XML VERİTABANI İÇİN TAVLAMA BENZETİMİ VE GENETİK ALGORİTMA TABANLI SORGULAMA

Yaşar GÖZÜDELİ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

OCAK 2007
ANKARA

Yaşar GÖZÜDELİ tarafından hazırlanan XML VERİTABANI İÇİN TAVLAMA BENZETİMİ VE GENETİK ALGORİTMA TABANLI SORGULAMA adlı bu tezin yüksek lisans tezi olarak uygun olduğunu onaylarım.

Doç. Dr. M.Ali AKCAYOL
Tez Yöneticisi

Bu çalışma, jürimiz tarafından oy birliği ile Bilgisayar Mühendisliği Anabilim Dalında yüksek lisans tezi olarak kabul edilmiştir.

Başkan : Prof. Dr. Sezai DİNÇER

Üye : Prof. Dr. Hadi GÖKÇEN

Üye : Doç. Dr. M.Ali AKCAYOL

Tarih : 25.01.2007

Bu tez, Gazi Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygundur

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Yaşar GÖZÜDELİ

XML VERİTABANI İÇİN TAVLAMA BENZETİMİ VE GENETİK ALGORİTMA TABANLI SORGULAMA
(Yüksek Lisans Tezi)

Yaşar GÖZÜDELİ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
Ocak 2007

ÖZET

Bu çalışmada tavlama benzetimi ve genetik algoritma tabanlı XML sorgu iyileştirmesi gerçekleştirilmiştir. XML sorgulamada, sorgu ağacında yer alan düğümlerin birleştirilmesi sıklıkla yapılmaktadır. Sorgu çalıştırma planı oluşturulurken, düğümlerin birleştirilme sıralaması, sorgu maliyetini belirleyen en önemli etkendir. Bu çalışmada, düğümlerin sıralaması genetik algoritma ve tavlama benzetimi algoritmaları ile yapılarak iki algoritmanın sonuçları karşılaştırılmıştır. Uygulama C# 2.0 dili ile hazırlanmış ve açık kaynak Timber XML Veritabanı Yönetim Sistemi kullanılarak gerçekleştirilmiştir. Yapılan deneysel çalışmaların sonucunda, klasik yöntemlere göre daha basit şekilde uygulanabilen tavlama benzetimi ve genetik algoritma yöntemlerinin XQuery iyileştirmesinde başarılı oldukları görülmüştür.

Bilim Kodu : **902.1.011**
Anahtar Kelimeler : **XML, XQuery, Sorgu İyileştirme, Birleştirme Sıralaması, Genetik Algoritma, Tavlama Benzetimi, en uygun şekle sokma, Sistem-R, Çalı Ağacı, SQL, Veri Sorgulama, Timber**
Sayfa Adedi : **80**
Tez Yöneticisi : **Doç. Dr. M.Ali AKCAYOL**

SIMULATED ANNEALING AND GENETIC ALGORITHM BASED QUERYING FOR XML DATABASES

(M.Sc. Thesis)

Yaşar GÖZÜDELİ

GAZI UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
January 2007

ABSTRACT

In this study, simulated annealing and genetic algorithm based XML querying has been implemented. In the XML query, joining all nodes in the query tree have been done regularly. During constructing query execution plan, the join order of nodes is the most important factor to determine the cost of the query. In this study, the join order of the nodes have been done with genetic and simulated annealing algorithms, and the results of both algorithms have been compared. Application has been developed using C# 2.0 language and implemented using the open source Timber XML database management system. In the experimental results it has been showed that simulated annealing and genetic algorithms which are implemented more easily than classical methods have been successfully applied for optimization of XML query.

Science Code : 902.1.011
Key Words : XML, XQuery, Query Optimization, Join Order, Genetic Algorithm, Sımulated Annealıng, System-R, Bushy Tree, SQL, Querying Data, Timber
Page Number : 80
Adviser : Assoc.Prof. Dr. M.Ali AKCAYOL

TEŞEKKÜR

Tez çalışmasına desteğinden dolayı Gazi Üniversitesi'ne teşekkürlerimi sunarım. Çalışmalarım boyunca bilgi birikimiyle beni yönlendiren, araştırmalarımın en umutsuz zamanlarında, sıra dışı ikna gücü ve motivasyonuyla tezimi tamamlamamı sağlayan danışmanım Doç. Dr. M.Ali AKÇAYOL'a ve ilkokuldan üniversiteye eğitimimde emeği geçen bütün öğretmenlerime; her şartta bana destek olan eşim Lale GÖZÜDELİ'ye henüz tez yazmanın ne demek olduğunu bilmeseler de bazı günler beni görememe nedenlerini merak eden Kızım Neva GÖZÜDELİ ve oğlum Hakan GÖZÜDELİ'ye, ilk öğretmenlerim annem ve babama, sıkıştığım anlarda Timber XML Veritabanı Yönetim Sistemi ile programatik temas kurmamı sağlayan Bilgisayar Mühendisi Süleyman SALIN 'ÖSYM''a teşekkürlerimi sunarım.

İÇİNDEKİLER

Sayfa

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış bazı kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	Açıklama
XML	Extensible Markup Language
SGML	Standart Generalized Markup Language
XQuery	XML Query Language
SQL	Structred Query Language
VTYS	Veritabanı Yönetim Sistemleri
İVTYS	İlişkisel Veritabanı Yönetim Sistemleri
XVTYS	XML Veritabanı Yönetim Sistemleri
XPath	XML Path Language
W3C	World Wide Web Consortium
DTD	Document Type Defination
XSD	XML Schema Defination
XQueryX	XML Syntax for XQuery
FLWOR	For, Let, Where, Order, Return ifadeleri
CML	Chemical Markup Language
MML	Math Markup Language
WML	WAP Markup Language
XHTML	XML HTML
MiMI	Michigan Molecular Interaction Database

1. GİRİŞ

Extensible Markup Language(XML) her ne kadar bir veri aktarım standardı olarak tasarlanmış olsa da, aktarılan verinin kendi tanımını da içermesi nedeniyle bir veri saklama formatı olarak da kabul görmüş bir Standart Generalized Markup Language (SGML) alt dilidir [1, 2]. Özellikle yarı yapılanmış veri saklama standardı olarak, İlişkisel Veritabanı Yönetim Sistemleri (İVTYS) tarafından da kullanılmaya başlanmıştır [3, 4].

XML verinin sorgulanma ihtiyacı, XML'in veri aktarım formatı olarak kullanıldığı zamanlara dayanmaktadır. Bu çerçevede XML'in düğümler halinde sorgulanmasını sağlayan XPathstandart olarak World Wide Web Consortium(W3C) tarafından kabul edilmiştir [5]. Ancak XML, zaman içerisinde veri aktarım dışında, veri depolama standardı olarak da kabul görmüştür. Özellikle yarı-yapılanmış(semi-structured) verilerin depolanmasında XML'in bir veri saklama formatı olarak kabul görmesinden sonra ortaya birçok sorgulama şekli atılsa da bunlar arasında en çok kabul göreni, XPath'i de içine alan XML Query Language(XQuery) olmuştur [5-9]. XQuery henüz bir W3C standardı olma yolunda bir aşama olan, 'Candidate Recommendation' statüsünde bir standart taslağı olmasına rağmen, birçok akademik araştırmada ve önde gelen VTYS üreticileri tarafından tercih edilen bir XML sorgulama yöntemi olarak kabul görmüştür [3, 10-12].

Öte yandan sorgu sürelerinin kestirimi ve sorgu iyileştirmesi, ilişkisel veritabanı yönetim sistemleri fikrinin ortaya çıktığı 1970'li yıllardan günümüze, ticari ve akademik araştırmalar için ilgi odağı olagelmiştir [13]. Bu konudaki tekniklerin gelişim süreci ve temel dayanakları Chaudhuri tarafından detaylı olarak incelenmiştir [14]. XQuery temelli olarak geliştirilen XVTYS'ler, hiç şüphesiz İVTYS'ler çerçevesindeki bu yaklaşımlardan istifade etmiştir [15].

İVTYS ortamındaki birçok akademik çalışma, XML ortamına da başarıyla aktarılmış ama XML ortamı yönlü grafların özel bir hali olması nedeniyle ilişkisel ortamdaki bazı yöntemler XML sorgulama iyileştirilmesine başarı ile uygulanamamıştır.

