıktan sonra istemcinin sunucu ile bağlantısı kopartılır. Ardından yeni bir istemci sorgulama yapmak üzere sunucuya bağlanır. Her yeni istemci sunucuya bağlandığında sunucunun tampon havuzunu oluşturan sayfaları temizlenir. Bu durumda, iki tane istemci art arda aynı sorguyu çalıştırmak üzere sunucuya bağlansa bile iki sorgu için de aynı sayıda disk erişimi gerçekleştirilir.

200 sorguluk iş yükünü oluşturan sorguların ardışık çalıştırılması ile yapılan testlerde iş yükünün işlenmesi için gereken toplam disk erişim sayısı ve toplam cevap süresi gözlenmiştir. Bunun için iş yükü, hiç dizin kullanılmadan, rastgele seçilmiş dizin kümesi ve kümeleme algoritmaları yardımıyla belirlenen dizin kümeleriyle işlenmiştir. Yapılan

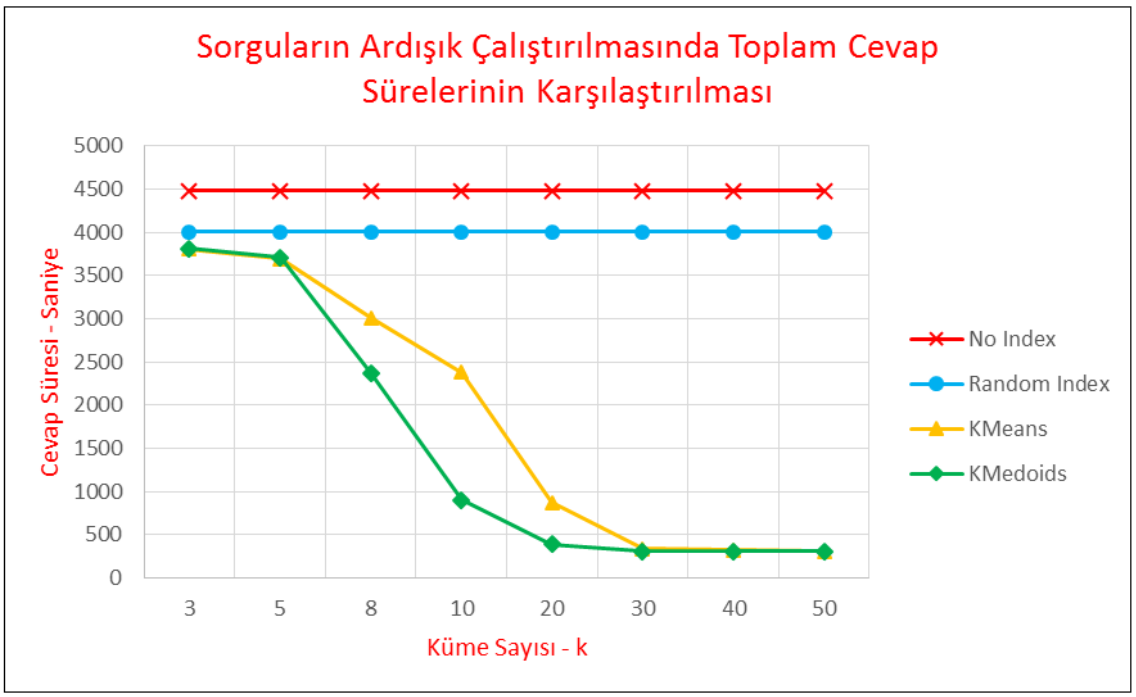
testler sonucunda elde edilen toplam disk erişim sayısı grafiği Şekil 6.4'de, toplam cevap süresi grafiği ise Şekil 6.5'de gösterilmiştir.



Şekil 6. 4 Sorguların ardışık çalıştırılmasında toplam disk erişim sayısının karşılaştırılması