Bundan dolayı birçok yeni sorgu iyileştirme yöntemlerinin geliştirilmesi de gerekmiştir [10, 11, 15, 16]. İlişkisel ortamdaki adaptasyonlara örnek bir çalışma olarak Bennett ve arkadaşları tarafından yapılan çalışmada ilişkisel VTYS için iyi bir iyileştirme tekniği olarak sunulan, en-soldan birleştirme(Left-Deep join veya Sistem-R) yaklaşımı, XQuery iyileştirmesinde başarılı bulunmamıştır [17]. Ancak XQuery optimizasyonu hakkında yapılan literatür taramasında meta sezgisel yaklaşımların kullanıldığı çalışmalara rastlanmamıştır. İVTYS sorgu iyileştirmesi ile ilgili yapılan bazı çalışmalar, meta sezgisel yaklaşım ile ilişkisel sorgu optimizasyonu yapılabileceğine dair iyi örnekler olarak ele alınabilir [15, 18]. M.Steinbrunn ve arkadaşları tarafından yapılan çalışmada rastsal yöntemler ile birlikte tavlama benzetimi ve genetik algoritma kullanarak birleştirme sıralaması yapılmış ve özellikle artan sayıda tablolar için, deterministik yöntemlere göre daha başarılı sorgulama sonuçları elde edilmiştir [18].

Bu tez çalışmasında, İVTYS için başarıyla uygulanmış olan Genetik Algoritma ile Birleşim Sıralaması (Join Order) sorununun XML temelli ortamda da çözümü gerçekleştirilmiştir. Birleşim Sıralaması, XML ortamın düğümlerden oluşan yapısının depolanıp veri alınması sırasında tekrar bir araya getirilme zorunluluğu nedeniyle, İVTYS'ye nazaran çok daha fazla öneme sahiptir. Çünkü artan sayıda birleşim gereksinimi ile birlikte deterministik yaklaşımlar yeterli performansı sağlayamamaktadır [17].

Bu çalışmada, XQuery sorgu iyileştirmesinde, ilk kez Genetik Algoritma kullanılmış ve bu kullanılabilirliğini ortaya çıkarmak ve bu sonuçları Tavlama Benzetimi Algoritması ile kıyaslamaktır. Yapılan çalışmada, sorguda kullanılacak düğümlerin öncelik sıralamaları ile ilgili iyileştirmeler için mantıksal sorgu planı oluşturulması aşamasında genetik algoritma kullanılmıştır.

Bu tezin ikinci bölümünde XML ve XQuery anlatılmış ve üçüncü bölümünde yapılan çalışmanın dayanağını oluşturan veri sorgulama yöntemleri ele alınmıştır. Tezin dördüncü bölümünde Genetik Algoritma ve beşinci bölümünde Tavlama Benzetimi, yapılan çalışma sonucu elde edilen deneysel sonuçları içermektedir. EK-1'de

projenin üstüne inşa edildiği Timber DB açık kaynak XML Veritabanı Yönetim Sistemi hakkında, yapılan çalışmanın anlamlandırılması için gerekli temel teknik ayrıntılara yer verilmiştir.

2. XML VE XQUERY

Bu bölümde XML'in yapısı, bir veri saklama yöntemi olarak kullanımı ve sorgulanması ele alınmaktadır.

2.1.XML

Kendi tanımını içeren veri işaretleme dili olarak SGML'den türetilen XMLstandart olarak yayınlanmasından bu yana veri aktarım standardı olarak kabul görmüştür [1]. XML' de önceden belirlenmiş sayıda etiket yer almaz. Sadece etiketlerin ve etiketlerle tanımı elemanların belli kuralları sağlaması istenir. Bu nedenle XML esnek bir veri işaretleme dilidir. Her türlü veri aktarım gereksinimine göre veri aktaran taraflar arasında ortak olmak kaydı ile farklı etiketler ve elemanlar kullanmak mümkündür.

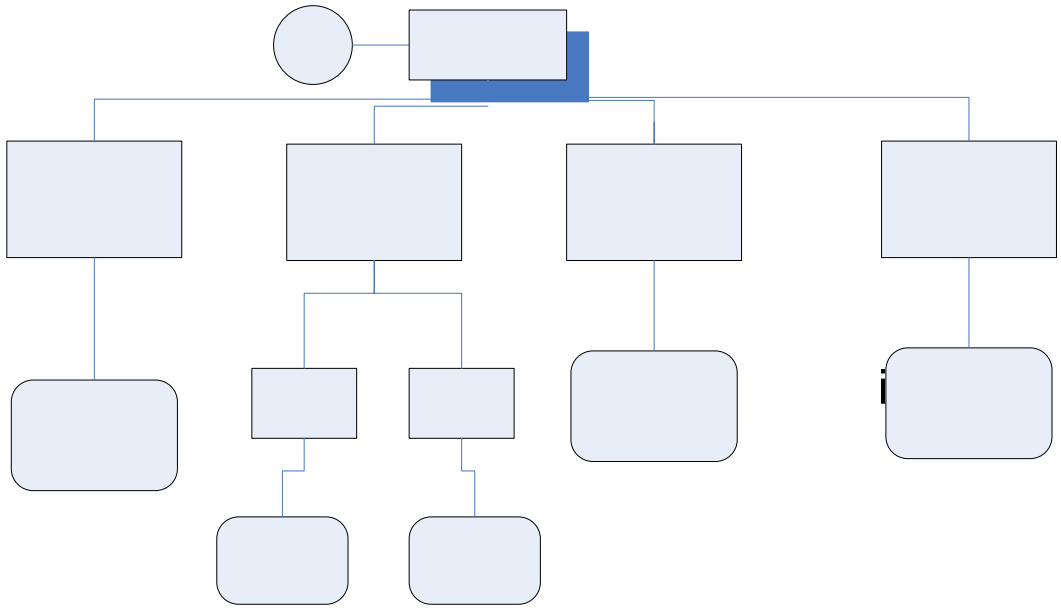
2.1.1.XML'in genel tanımı

XML, veri işaretleme dilleri için genel bir platform sağlamaktadır. Bu kapsamda kullanılmakta olan birçok işaretleme dili mevcuttur. XML HTML(XHTML), Chemical Markup Language (CML), WAP Markup Language (WML) ve Math Markup Language (MML) gibi standartlar bu platformda geliştirilmiş işaretleme dillerine örnek olarak verilebilir.

İçerisinde XML verisi içeren dokümana XML Dokuman denir. Bir XML doküman şu özellikleri sağlar:

- Bir XML Doküman, sıralı ve etiketli ağaç olmalıdır.
- Ağacın her bir düğümü veri veya başka bir düğüm içerebilir
- Eleman içeren düğümlerde her bir eleman, elemanın adını içeren bir etiket ve gerekli ise nitelikler yardımı ile etiketlendirilmiş olmalıdır.

Bu bilgiler ışığında örnek bir XML dokümanı Şekil 2.1'de verilmiştir.



Şekil 2.1.Bir XML dokümanının yapısı [19]

Bütün bunların yanı sıra aşağıdaki kuralları da sağlayan bir dokümana iyi yapılanmış XML denir:

- Her açılan etiket kapanmalıdır
- Elemanlar doğru sırada açılıp kapanmış olmalıdır.
- Her doküman tek bir tane kök eleman içerebilir. Diğer elemanlar (varsa) bu eleman içerisinde yer almalıdır.

2.1.2.XML'in veri işaretlemede kullanımı

XML Veri işaretlemede iki farklı şekilde kullanılabilir: tip tanımlı ve tip tanımsız XML. Tip tanımsız XML, iyi yapılanmış olmalıdır. XML tanımını içermek kaydı ile her türden veri içerebilir. Ancak tip tanımlı olarak tayin edilmiş bir XML'in içerebileceği elemanlar ve elemanların alabileceği değerler önceden bir XML Schema Defination(XSD) veya Document Type Defination(DTD) yardımı ile tanımlanır. Böylece, özellikle veri aktarımında farklı taraflar arasında bir veri formatı uyumu sağlanmış olur.

2.1.3.XML'in veri saklamada kullanımı

XML'in geniş bir çevrede veri işaretleme dili olarak kabul görmesi ve internetin de etkisi ile veri aktarım ve paylaşım işlemlerinin yaygınlaşması sonucunda XML'i veri gösteriminin dışında veri saklama formatı olarak ele alınmaya başlanmıştır.

İlişkisel veritabanı yönetim sistemlerinin performansı ve hızlı işlem yapabilme desteği nedeniyle başlarda birçok yaklaşımda XML verinin ilişkisel veri formatına dönüştürülerek, satır ve sütunlardan oluşan tablolarda saklama yönteminin tercih edildiği görülür. Florescu ve Kossman bu konuda, yetenek olarak gelişmiş bir yöntem sunarlar [20]. Önce XML dokümanını bir sıralı, yönlü graf olarak ele alırlar ve daha sonra bu yönlü grafın her bir kenarı ile etiket değerini binary veya kenar tablosu olarak adlandırılan tablolarda saklarlar. Bu türden bir yaklaşımın, avantajı tip tanımlı veya tip tanımsız bütün dokümanların kaydedilebilir olması iken, dezavantajı ise sağladığı performansın veri karmaşıklaştıkça düşmesidir.

XML veriyi ilişkisel ortamda saklama yaklaşımlarından birisi de ilişkisel veritabanı tabloları üstünde XML view'ler tanımlamak suretiyle tabloları birer sanal XML saklama ortamı haline dönüştürmektir. M.Fernandez ve arkadaşlarının önerdiği SilkRoute çalışması bu türden bir yaklaşıma örnek olarak ele alınabilir [21]. Bu araştırmada RXL adında bir ara dil önerilmiştir. Bu dil sayesinde ilişkisel verilerin XML olarak view haline getirilebilmesi mümkündür. Bu dil retrieval ve construction olmak üzere iki kısımdan oluşur. Birinci kısım ilişkisel ortamdan verilerin çalışması ile ilgilidir. İkinci kısım elde edilen verilerin XML olarak sunulmasını sağlamaktadır. Bu ifadeler daha sonra bir sorgu yorumlayıcısı tarafından iki parçaya ayrılarak retrieval kısmı bir SQL ifadesine dönüştürülür ve ilişkisel ortamda çalıştırılır. Construction kısmı ise XML üretimi yapılacak bir şablon tayin edilmesinde kullanılır.

Bu tür yaklaşımlar çerçevesinde birçok çalışma yapılmıştır. Bir örnek olarak Shanmugasundaram ve arkadaşları bir çalışma yapmıştır [22]. Bu türden yaklaşımların çeşitli zayıf yönleri mevcuttur. Bunlar şu şekilde sıralanabilir:

- XML dokümanlarını ilişkisel veriye parçalama ve tekrar XML'e çevirme maliyeti
- Yarı yapılanmış(düzenli şeması olmayan) veriyi saklamak için tabloları kullanmanın normalizasyon kuralları nedeniyle talebi karşılamada etkin bir çözüm olmakdan uzak kalması. Ya performanstan veya veri ormalizasyonundan ödün verme zorunluluğu vardır.
- Dokümanın gerçek manada kaydedildiği şekli ile oluşturulma zorluğu. Elemanların sıralarının kaybedilmesi veya kaybetmemeye dönük ek sıralama maliyetleri.(XML Sıra duyarlı bir ortam iken tablolarda kayıtların fiziksel sırası önemli değildir)
- Kendine atıfta bulunan şemaların işlenmesinde ortaya çıkan performans sorunlarına yeterince etkin bir çözüm bulunamaması.
- XML sorgulamaya tam destek verilememesi. (XPath ile sorgulamaya kısıtlı bir destek verebilmektedir.)

Bütün bu zayıf yönleri gidermek için veritabanı yönetim sistemleri sektörü, XML verinin bir skalar tip gibi ilişkisel ortamın içine entegre edilmesinin en iyi seçenek olduğuna karar vermiştir. Bu gelişmelere paralel olarak, sektörde söz sahibi VTYS üreticileri (SQL Server 2005, Oracle 10g…) hibrit veri gereksinimine cevap olarak XML veri tipi geliştirmişlerdir. Böylece tabloların bir sütununda da XML veri saklanabilir hale getirilmiştir.

Bu şekilde, ilişkisel veritabanı sistemi içerisinde XML verinin doğal hali ile saklanabilmesiyle elde edilen yeni ilişkisel sisteme hibrit ilişkisel veritabanı yönetim sistemi denmektedir. Modern İVTYS'lerin hepsi ANSI standartları çerçevesinde hibrit VTYS'ler olarak değerlendirilebilirler.

Bu sistemin de çeşitli avantajları vardır. Araştırmacılar tarafından bu türden bir sistemin avantajları şu şekilde sıralanmıştır [23]:

- XML-İlişkisel veri dönüşüm maliyeti azaltılmış olur. Doğrudan XML olarak üretilip kaydedilebilen veriler varsa, ilişkisele çevrilip saklanması ve geri dönüştürülüp istemciye sunulması gibi aşamalar bertaraf edilebilir hale getirilmiştir.
- Bazı durumlarda karmaşık verileri normalize etmek çok fazla tablo gereksinimi doğurabilir. Bu tür durumlarda hibrit bir veritabanı yönetim sistemi doğrudan XML kaydedilmesine olanak sağladığı için avantaj sağlayacaktır.
- Bazı XML veriler için hiç şema tanımı yapılamıyor olabilir. Bu türden verileri de hibrit ortamda kaydetmek mümkündür.
- XML verinin doğrudan bir XSD veya DTD ile denetlenebilmesi mümkün hale getirilmiştir.

2.2.Doğal XML Veritabanı Yönetim Sistemleri

Hibrit Veritabanı Yönetim sistemlerinin dışında Doğal Veritabanı Yönetim Sistemleri de doğrudan XML kaydeden uygulamalar olarak sayılabilir. Bu türden veritabanı yönetim sistemler her geçen gün artmaktadır [10, 11, 24, 25]. Bunlardan belli başlıları Tamino, Galax ve Timber olarak sıralanabilir.

2.2.1.Tamino

Software AG tarafından gerçekleştirilen bu doğal XML veritabanı yönetim sistemi temelde Adabas'a dayalı bir doğal XML veri depolama yöntemi kullanmaktadır. Ticari olarak piyasada bulunan nadir bir doğal XVTYS olarak kabul görmüştür [25]. XPath benzeri bir veri sorgulama diline olanak sağlar. Ürün geliştirme tarihi XQuery standartlaşmasından daha öncedir. Ancak buna rağmen takip eden sürümleri XQuery desteği ile sunulmaktadır.

2.2.2.Galax

AT&T, Bell, IBM ve çeşitli üniversite mensupları tarafından geliştirilen bir açık kaynak XML Veritabanı Yönetim Sistemi olan Galax, yine proje takımı tarafından geliştirilen Jungle adında bir XML temelli depolama motoru üstüne kuruludur. Jungle da açık kaynak veritabanı depolama motoru olarak BerkeleyDB'yi kullanmaktadır. Caml dilinde geliştirilen bu XVTYS, XQuery temelli veri sorgulamaya olanak sağlayan ilk ürünlerden biri olarak kabul görmüştür [11].

2.2.3.Timber

University of Michigan bünyesinde geliştirilen Timber açık kaynak XML veritabanı yönetim sistemidir [10]. Yapılan tez çalışmasındaki yöntemler, Timber XVTYS kullanılarak çalıştırılmıştır. Özellikle 100MB civarına kadar XML dokümanlarını başarı ile yükleyip sorgulayabilen Timber, bu yönü ile birçok küçük seviye XVTYS'den ayrılmaktadır.

Sonuç olarak, doğal XML veya hibrit ortamlarda XML olarak kaydedilen verilerin belli kısımlarına erişilebilme gereksinimi, XQuery adı ile anılan XML Sorgulama dilinin geliştirilmesine yol açmıştır.

2.3.XQuery

XQuery, XML veriyi sorgulamak için kullanılan bir sorgu dilidir [9]. Henüz standart olarak kabul edilmemiş olsa da W3C'nin çalışmaları devam etmektedir. Tavsiye Taslağı[1] aşamasındaki bu dil hâlihazırda, MS SQL Server 2005, Oracle 10g ve DB2 gibi sektörde kabul görmüş birçok İlişkisel Temelli Hibrit Veritabanı Yönetim Sistemleri ve Galax, Timber gibi doğal XML Veritabanı Yönetim Sistemleri tarafından kullanılmaktadır.

[1] " Proposed Recommendation" kavramı W3C için bir standart'ın geliştirmesindeki son ön aşamadaki adıdır. Standart yolunda konunun bir olgunluğa ulaştığını ve ticari uygulamalardaki sonuçlarının değerlendirilmesini kapsayan süreçtir. XQuery 2006 Aralık sonuna kadar bu seviyede bekletilmektedir.