Şekil 6.4'deki disk erişim sayısını gösteren grafiğe bakıldığında 1353963 adet disk erişimi ile en yüksek rakamın hiç dizin kullanılmayan senaryoda olduğu görülmektedir. Bu sayı sezgisel planlayıcı yerine sistemin basit planlayıcısı kullanılsaydı çok daha yüksek olurdu. Rastgele belirlenen dizin kümesi ile toplam disk erişim sayısının 1144750 olduğu görülmüştür. Rastgele belirlenen dizin kümesi ile elde edilen sonucun hiç dizin kullanılmayan senaryoya göre daha iyi olduğu görülmektedir. K-means ve k-medoids algoritmaları yardımıyla elde edilen dizinlerle yapılan testlerde küme sayısının 3 olduğu durumda k-means için 1117789 ve k-medoids için 1130792 adet disk erişimi olduğu görülmektedir. Küme sayısının artmasıyla bu değerlerin hızlı bir şekilde düştüğü görülmektedir. Çünkü küme sayısı arttıkça küme içi benzerlik artmakta ve dolayısıyla bir küme için daha fazla ve daha kullanışlı dizin önerisinde bulunulmaktadır. Aradaki fark az da olsa, k-medoids algoritmasının önerdiği dizinlerle daha az disk erişiminin yapıldığı grafikten görülmektedir. Küme sayısının 30 olmasıyla birlikte her iki algoritmanın da önerdiği dizinlerin kullanılmasıyla elde edilen sonuçlar birbirine çok yaklaşmaktadır. Küme sayısı 30 olduğunda k-means için disk erişim sayısı 414966 ve k-medoids için 413522 olmaktadır. En iyi sonuç ise küme sayısının 50 olduğu durumda hem k-means

hem k-medoids için 411486 adet disk erişimi ile görülmektedir. En iyi sonucun küme sayısının 50 olduğu durumda görülmesi, iş yükünün 50 farklı sorgudan oluşmasıyla ilişkilidir. Çünkü küme sayısı 50 olunca her bir kümeye farklı bir sorgu düşmekte ve böylece önerilen dizin sayısı maksimum olmaktadır. Kümeleme algoritmalarının kullanılması ile elde edilen en iyi sonuca bakıldığında, hiç dizin kullanılmayan duruma göre %69, rastgele dizin kümesiyle elde edilen sonuca göre ise %64 iyileşme sağlandığı görülmektedir.



Şekil 6. 5 Sorguların ardışık olarak çalıştırılmasında toplam cevap süresinin karşılaştırılması

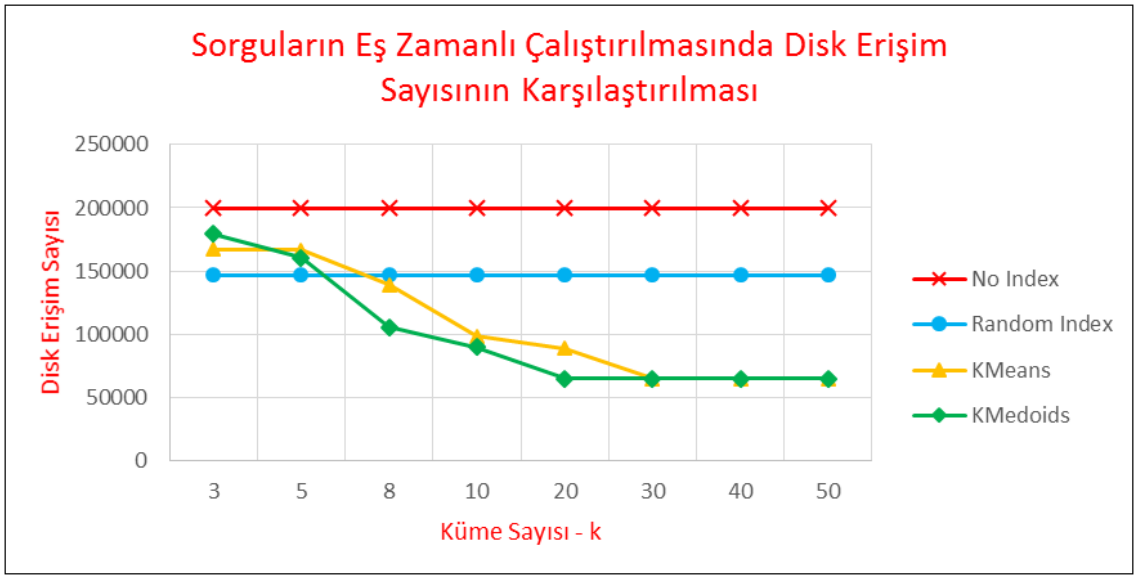
Sorguların ardışık çalıştırılmasıyla oluşan Şekil 6.5'deki grafik incelendiğinde hiç dizin kullanılmayan durumda iş yükünün 4480 saniyede işlendiği, rastgele belirlenen dizin kümesi ile ise 4010 saniyede işlendiği görülmektedir. Kümeleme algoritmaları yardımıyla elde edilen dizin önerilerinin kullanılmasıyla ise Şekil 6.4'deki grafiğe benzer şekilde, küme sayısı arttıkça toplam cevap süresinin azaldığı görülmektedir. Bu durum da küme sayısı arttıkça dizinlerin sayısının ve kullanışlılıklarının artmasıyla ilgilidir. Hem k-means hem de k-medoids algoritmaları için en iyi sonucun küme sayısının 50 olduğu durumda 303 saniye ile olduğu görülmektedir. Küme sayısının 5 ve 30 arasındaki değerlerinde k-medoids algoritması ile edilen dizin önerileri ile daha iyi performans elde edilmiştir. Kümeleme algoritmalarının kullanılması ile elde edilen en iyi toplam cevap süresine

bakıldığında, hiç dizin kullanılmayan duruma göre %93, rastgele dizin kümesiyle elde edilen sonuca göre ise %91 iyileşme sağlandığı görülmüştür.

6.3.2 Sorguların Eş Zamanlı Çalıştırılması İle Yapılan Testler

Sorguların eş zamanlı çalıştırılmasında, istemciler aynı anda sunucuya bağlanmaya çalışmaktadırlar. Sunucu ile bağlantı kuran istemci, SQL sorgusunu sunucuya gönderir, sunucu üzerinde sorgulama işlemi yapılır ve sorgulama işlemi tamamlandıktan sonra sonuçlar istemciye geri döndürülür. Ardışık çalıştırmadan farklı olarak yeni bir istemci sunucuya bağlandığında tampon havuzu temizlenmez dolayısıyla farklı istemciler erişmek istedikleri disk bloklarını tampon havuzunda diske gitmeden bulabilme şansları artar.

Sorguların eş zamanlı çalıştırılmasıyla yapılan performans testlerinde de ardışık çalıştırmada kullanılan dizinler kullanılmıştır. Yapılan testler, ardışık çalıştırmaya benzer şekilde, iş yükünün hiç dizin kullanılmadan, rastgele seçilmiş dizinlerle ve kümeleme algoritmaları yardımıyla seçilmiş dizinlerle işlenmesi ile yapılmıştır. Yapılan testler sonucunda elde edilen toplam disk erişim sayısı grafiği Şekil 6.6'da, toplam cevap süresi grafiği ise Şekil 6.7'de gösterilmiştir.

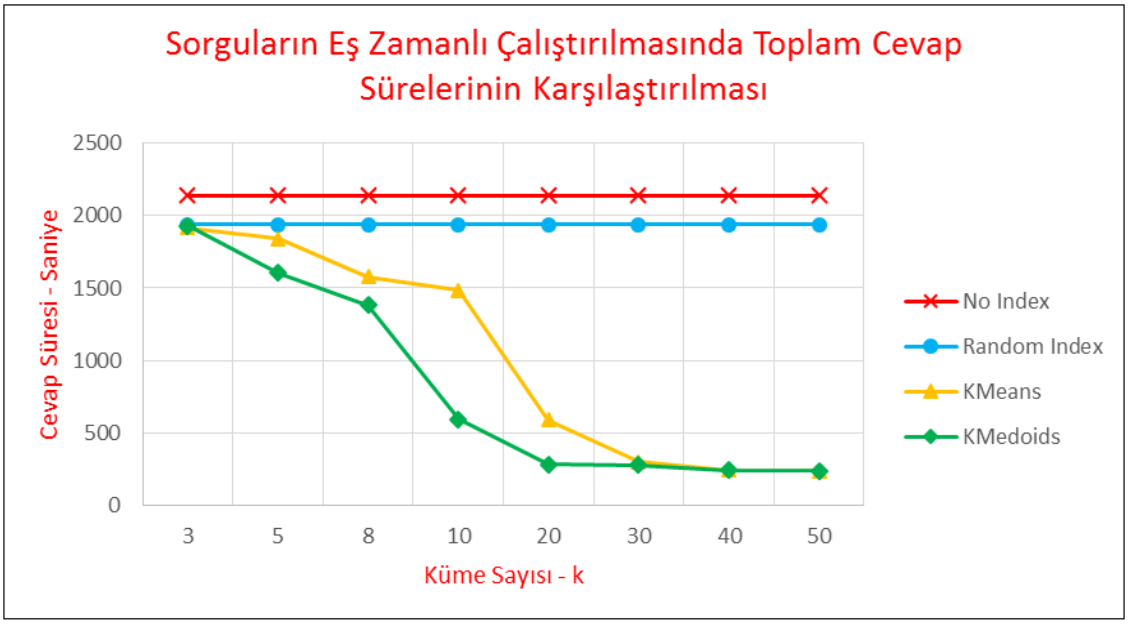


Şekil 6. 6 Sorguların eş zamanlı çalıştırılmasında toplam disk erişim sayısının karşılaştırılması