XQuery, bir XML Sorgulama dilidir ve XML temelli dil olarak kabul edilmez. XML uyumlu XQuery yazım tanımı XQueryX olarak adlandırılmaktadır ve XQuery ile aynı olgunluk düzeyinde bulunmaktadır [26].

W3C tarafından sunulan taslağa göre, XQuery'nin gerçeklenebilmesi için birçok temel gereksinim yer almaktadır [27]. Bunlar şu şekilde sıralanmaktadır:

- İnsanlar tarafından da anlamlandırılabilen bir yazım şekli.
- Tanımlanabilirlik.
- Protokol bağımsızlık.
- XML Veri Modeli ile tutarlılık.
- XML isim uzayları ile uyumluluk.
- Basit ve karışık veri tiplerini destekleme.
- Hiyerarşiyi ve doküman sırası gibi XML'e özgü işlemleri destekleme.
- Birden fazla dokümandan veri birleştirebilme.
- Gruplama yapabilme.
- XML dokümanlarından istenen formatta yeni XML dokümanları oluşturabilme.
- ID'lerle referans geçişlerine izin verme.

XQuery, SQL'e göre daha kısıtlı bir dil olsa da, SQL'de yer alan Data Manumlation Language(DML) ifadelerinden SELECT'e eşdeğer bir dil olduğu söylenebilir. Tek farkları, SELECT ifadesi ile tablo ve sütunlarda saklanan ilişkisel veri sorgulanırken, XQuery ifadeleri ile de XML formatında saklanmış veriler sorgulanabilmektedir.

XQuery, şu ön aşama kullanımların ve standartların devamı niteliğindedir:

- XML-QL
- YATL
- Lorel
- Quit
- XPath

- XSD

XQuery, XML gibi büyük-küçük harf duyarlı bir dildir ve ayrılmış kelimelerinin tamamı küçük harflidir. XQuery ifadeleri genel olarak bir takdim(prolog) ile başlarlar.

Açıklamalar aşağıda gösterildiği gibi (: ve :) işaretleri arasına yazılarak belirlenir.

```
(: XQuery içinde bazı açıklamalar
çok  satıra yayılmış  olabilir :)
```

Bir XQuery ifadesi, bir XML dokümanından XML düğümü veya atomik bir değer okur ve bir düğüm veya atomik değeri sorgu sonucu olarak döndürür.

Bir XQuery ifadesi temelde şu tür içeriklerden oluşabilir:

- XPath 2.0 benzeri yol ifadeleri
- Şayet seçtiği elemanı farklı bir eleman olarak döndürüyorsa eleman tanımlayıcısı
- FLWOR ifadeleri
- Listeleme ifadeleri
- Şart ifadeleri
- Tanımlı ifadeler
- Veri Tipi İfadeleri
- Değişkenler

2.2.1.Yol ifadeleri

XQuery, yol ifadeleri tanımlarını XPath2.0 standardından miras almıştır. XPath2.0 ile kullanılabilen bütün yol tanımları XQuery'de de kullanılabilir. Bir XQuery'nin açılış ifadesine prolog denir. Bu ifade bir doküman adı ve bu doküman üstünde tanımlı bir yol ifadesinden ibarettir.
Yol ifadesi de içeren bir örnek XQuery prolog ifadesi aşağıda gösterilmiştir.

```
doc("kitaplar.xml")/kitaplar//kitap/[isim=”XQuery”]
```

Yukarıda verilen ifade, kitaplar düğümü altındaki herhangi kitap düğümü altında, adında “XQuery” geçen kitapların[2] bir listesini döndürecek bir referans tanımlamaktadır.

2.2.2.Değişkenler

XQuery ile değişken tanımlanabilir ve SQL'deki atama işlemine eşdeğer olarak, değişkenlere değer atanabilir.

Bir değişkene atomik değerler atanabileceği gibi bir düğüm de atanabilir.

Aşağıda bir değişkene atomik değer atanması gösterilmektedir. := işareti ile şayet bir düğüm değişkene aktarılırsa, bu durumda düğümler tekil olarak ilgili değişkene bağlanır.

```
let  $msg := 'değişken değeri'
```

2.2.3.Eleman tanımlayıcıları

Bir XQuery sorgusu yeni bir XML dokümanı veya elemanı oluşturmak için kullanılabilir. Bu durumda, taslak tanımında yer alan yapıcı(construct) kullanılabilir ve XQuery ifadeleri { ve } işaretleri ile sonlandırılıp yeniden başlatılabilir [11].

2.2.4.FLWOR ifadeleri

FLWOR kısaltması XQuery'nin temelini oluşturan 5 harflik kısaltma olup sırayla F:Döngü (For), L:Atama(Let), W:Kriter(WHERE), O:Sıralama(ORDER BY) ve R:Döndürme(RETURN) ifadelerine karşılık gelir. SQL dili ile XQuery FLWOR ifadelerinin işlevsel karşılaştırmasına Çizelge 2.1’de yer verilmiştir.

[2] XML örneklerinde geçen “kitaplar.xml” dosyasının bir dökümüne tezin ek-b kısmında yer verilmiştir.

Çizelge 2.1.SQL ve XQuery deyimlerinin işlevsel karşılaştırılması [23]

Xpath FLWOR	SQL'deki SELECT Sorgu ifadesi Karşılığı
RETURN	SELECT
FOR	FROM
LET [3]	SET
WHERE	WHERE
ORDER BY	ORDER BY
doc("....")....**[**...**]**	WHERE

Bu işlemlerin yapılabildiği yere de XQuery Deyim Gövdesi (Expression Body) adı verilmektedir. Bir XQuery ifadesine öncelikle bir girdi olması gerekir. Bunun için fn:doc() fonksiyonu kullanılır. doc() fonksiyonu ile bir XML veri kaynağına erişim sağlanır ve basitçe bir XML veri dosyasına erişim için aşağıda olduğu gibi kullanılır.

```
doc("xmldosya_ismi.xml")
```

Ardından açılan doküman üstündeki düğümlere XPath'deki kullanımına eşdeğer olarak, listelenecek düğümlerin yol tanımı verilebilir.

Aşağıda EK-2'de yer alan kitaplar.xml dokümanındaki kitapların ismini seçecek bir XQuery ifadesi ve sonucu gösterilmektedir.

```
for $kitaplar=doc("kitaplar.xml")/kitaplar/kitap/isim
return kitaplar
```

sonuç:

```
<isim>Visual Basic.NET</isim>
```

[3] XML doküman düğümlerine erişimi tayin eden for() fonksiyonundan sonra verilen şartlarla(XPath) da neticede yer alacak düğümleri filtrelemek mümkündür. Bu nedenle, SQL'deki WHERE ifadesine bu işlem de karşılık olarak gösterilebilir.

```
<isim>Telkin ve Hipnumaraz ile ögrenme Teknikleri</isim>
<isim>Yatirim Plani Yapma</isim>
<isim>Is Basinda Duygusal Zeka</isim>
<isim>Is Hayatinda Motivasyon</isim>
<isim>Hayat Yolunda zorluklarla Mücadele</isim>
```

Birden fazla XML dokümanı bir biri ile birleştirilerek sorgulanabilir. Bu türden bir örnek XQuery ifadesi aşağıda gösterilmektedir.

```
for $y in document("yazarlar.xml")/yazar
let $k := document("kitaplar.xml")//yazarNo = $y/yazarNo
where count($k) >= 2
order by count($k) descending
return
  <fazlaKitapYazanlar>
    {
     $y
    }
    </fazlaKitapYazanlar>
```

2.2.5.Listeleme ifadeleri

SQL'de SELECT ifadesi ile birlikte çeşitli fonksiyonlar kullanılabildiği gibi, XQuery ifadeleri ile birlikte de hazır fonksiyonlar kullanılabilir. XQuery içerisinde hali hazırda bulunan fonksiyonların genel bir listesi ilgili W3C dükumanında yer almaktadır [28]. İhtiyaç halinde, yeni fonksiyonların XQuery çerçevesinde kullanıcı tarafından oluşturulması mümkündür.