Sorguların eş zamanlı çalıştırılması neticesinde toplam disk erişim sayılarının ölçülüp karşılaştırılmasıyla elde edilen Şekil 6.6'daki grafiğe bakıldığında en yüksek disk

erişiminin 199776 ile hiç dizin kullanılmayan senaryoda olduğu görülmektedir. Rastgele belirlenen dizin kümesinin kullanılmasıyla da 146996 adet disk erişimi yapılmıştır. Rastgele dizin kümesiyle elde edilen sonuç k-means ve k-medoids algoritmalarının küme sayısının 3 ve 5 olduğu durumlara göre daha iyi olduğu görülmektedir. Bu durumun oluşmasında rastgele seçilmiş dizinlerin sayısı etkili olmuştur. Üniversite veri tabanı için hesaplanan 376 tane dizin ihtimali arasından rastgele seçilen 25 tane dizin sayısının, tüm ihtimallere göre oranının yüksek olduğu söylenebilir. Kümeleme yöntemlerinde ise küme sayısı 3 ve 5 olarak belirlendiğinde, yaklaşık 8 adet dizin seçimi yapılmıştır. Bunun yanında küme sayısı arttıkça k-means ve k-medoids yardımıyla elde edilen dizin önerilerinin kullanılmasıyla elde edilen sonucun iyileştiği görülmektedir. Çünkü, küme sayısı arttıkça kümelerin kendi içindeki benzerliklerinin artmakta ve dolayısıyla bir küme içindeki sorguların dizinlenebilir ortak niteliklerinin sayısı da artmaktadır. Dolayısıyla önerilen dizinler birden çok sorgu için yararlı olmaktadır.

Hem k-means hem de k-medoids için en az disk erişim sayısının küme sayısının 40 ve 50 olduğu durumlarda 64703 adet disk erişimi ile olduğu görülmüştür. İş yükünün eş zamanlı işlenmesinde, kümeleme algoritmalarının kullanılması ile elde edilen en iyi sonuca bakılırsa, hiç dizin kullanılmayan duruma göre %68, rastgele dizin kümesiyle elde edilen sonuca göre ise %55 iyileşme sağlandığı görülmektedir.



Sorguların ardışık çalıştırılmasıyla oluşan Şekil 6.7'deki grafik incelendiğinde, iş yükünün hiç dizin kullanılmayan durumda 2138 saniyede işlendiği, rastgele belirlenen dizinlerle ise 1938 saniyede işlendiği görülmektedir. Kümeleme algoritmaları yardımıyla elde edilen dizin önerilerinin kullanılmasıyla ise toplam cevap süresinin keskin bir şekilde azaldığı görülmektedir. Çünkü küme sayısı arttıkça kümeleme işlemi sırasında oluşan kümelerde yer alan sorguların benzerlikleri artmaktadır, dolayısıyla sorguların dizinlenebilir ortak niteliklerinin sayısı da artmaktadır. Böylece hem daha fazla sorguya fayda sağlayan kullanışlı dizinler seçilmekte hem de seçilen dizinlerin sayısı artmaktadır.

Küme sayısı, 3 ve 30 arasında iken k-medoids algoritması yardımı ile daha iyi sonuçlar elde edilmiştir. Bu durumun oluşmasında, k-means algoritmasının aykırı değerlere karşı daha duyarlı olmasının etkili olduğu söylenebilir. Her iki algoritmanın kullanımı ile elde edilen sonuçlardaki değişim, küme sayısı 30 olduktan sonra azalmış ve sonuçlar birbirine iyice yaklaşmıştır. En iyi sonuçlar, hem k-means hem de k-medoids için küme sayısı 50 iken 233 saniye olarak görülmüştür. Kümeleme algoritmalarının kullanılması ile elde edilen en iyi toplam cevap süresine bakıldığında, hiç dizin kullanılmayan duruma göre %89, rastgele dizin kümesiyle elde edilen sonuca göre ise %86 iyileşme sağlandığı görülmüştür. Rastgele seçilmiş dizin kümesinin eleman sayısı yüksek olmasına rağmen, performansa fazla katkıda bulunmamasında kullanışlı dizinlerin seçilmemesi gösterilebilir. Örneğin 1 saydan oluşan dept tablosunun nitelikleri üzerinde veya iş yükündeki sorguların hiçbirinin "where" koşul ifadesi içinde yer almamasına rağmen section tablosunun "syllabus" niteliği üzerinde bile oluşturulması için rastgele dizin önerisinde bulunulmuştur.

BÖLÜM 7

SONUÇ VE ÖNERİLER

Bu tez kapasamında, sorgulardan oluşan bir iş yükünün işlenmesinde performansı artırmaya yönelik, otomatik olarak dizin seçimi yapmaya yarayan bir araç geliştirilmesi hedeflenmiştir. Dizin seçim probleminin NP-tam olması gerçeği de göz önünde bulundurulduğunda bir iş yükü için en uygun dizin kümesinin bulunması çok zordur, bu yüzden otomatik araçlar bir gerekliliktir. Geliştirilen otomatik dizin seçim aracı ile veri tabanı yöneticisinin yükünü hafifletmek, bir iş yükünü oluşturan sorguların cevaplanması için gereken toplam zamanın ve toplam disk erişim sayısının azaltılması hedeflenmiştir.

Dizin seçim aracı gerçekleştirilirken k-means ve k-medoids kümeleme algoritmalarından yararlanılmıştır. İş yükünü oluşturan sorgular benzerliklerine göre gruplara ayrılmış ve daha sonra her bir kümede yer alan sorguların ortak nitelikleri üzerinde dizin oluşturulması önerisinde bulunulmuştur. Böylece önerilen dizinlerin birçok sorgu için kullanışlı olması hedeflenmiştir.

5. bölümde bir iş yükü, hiç dizin kullanmadan, rastgele seçilmiş dizinler kullanarak ve kümeleme algoritmaları yardımıyla elde edilen dizin önerilerinin kullanılması ile işlenerek performans testleri yapılmıştır. Testlerde iş yükünü oluşturan sorgular ardışık ve eş zamanlı olarak çalıştırılmıştır. Bu testlerde toplam disk erişim sayısı ve toplam cevap süresi değişimleri izlenmiştir.

Yapılan testlerde kümeleme algoritmaları yardımıyla elde edilen sonuçların hiç dizin kullanılmayan ve rastgele belirlenen dizinlerle elde edilen sonuçlara göre daha iyi olduğu görülmüştür. Ayrıca genel olarak k-medoids algoritmasının kullanılmasıyla elde edilen

sonuçların da k-means kullanımına göre daha başarılı olduğu görülmüştür. Bu sonucun elde edilmesinde k-medoids algoritmasının daha iyi kümeler oluşturması gösterilebilir. Her iki algoritma için de küme sayısı arttıkça kümelerdeki sorgular birbirine daha çok benzemeye başlamakta ve dolayısyla bir kümede yer alan sorguların daha çok ortak nitelikleri olmaktadır. Bunun sonucunda da dizin seçim aracı daha çok ve daha kullanışlı dizinler önerileri yapmaktadır. Küme sayısı 50 olduğunda k-means ve k-medoids algoritmasının önerdiği dizinler aynı olmaktadır. Çünkü küme sayısı 50 iken, testlerde kullanılan 200 sorguluk iş yükünü oluşturan 50 farklı sorgunun her biri bir kümeye düşmektedir. Bu durumda da her iki algoritmanın ürettiği sonuçlar aynı olmaktadır. Böylece dizin önerisi sayısı artmakta ve performans da en iyi seviyeye ulaşmaktadır. Bu yüzden en iyi sonuç hem k-means hem de k-medoids algoritmaları için küme sayısı 50 iken görülmektedir.

İş yükünü oluşturan sorguların yürütülmesinde kümeleme algoritmaları yardımıyla elde edilen dizin önerilerinin kullanılmasıyla hem toplam disk erişim sayısı hem de toplam cevap süresinde hiç dizin kullanılmayan senaryonun sonuçlarına göre iyileşme olduğu gözlenmiştir. Bu iyileşme, sorguların ardışık çalıştırılmasında disk erişimi sayısında %69, toplam cevap süresinde ise %93 olarak görülmüştür. Sorguların eş zamanlı çalıştırılmasında ise toplam disk erişimi sayısında %68, toplam cevap süresinde ise %89 oranında iyileşme olduğu görülmüştür.

İş yükünü oluşturan sorguların eş zamanlı olarak çalıştırılmasındaki toplam disk erişim sayısının ve toplam cevap süresinin, sorguların ardışık çalıştırılmasına göre daha az oldukları görülmüştür. Bu duruma sebep olarak ardışık çalıştırmada her yeni istemci sunucuya bağlandığında tampon havuzunun temizlenmesi gösterilebilir. Eş zamanlı çalıştırmada ise tamponda bulunan sayfalar ortak olarak kullanılabilmektedir.