2.2.6.Şart ifadeleri

XQuery, if-then-else genel şart ifadelerini destekler. Bu türden bir kullanım aşağıda gösterilmiştir.

```
for $k in document("kitaplar.xml")//kitap
return
```

```
  <kitapKalinlik>
    { $k/isim,
      if ($k/sayfaSayisi/text() >100)
     then
   }kalın{
      else
   }
ince
  </kitapKalinlik>
```

2.2.7.Veri tipi ifadeleri

XQuery XML Schema öneri tanımında yer alan basit ve karmaşık veri tiplerinin tamamını destekler [29]. Sabit değerlerin literal olarak verilmesine olanak tanır. Başlatıcı fonksiyonlarla, sabitlerin veri tiplerine doğrudan dönüşümünü destekler. Bu türden bir örnek aşağıda verilmiştir.

```
true(1)
date("2006-12-22")
```

Benzer şekilde, bilinçli tür dönüşümlerini de destekler. Bu türden bir örnek aşağıda verilmiştir.

```
let $a= xsd:positiveInteger(47)
```

3. VERİ SORGULAMA YÖNTEMLERİ

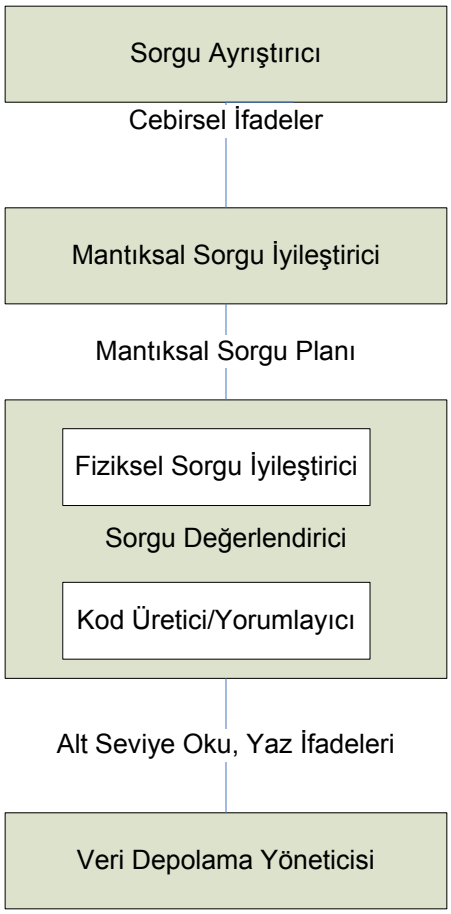
Bu bölümde veri sorgulama yöntemleri gelişim süreci içerisinde ele alınacaktır. Bu çerçevede ilişkisel bir VTYS'de veri sorgulamanın geçtiği evreler ve veri sorgulama iyileştirmesinin yapılması anlatılacaktır.

3.1.İlişkisel Veri Sorgulama Yöntemleri ve Gelişim Süreci

Büyük ölçekli verilerin yönetimi için İlişkisel Veritabanı yaklaşımının otaya atıldığı 1970'li yıllardan bu yana veritabanları verinin popülerliğine paralel gelişime maruz kalmışlardır [13]. Özellikle eş zamanlı kullanıcı erişim gereksinimiyle birlikte, sorguların daha hızlı yanıtlanması noktasında birçok çalışma yapılmıştır. Chaudhuri tarafından yapılan araştırma, bu alanda yapılan çalışmalar için genel bir özet niteliğindedir [30]. Endüstrinin önde gelen VTYS üreticilerinin de en çok insan gücü ayırdığı konulardan biri sorgu optimizasyonu araştırmaları olmuştur. Çünkü artan veri miktarlarıyla birlikte veriye hızlı erişim talebini karşılama gereksinimi ortaya çıkmaktadır.

3.1.1.Sorgulamanın aşamaları

Günümüzde birçok ilişkisel veritabanı yönetim sistemi veri sorgulamada benzer aşamalardan geçerek sorguları cevaplayabilmektedir. Bu sürecin nasıl olması gerektiği blok diyagramlarla Şekil 3.1'de gösterilmiştir.



Şekil 3.1. VTYS'de sorgunun cevaplanma süreci [30]

- *Sorgu Ayrıştırıcı*: Sorgu ayrıştırıcı, kullanıcıdan gelen sorgu dilini ayrıştırarak sentaks olarak geçerliliğini denetler. Şayet geçerli bir ifade ise cebirsel operatörlere veya eşdeğeri dâhili ifadelere dönüştürülür.

- *Mantıksal Sorgu İyileştirici*: Mantıksal sorgu iyileştirici, gerçek veri hakkında fikir sahibi değilken, eşdeğer ifadeleri sadeleştirerek daha kolay işletilebilir bir hale getirmeye çalışır. Buna ifade arıtımı da denir. Mantıksal Sorgu iyileştiriciden çıkan sonuç bir Mantıksal Sorgu Planıdır.

- *Fiziksel Sorgu İyileştirici*: Fiziksel sorgu iyileştirici, bir sorgu çalıştırma planı üreticisi ve bir de sorgu planı değerlendiricisinden oluşur. Üretici mümkün oldukça fazla plan üretmeye çalışırken, değerlendirici bu planlardan en düşük maliyetlisini

seçmeye çalışır. Bu aşamada, Veri hakkındaki veri(metadata) ve İndekslerden faydalanarak çok fazla çalıştırma planı arasından maliyeti düşük bir plan elde edilmeye çalışılır. Sorgu değerlendiricisi en iyi çalıştırama planı olarak seçtiği bir fiziksel planı Kod üreticiye çıktı olarak verir.

- *Kod üreticisi*: Sorgu işleyicisine ilgili fiziksel planın neticesinin döndürülebilmesi için gerekli okuma veya yazma işlemlerini anlamlandırabileceği alt seviye komutlar (write, retrieve gibi) olarak ifade eder ve sorgu işleyicisi bu işlemleri gerçekleştirerek sorguyu neticelendirebilir.

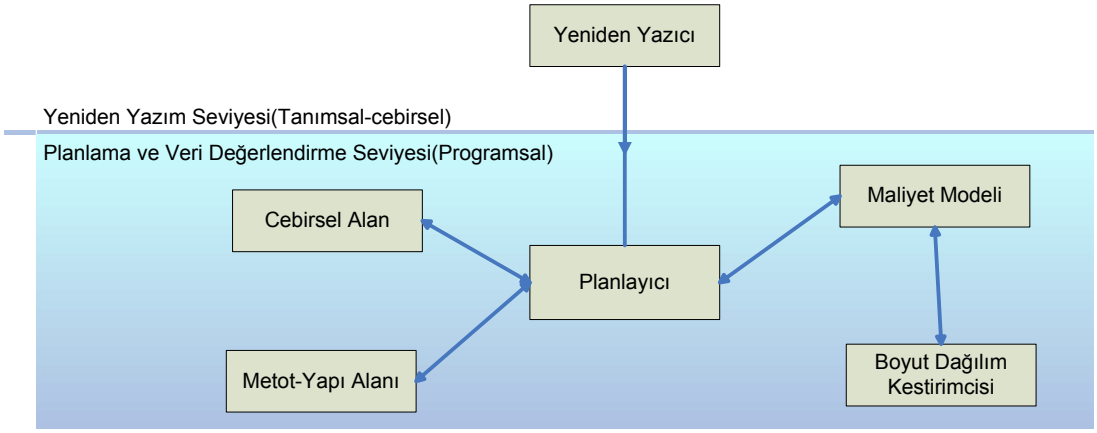
3.1.2.Sorgu iyileştirme yaklaşımları

Ele alınan sorgu için üretilen sorgu planı iyileştirmesi yapılabilmesi için temel gereksinimler şu şekilde verilmektedir:

- Ele alınan sorgu için alternatif sorgu planları türetebilen bir algoritma
- Kestirimsel olarak en ucuz bedelli sorgu planını arayacak bir algoritma
- Planın bedelini kestirecek bir yöntem

Bir sorgu planı oluşturulurken ideal durum, bütün sorgu planlarının oluşturulması ve bu planlar arasından en ucuz maliyetli olanının seçilmesi işlemidir. Ancak bunun bedeli çok fazla olduğundan bir sorgunun yanıtlanması sırasında pratikte uygulanamaz. Pratikteki sorgu iyileştirme yöntemlerinin çoğu sadece en kötü sonuçtan kaçınmayı ele alırlar.