Zaman vd. tarafından [3] yapılan çalışmada, test amaçlı kullanılan iş yükü sadece ardışık olarak işlenmiştir. Yaptıkları testlerde sadece toplam cevap süresini ölçmüşlerdir. Geliştirdikleri araç ile yaptıkları performans testlerinde hiç dizin kullanılmayan senaryoya göre elde ettikleri en iyi sonuç %79.95' dir. Bu tez çalışmasında ise ele alınan iş yükü, hem ardışık hem de eş zamanlı olarak işlenmiştir. Ayrıca bu çalışmada yapılan testler ile iş yükünün işlenmesi için gereken hem toplam disk erişim sayısı hem de toplam cevap

süresi ölçülmüştür. İş yükünün oluşturan sorguların ardışık olarak işlenmesinde toplam cevap süresinin ölçülmesiyle elde edilen en iyi sonuç, hiç dizin kullanılmayan senaryonun sonucuna göre %93 iyileşme göstermiştir. Bu sonuç da Zaman vd. tarafından [3] elde edilen sonuca göre daha iyidir.

Yapılan testlerde sezgisel planlayıcının kullanılması, aslında dizinlerin kullanılması durumundaki performans değişimini tam olarak görmeyi engellemektedir. Çünkü sezgisel planlayıcı, dizinlerin olmadığı durumlarda iki tablo arasındaki "product" veya "join" işlemi gerçekleştirilirken çoklu tampon ile "product" metodundan yararlanabilmektedir. Bu durum da dizinlerin hiç kullanılmadığı testlerin sonuçlarının bile nispeten iyi olmasını sağlamaktadır. Sistemdeki basit planlayıcı ise yürütme planına dizinleri dahil etmemekte ve sorgu optimizasyonu için hiçbir sezgisel yaklaşımdan yararlanmamaktadır. Bu durum da basit planlayıcının kullanılması halinde en basit sorgularda bile saatlerce dosya taraması yapmayı gerektirir. Bu yüzden basit planlayıcı kullanılmamıştır.

İleriye yönelik hedeferden ilki testlerde kullanılan iş yüküne, güncelleme içeren sorguların da dahil edilmesidir. Güncellemeler, dizinlere ek yük getirecek dolayısıyla problemin karmaşıklığı daha da artacaktır. Bu durumda ele alınan iş yükü için otomatik olarak dizin seçimi yapılmasının önemi artacaktır. Hâlihazırda SimpleDB kıstlı bir SQL setini desteklemektedir. İleriye yönelik hereflerden bir diğeri de SimpleDB'nin desteklediği SQL seti genişletilerek, TPC değerlendirme deneyleri (benchmark) aracılığıyla da performans testlerinin yapılmasıdır. Son olarak bu tez kapsamında geliştirilen dizin seçim aracı, eğitimsel olmayan gerçek bir veri tabanı sisteminde (Apache Derby, PostgreSQL gibi) denenebilir. Geliştirilen dizin seçim aracı harici bir araçtır dolayısıyla başka veri tabanı sistemleri üzerinde de denenip ürettiği sonuçlar sınanabilir.

KAYNAKLAR

[1] Choenni, S., Blanken, H. ve Chang, T., (1993). "Index Selection in Relational Databases", Fifth International Conference on Computing and Information, 27-29 May 1993, Ontario, Canada, 491-496.

[2] Bruno, N., (2011). Automated Physical Database Design and Tuning, First Edition, CRC Press, New York.

[3] Zaman, M., Surabattula, J. ve Gruenwald L., (2004). "An Auto-Indexing Technique for Databases Based on Clustering", Proeedings of the 15th International Workshop on Database and Expert System Applications, 3-3 September 2004, Zaragoza, Spain, 776-780.

[4] Graefe, G. ve Kuno, H., (2010). "Self-selecting, Self-Tuning, Incrementally Optimized Indexes", Proceedings of the 13th International Conference on Extending Database Technology, 22-26 March 2010, Lausanne, Switzerland, 371-381.

[5] Ameri, P., Meyer, J. ve Streit, A., (2015). "On a New Approach to the Index Selection Problem Using Mining Algorithms", IEEE International Conference on Big Data, 29 October – 1 November 2015, Santa Clara, CA, USA, 2801-2810.

[6] Aouiche, K., Darmont, J. ve Gruenwald L., (2003). "Frequent Itemsets Mining for Database Auto-Administration", Seventh International Database Engineering and Applications Symposium, 18-18 July 2003, Hong Kong, China, 98-103.

[7] Pedrozo, W.G. ve Vaz, M.S.M.G., (2014). "A Tool for Automatic Index Selection in Databse Management Systems", International Symposium on Computer, Consumer and Control, 10-12 June 2014, Taichung, Taiwan, 1061-1064.

[8] Schnaitter, K., Abiteboul, S., Milo, T. Ve Polyzotis, N., (2006). "COLT: Continuous On-Line Database Tuning", Proceedings of the ACM SIGMOD International Conference on Management of Data, 27-29 June 2006, Chicago, IL, USA, 793-795.

[9] Elmasri, R. ve Navathe, S.B., (2010). Fundamentals of Database Systems, Sixth Edition, Addison-Wesley Publishing Company, Boston.

[10] Ramakrishnan, R. ve Gehrke, J., (2002). Database Management Systems, Third Edition, McGraw Hill, New York.

[11] Lightstone, S., Teorey, T. ve Nadeau, T., (2007). Physical Database Design: The Database Professionals's Guide to Exploiting Indexes, Views, Storage, and More, Fourth Edition, Morgan Kaufmann Publishers, San Francisco.

[12] Lightstone, S., Teorey, T. ve Nadeau, T., (2006). Database Modeling & Design: Logical Design, Fourth Edition, Morgan Kaufmann Publishers, San Francisco.

[13] Sciore, E., (2009). Database Design and Implementation, First Edition, John Wiley High Education, Hoboken.

[14] Molina, H.G., Ullman, J.D. ve Widom, J., (2008). Database Systems The Complete Book, Second Edition, Pearson Prentice Hall, Upper Saddle River.

[15] Ramakrishnan, R. ve Gehrke, J., (2000). Database Management Systems, Second Edition, McGraw Hill, New York.

[16] Yanatma, M.A. ve Kalay, M.U., (2018). "Extension for Workload Performance Measurement in an Educational Database System", Third International Conference on Trends in Computing and Information Technology, 26-27 April 2018, İstanbul, 12-21.

[17] Kalay, M.U., (2013), Eğitimsel Veri Tabanı: VT Gerçeklenmesi-Ders Notları #3, http://www.yildiz.edu.tr/~ukalay/index_files/VTSG/vtsg_files/3_HafizaYonetimi.pdf, 8 Ocak 2019.

[18] Cormen, T.H., Leiserson, C.E., Rivest, R.L. ve Stein, C., (2009). Introduction to Algorithms, Third Edition, MIT Press, London.

[19] Kerr, W., (2008), Investigation into Knapsack, https://www.researchgate.net/publication/228425153_Investigation_into_Knapsack, 8 Ocak 2019.

[20] Shasha, D. ve Bonnet, P., (2003). Database Tuning: Principles, Experiments, and Troubleshootings Techniques, First Edition, Morgan Kaufmann Publishers, San Francisco.

[21] Tan, P., Steinbach, M. ve Kumar, V., (2006). Introduction to Data Mining, First Edition, Addison-Wesley Publishing Company, Boston.

[22] Two Crows Corporation, (1999). Introduction to Data Mining and Knowledge Discovery, Third Edition, Potomac.

[23] Han, J., Kamber, M. ve Pei, J., (2012). Data Mining Concept and Techniques, Third Edition, Morgan Kaufmann Publishers, Waltham.

[24] Zaki, M.J. ve Meira, W., (2014). Data Mining and Analysis: Fundamental Concepts and Algorithms, First Edition, Cambridge University Press, Cambridge.

[25] Grananaraj, T.N., Kumar, K.R. ve Monica, N., (2014). "Survey on Mining Clusters Using New K-mean Algorithm from Structured and Unstructured Data", International Journal of Advances in Computer Science and Technology, 3(2):60-65.

[26] Jain, A.K., (2010). “Data Clustering: 50 Years Beyond K-Means”, Pattern Recognition Letters, 31(8):651-666.

[27] Velmurugan, T., (2012). “Efficiency of K-means and K-medoids Algorithms for Clustering Arbitrary Data Points”, International Journal of Computer Applications in Technology, 3(5):1758-1764.

[28] Sciore, E., (2007). “SimpleDB: A Simple Java-Based Multiuser System for Teaching Database Internals”, Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, 7-11 March 2007, Covington, 561-565.

EK-A

İŞ YÜKÜ

Performans testlerinde kullanılan iş yükünü oluşturan sorgular ve bu sorguların frekansları Çizelge A.1'deki gibidir.

Çizelge A. 1 Performans testlerinde kullanılan iş yükü

Sorgu Numarası	Frekans	Sorgu
1	7	select sid from student where sname between 'l' and 'p' and gradyear=1988
2	6	select count(sid), majorid from student where gradyear=1981 group by majorid
3	2	select count(sid), gradyear from student where majorid=48 group by gradyear
4	1	select dname from dept where did=46
5	1	select did from dept where dname='fo'
6	1	select title, description from course where deptid=7
7	3	select max(cid), title from course where cid between 88 and 188 group by title
8	4	select cid, credit from course where title between 'q' and 's'
9	9	select cid, title from course where credit=9

Çizelge A. 1 Performans Testlerinde kullanılan iş yükü (devamı)