Sorgu iyileştirici bazı VTYS'lerde izole bir program bloğu iken bazı programlarda diğer işlevlerle iç içe geçmiş olabilmektedir. Genel bir Sorgu İyileştirici modeli aşağıdaki şekilde verilmiştir:



Şekil 3.2.Genel bir sorgu iyileştirici modeli [30]

Araştırmada, bir sorgu iyileştirmesi iki sahada ele alınmakta. Sorgunun neticesini değiştirmeyecek şekilde ve çalıştırılmasını kolaylaştırmak üzere ara seviyelerde yeniden yazılması işlemi tanımsal yöntem olarak sunulurken, sorgunun fiziksel veri dağılımı ve meta verilerler desteği ile çeşitli kuramlar dâhilinde maliyet hesabı yapılarak iyileştirilmesini sağlamak da yordamsal kısım olarak tanımlanmakta.

3.1.3.İlişkisel cebir operatörleri

Sorgu dili, kullanıcıların veritabanından veri çekmelerini, veri değiştirip ekleme yapmalarını ve ortamı yönetebilmelerini sağlayabilen bir dildir. İlişkisel Yaklaşım, SQL adı verilen sorgu dilini destekler.

Sorgu dilleri, karışık hesaplamalar için değildir. Sadece veri ile ilgili işlemler için geliştirilmişlerdir. Bu nedenle yeni geliştirilen VTYS'ler hesaplama bazlı işlemler için ek programlama konseptleri sağlarlar (SQL Server 2005 .NET Framework, Oracle Java Framework ile birlikte piyasaya sunulmaktadır). Bunun yanında sorgu dillerini, çok büyük verilere erişebilirler. Veri erişiminin ifade edilmesinde ilişkisel cebirden faydalanılır. Sorgu çalıştırma planı gösterimi ve yeniden yazımla indirgenmesi gibi işlevleri sağlar. İlişkisel cebir'in ifade ettiği sorgular, ilişkisel nüshalara (tablo sonuçlarına) uygulanır ve elde edilen sonuç yine ilişkisel bir nüshadır.

İlişkisel cebir 6 temel operatör ile tanımlıdır. Bu operatörlerin genel bir tanımı ve Çizelge 3.1'de verilmiştir.

Çizelge 3.1.İlişkisel operatörler ve işlevleri [31]

Operatör	İşlev
SEÇME(σ)	Bir ilişkiden elemanların bir kısmını veya tamamını seçmede kullanılır.
İZDÜŞÜM(π)	İstenemeyen sütunların sonuçta çıkmasını önlemek için kullanılır.
KARTEZYEN ÇARPIM(×)	İki ilişkiyi birleştirmek için kullanılır.
FARK ALMA(-)	İki ilişkiden birincide bulunup ikincide bulunmayan elemanları bulmak için kullanılır.
BİLEŞİM(U)	Her iki ilişkinin kapsadığı elemanları bulmak için kullanılır.
İSİM DEĞİŞTİRME(ρ)	İki veya bir ilişkiyi parametre olarak alır ve dışarıya yeni bir isimle ilişki verir.

- *Seçme(σ)*: Gösterimi, σp(r) şeklindedir. Tanımlanması ise σP (r) := {t | t € r ve P(t)} şeklindedir. Burada r bir ilişki göstermektedir. P verilen şartlar dahilinde bir hesaplama belirtir. Şartlar şu şekilde verilir:

<nitelik> = <nitelik> veya <sabit literal>

Karşılaştırmada, = yerine diğer mantıksal karşılaştırma operatörleri de gelebilir. Bu operatörlerin bir listesi aşağıda yer almaktadır.

Çizelge 3.2. İlişkisel sorgularda kullanılabilecek cebirsel karşılaştırma işaretleri [23]

Sembol	Mantıksal Operatör
=	Eşillik
<	Küçüklük
>	Büyüklük
∩	Kesişim-Ve
U	Birleşim-Veya
!	Değil
<> veya !=	Eşit değil

- *İzdüşüm(π)*: Gösterimi, $\pi_{A1, A2, ..., Ak(r)}$ şeklindedir. Burada $A_1, \ldots, A_k$ nitelileri r ilişkisinden gelen niteliklerdir. İzdüşürme operatörünün neticesi k <= n olmak üzere k sütundan oluşan yeni bir ilişki tanımlar ve k < n için en az bir sütunun yeni ilişkide yer almadığını gösterir. Aynen tekrarlayan satırlar varsa küme olduğu için sadece bir defa gösterilirler.

- *Kartezyen çarpım(×)*: r ve s iki ayrı ilişki olmak üzere, r ile s'nin kartezyen çarpımı r × s şeklinde gösterilir ve şu şekilde tanımlıdır:

$r \times s := \{tq \mid t \in r \text{ ve } q \in s\}$ ile tanımlıdır.

Şayet r(R) ve s(S) ayrık nitelikler barındırıyorsa ya da örneğimiz için $R \cap S = \varnothing$ durumunda operatör için yeniden isimlendirme zorunluluk değildir. Ancak $R \cap S \varnothing$ şartı bozulduğu zaman(aynı ilişki kendisiyle kartezyen çarpılıyorsa) bu durumda ilişkilerden biri için isim değiştirme operatörü kullanmak zorunluluk halini alır.

- *Fark alma(-)*: r ve s birer ilişki olmak üzere r'nin s'den farklı olan satırlarını ifade eden fark operatörü şu şekilde gösterilir: r-s ve bu operatörün tanımı

$r - s := \{t \mid t \in r \text{ ve } t \notin s\}$

Ancak r - s operatörünün geçerli olabilmesi için r ve s'in niteliklerinin aynı tip sisteminden olması ve aynı sırada olması gerekir.

- *Birleşim(U)*: r ve s birer ilişki olmak üzere r ve s'in içerdiği satırları ifade eden BİLEŞİM operatörü r U s şeklinde gösterilir ve

r U s := {t | t ∈ r veya t ∈ s}

şeklinde tanımlıdır.

Ancak r U s in geçerli olabilmesi için şu kuralın sağlanması gerekir: r ve s'nin aynı sayıda ve aynı özellikte niteliklerden oluşmuş ilişkiler olması gerekir

- *İsim değiştirme(ρ)*: Bazı durumlarda bir ilişkisel cebirin ürettiği sonuca yeni bir isim vermek gerekebilir. Bu tür durumlarda isim değiştirme operatörü kullanılır. Bazen aynı ilişkiyi farklı iki girdiymiş gibi bir operatöre vermek gerekebilir. Bu durumda bu aynı sonuç farklı isimlendirmelerle kullanılır.

Bu temel ilişkisel cebir işlemlerinin yanı sıra, bu işlemler üstünden hesaplanabilmelerine rağmen bazı ek işlemler daha tanımlanmıştır. Bu işlemlerin bir listesi Çizelge 3.3'de gösterilmiştir.

- *Ek ilişkisel cebir operatörleri*: Ek operatörlerin ilişkisel cebire kazandırdığı fazladan bir işlev yoktur. Ancak ifadelerin kısaltılmasında ve karmaşıklıkların azaltılmasında fayda sağlarlar.

Çizelge 3.3. Ek ilişkisel cebir operatörleri [31]

Operatör	Açıklama
Kesişim Küme (∩)	İki ilişkinin ortak elemanları
Doğal Birleştirme (⋈)	İki tablonun birleştirilmesi
Şartlı Birleştirme(⋈c)	İki tablonun ek bir şart üstünden birleştirilmesi
Bölüm(÷)	Bir tablonun alt kümesi sütunlardan oluşan başka bir tabloya bölümü fark sütunları içeren yeni bir tablo verir.
Atama(←)	İlişkileri geçici bir değişkene aktarmak için kullanılır

3.1.4.İlişkisel sorgu iyileştirmesi süreci

Örnek iki tablo üstünden bir sorgu optimizasyonun nasıl geliştiği şu şekilde gösterilebilir:

```
Yazar (yazarKod: integer, yazarAd: string, biyografi: string,
yas: real)
Kitap(kitapKod: integer, yazarKod: integer, tarih: date, KitapAd:
string, ilKod:integer)
Kategori(kategoriKod:integer,kategoriAd:string)
Il(ilKod:integer, ilAd:string)
```

Yazar Tablosunun diskteki dağılımının şu şekilde olduğunu varsayılmaktadır:

- Her satırda 50Byte ve her fayfada 80 satır veri olmak üzere toplam 500 sayfa veri
- Kitap tablosunun diskteki dağılımının şu şekilde olduğunu varsayılmaktadır:
- Her saftada 40 Byte veri ve her sayfada 100 satır veri olmak üzere toplam 1000 sayfalık veri
- Tampon havuzunda(buffer pool) en fazla 5 sayfa bilgi tutulabildiğini varsayılmaktadır.

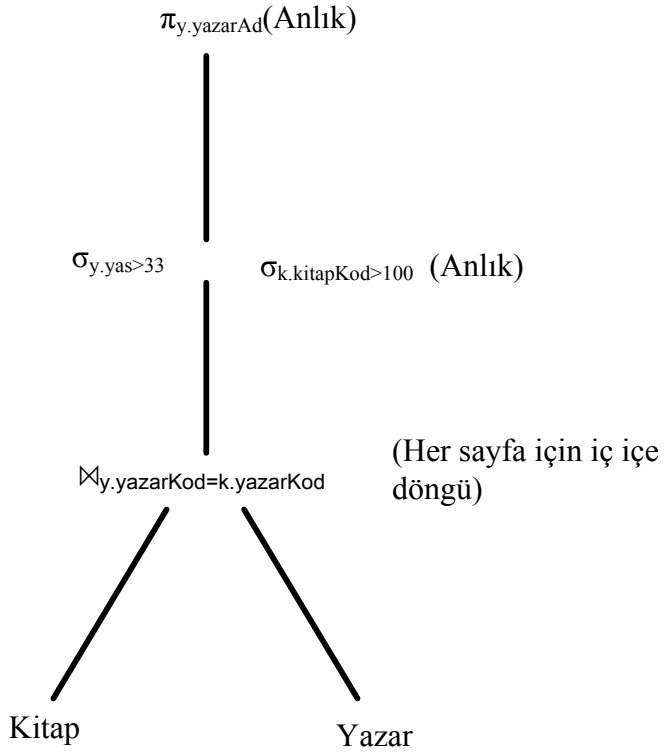
Burada geçen Sayfa birimi, İlişkisel veritabanı yönetim sistemlerinin disk erişim işlemlerini hızlandırmak için diskte oluşturduğu, genellikle VTYS'lerce 8060Byte olarak ayrılan bölmeye verilen addır.

Sorgu motoruna aşağıdaki ifadenin gönderildiğini varsayılırsa,

```
SELECT  Y.yazarAd
FROM  Yazar Y, Kitap K
WHERE  Y.yazarKod = K.yazarKod AND
    K.kitapKod=100 AND Y.yas>33
```

Bu türden bir sorgunun neticesini verebilecek alternatif birkaç sorgu planı aşağıdaki şekilde verilebilir:

Başlangıç sorgu planı aşağıdaki şekilde olabilir:

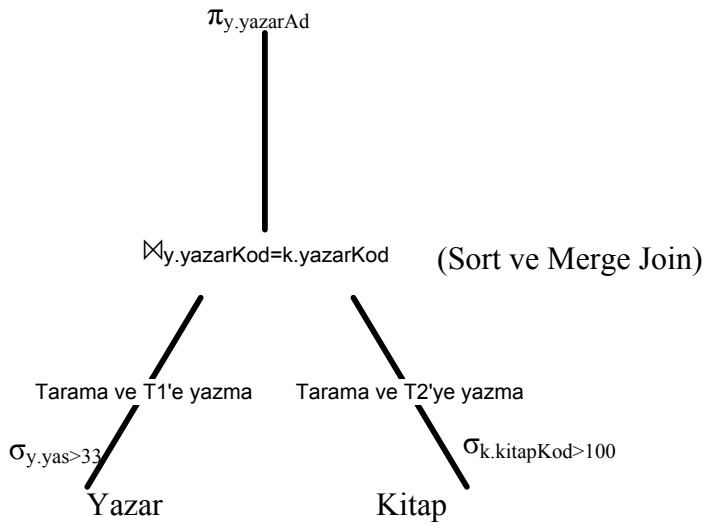


Şekil 3.3. İfadeden ayrıştırılmış sorgu ağacı [31]

- Maliyet (500 + 500*1000 adet I/O)
- En kötü planlardan biridir.
- Birçok olumlu durumdan yararlanamamaktadır.

Alternatif bir sorgu planı aşağıdaki şekilde verilebilir:

- Varsayım (seçmeleri sorgu ağacında aşağıya itelemek, IO maliyetini azaltacak bir etki yapar.)

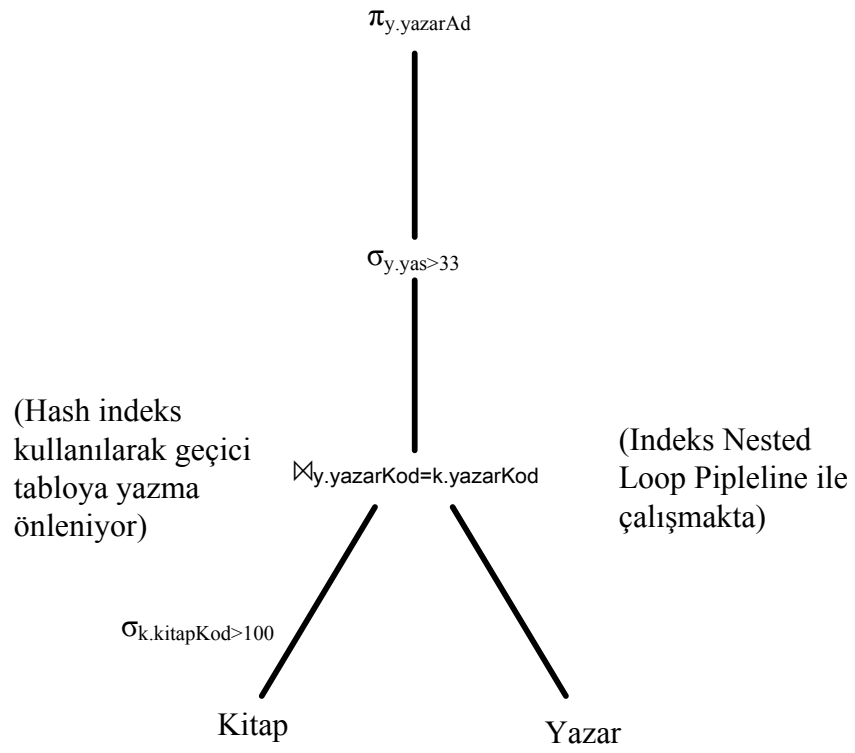


Şekil 3.4. Alternatif plan için oluşturulmuş bir sorgu ağacı [31]

5 birimlik tampon ile plan maliyeti:

- Kitap Tarama (1000) + temp T1'e yazma (10 sayfa normal dağılıma uygun olduğunu varsayıyoruz)
- Yazar Tarama (500) + temp T2'ye yazma (250 sayfa, şayet 10 farklı yazar varsa ve normal dağılmışsa)
- Sıralama T1 (2*2*10),
- Sıralama T2 (2*3*250),
- Birleştirme (10+250)
- Toplam(3560 sayfa I/O'luk bir maliyetle) sorgu neticelendirilebilir.

Başka bir alternatif çalıştırma planı şu şekilde olabilir. Sorgularda iyileştirme yapılırken, en etkin faktörlerden biri de indekslerdir. Buraya kadar indekslerin olmadığını varsayarak bir iyileştirmenin maliyetini azaltmak için neler yapılabileceğini ele aldık. Bir indeks olduğunda, sorgu maliyetini daha da aşağılara çekmek mümkündür.



Şekil 3.5. İndeks kullanımı ile iyileştirilmiş sorgu ağacı [31]

Kitap.kitapKod üstünde tanımlı bir clustred Indeks olursa 100,000/100 = 1000 satır ve 1000/100 = 10 sayfa kullanılıyorsa, Indexed Nested Loop ile pipelining yapılırsa, geçici tabloya yazma maliyeti azalacaktır. Ayrıca, yaş>33 ifadesini aşağı itelemek çok fayda sağlamayacaktır. Yazar.yazarKod indekslenmiş sütun olsun.

Plan Maliyeti;

- Kitap Satırlarının Seçilmesi (10 I/O)
- Her bir kitaba karşılık, 0.2 yazar yer almakta(500/1000=0.2)

- Her bir yazarın seçilmesi için 1000 I/O
- Toplam 1000*(1,2) + 10 = 1210 I/O ile seçim yapılabilir.