Sorgu Numarası	Frekans	Sorgu
10	3	select syllabus from section where sectid=3822
11	3	select courseid, syllabus from section where classid='1107'
12	5	select sectid from section where profid=1903 and yearoffered=2011
13	4	select sectid from section where courseid=2093 and yearoffered=1989
14	4	select studentid from enroll where eid=756375
15	4	select studentid from enroll where eid between 251565 and 252565
16	4	select eid from enroll where studentid=27237 and sectionid=15681
17	1	select eid, studentid, sectionid from enroll where gradeid between 14 and 16
18	4	select count(sectionid), gradeid from enroll where sectionid=16946 group by gradeid
19	4	select pname, rank from professor where pid=5158
20	6	select pid, pname from professor where rank='sjfxtrphk'
21	5	select pid, rank from professor where pname='qdt'
22	3	select buildname from classroom where clid=2401
23	5	select clid from classroom where buildname='nu'
24	3	select gid from grade lettergrade between 'x' and 'z'
25	1	select lettergrade from grade where gid=2
26	6	select sid, dname from student, dept where majorid=did and sname between 'u' and 'y'

Çizelge A. 1 Performas testlerinde kullanılan iş yükü (devamı)

Sorgu Numarası	Frekans	Sorgu
27	1	select sid from student, dept where majorid=did and dname='krsgjk'
28	6	select count(cid) from dept, course where did=deptid and title between 'i' and 'k'
29	6	select dname, title from dept, course where did=deptid and credit=8
30	3	select title, yearoffered from course, section where cid=courseid and profid=4384
31	3	select title, sectid, yearoffered from course, section where cid=courseid and yearoffered between 1986 and 1990
32	6	select count(sectid), pname from section, professor where profid=pid and rank between 'r' and 't' group by pname
33	4	select sectid, pname from section, professor where profid=pid and yearoffered=1986 and rank='sjfxtrphk'
34	2	select clid, buildname from section, classroom where classid=clid and sectid=14820
35	6	select sectid, classid from section, classroom where classid=clid and buildname='nc'
36	3	select studentid, gradeid from section, enroll where sectid=sectionid and eid between 83642 and 84642
37	3	select studentid, sectid, gradeid from section, enroll where sectid=sectionid and eid=379031
38	7	select studentid, sectionid from enroll, grade where gradeid=gid and lettergrade between 'x' and 'z'
39	5	select eid from student, enroll where sid=studentid and sname between 'a' and 'e' and majorid=0
40	5	select sid from student, enroll where sid=studentid and gradyear=2008 and eid between 873154 and 874154

Çizelge A. 1 Performas testlerinde kullanılan iş yükü (devamı)

Sorgu Numarası	Frekans	Sorgu
41	5	select title from student, dept, course where majorid=did and did=deptid and sid=25262 and credit=2
42	4	select yearoffered from dept, course, section where did=deptid and cid=courseid and title='wzyl'
43	4	select title, pname from course, section, professor where cid=courseid and profid=pid and courseid=3029 and rank='xc'
44	4	select clid, buildname from course, section, classroom where cid=courseid and classid=clid and title='ohrtn' and yearoffered=1980
45	4	select courseid, sectid, yearoffered from section, professor, classroom where classid=clid and profid=pid and buildname='ou'
46	3	select studentid, sectid from course, section, enroll where cid=courseid and sectid=sectionid and sectionid=6708 and title='tfzfpzvkk'
47	5	select sectid, studentid from enroll, section, professor where sectionid=sectid and profid=pid and sectionid=10927 and pname='sdx'
48	5	select sectid, clid from enroll, section, classroom where sectionid=sectid and clid=classid and buildname='qrdfsranv'
49	3	select sname from student, enroll, section where sid=studentid and sectid=sectionid and sectionid=11948 and majorid=17
50	4	select sname from student, enroll, grade where sid=studentid and gradeid=gid and studentid=5508 and gradyear=2008

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mehmet Akif YANATMA

Doğum Tarihi ve Yeri : 01.02.1988 / Fethiye - Muğla

Yabancı Dili : İngilizce

E-posta : akifyanatma@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Müh.	İstanbul Teknik Üniversitesi	2013
Lise	Fen Bilimleri	Burdur Fen Lisesi	2005

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2019-halen	ITG (Information Technology Group)	Yazılım Mühendisi
2013-2013	Ziraat Teknoloji	Yazılım Uzman Yardımcısı
2012-2013	Baykar Makina	Stajyer

YAYINLARI

Bildiri

1. Yanatma, M.A. ve Kalay, M.U., (2018). “Extension for Workload Performance Measurement in an Educational Database System”, Third International Conference on Trends in Computing and Information Technology, 26-27 April 2018, İstanbul, 12-21.