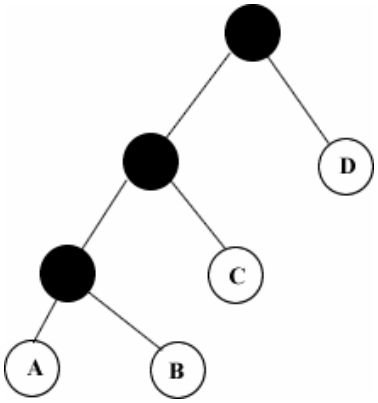
Görüldüğü gibi, sorgu iyileştirmesinde en önemli noktalardan biri veriler hakkındaki istatistik ve indekslerin doğru oluşturulup kullanılması iken bir diğeri ve daha önemlisi birden fazla tablodan veri gösterilecekse bu tabloların hangi sırada birleştirileceğine karar vermektir. Tez çalışmasında özellikle bu konu üstünde durulmuştur.

3.1.5.Birleştirme sıralaması problemi

Birden fazla tabloyu bir arada sorgulamak gerektiğinde tabloların birleşme işlemine katılım süreleri, sorgu iyileştirmesinden anahtar bir nokta olarak önem kazanmaktadır. Birden fazla tabloyu birlikte sorgulamak gerektiğinde temel birkaç birleştirme çözüm uzayı kabul görmüştür.

Sistem R iyileştirmesi veya Left-Deep Join yaklaşımı

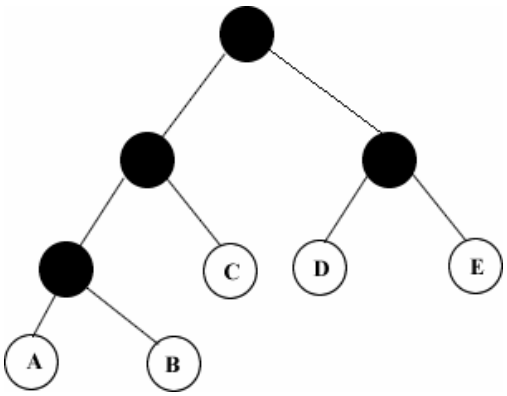
Sistem-Riyileştirmesi, çok fazla sayıda sorgu planı için kestirim yapmak yerine sadece soldan devam eden plan(left-deep-join) üstünden devam edilir [31]. Bu yöntemin en büyük avantajı, bir önceki birleştirmeden elde edilen sonuçların bir sonraki birleştirmede doğrudan ele alınabilmesi nedeniyle geçici bir tabloya yazma ve tablodan okuma gerektirmemesidir. Zaafı ise, 16'dan fazla tablo içeren sorgularda tepki süresinin aşırı şekilde yavaşlayarak IO maliyeti nedeniyle sonsuza yaklaşmasıdır [17]. Neticede az sayıda tablo sorgularken belirgin bir I/O işlemi azalması ve budama nedeniyle de daha az olasılık üstünden daha kısa sürede sonuca ulaşılmasını sağlayan bir algoritmadır. Bir diğer faydası, birleştirmeye giren tablolar arasında kartezyen çarpımından kaçınılmasına olanak verir.



Şekil 3.6. Sistem-R birleştirme uzayının gösterimi [32]

Çalı ağacı çözüm uzayı

Çalı Ağacı birleştirmesinde bir önceki sonuçların tutulabileceği bir geçici hafıza kullanımı gereksinimi olmasıdır. Sistem-R veya Left-Deep yaklaşımı, iyi sonuç gelme olanağı olacak şekilde en soldan her seferinde tek tablo birleştirerek bütün tabloları ekleme şeklinde bir yöntem iken, daha kötü sonuçlarla birlikte daha iyi sonuçların da gözden kaçırılmasına neden olabilen bir birleştirme yöntemidir [31].



Şekil 3.7. Çalı Ağacı birleştirme uzayının gösterimi [30]

Oysa olasılıklar da değerlendirildiğinde daha iyi birleştirme sıralamaları elde edilebilir olduğu K. Ono ve arkadaşları tarafından ispat edilmiştir [33]. Yapılan araştırmaya göre, n birleşmeye katılacak tablo sayılarını göstermek üzere, $(n^3-n)/6$ farklı çalı ağacı birleştirmesi yapılabilir durumdadır. Ancak bu sayı Sistem-R için

sadece $(n-1)^2$ ile sınırlıdır. Öte yandan bu çalışmada yapılan beşgen yıldız grafların sorgu maliyetlerinin en kötü durumda bile kabul edilebilir durumda olduğu gözlemlenmiştir. Beşgen yıldız ile taranabilen birleştirme uzayı ise $(n-1)2^{(n-2)}$ olarak hesaplanmıştır ki bu da Sistem-R'ye göre oldukça büyük bir tarama uzayının varlığına işaret etmektedir. Şüphesiz ki tarama uzayının büyümesi nedeniyle çalı ağacı yaklaşımında birleştirme planı üreticisine daha fazla yük binmektedir. Ancak daha iyi çözüm ihtimalleri de değerlendirilmiş olur.

Vance ve arkadaşları tarafından yapılan, çalı ağacı birleşim sıralamasında kartezyen çarpımı yönteminin kullanılması bu alanda yapılmış deterministik bir çalışma olarak verilebilir [34].

Birleştirme uzaylarının oldukça fazla olması, deterministik ve non-deterministik çeşitli birleştirme uzayı tarama algoritmalarının geliştirilmesine neden olmuştur. Bu algoritmaları deterministik ve non-deterministik olmak üzere iki ana başlık altında ele alınacaktır.

Birleştirme sıralaması problemine deterministik çözümler

Bu alanda yapılan çalışmalardan en ünlüsü Dinamik Programlama, Sistem-R' deki ağaç yapısını soldan başlayarak üretir [34, 36]. Her seferinde bir yeni tablo ekleyerek çözüme ilerler. Bir başka yaklaşım olarak, E.Wong ve arkadaşları tarafından Sistem-R'nin daha basit bir uygulaması ortaya atılmış ve Minimum Seçim yöntemi olarak kabul görmüştür [36]. Buna göre orta seviyedeki düğümlere mümkün oldukça küçük tutmak hedeflenir. En az sayıda seçim yapılacak tablo en başa alınarak, daha sonraki aşamalarda daha az kaydın taranması esasına dayanır.

Bu yöntemlerin dışında çalışmayı yapanların soyadlarının baş harfleri olan Krishanmurthy, Boral ve Zaniolo 'den oluşan KBZ Algoritması bu alanda başka bir örnek olarak gösterilebilir [37]. KBZ algoritması, T.Abaraki ve arkadaşları tarafından yapılan çalışmadaki en iyi sıralamanın polinomal zamanda hesaplanabileceğini gösteren araştırmanın yardımcı kuramlarla $O(n^2)$ olarak

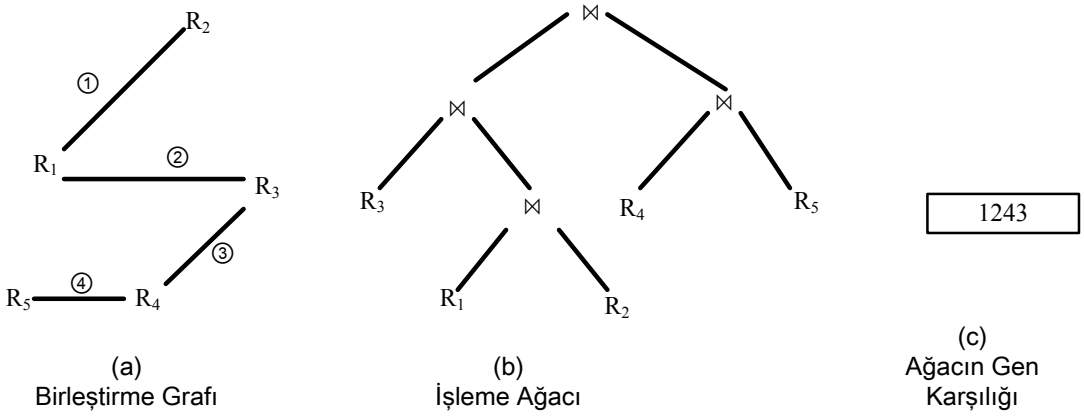
hesaplayabildikleri araştırma olup Sistem-R dışındaki uzayda çalışan bir örnek olarak ele alınabilir [38] .

Birleştirme sıralaması problemine rastsal çözümler

İlişkisel veritabanı yönetim sistemlerinde meta sezgisel yöntemlerin sorgu iyileştirmesinde kullanıldığı çalışmada demir tavlama benzetimini başarıyla kullanmıştır [39]. D.E. GoldBerg tarafından yapılan çalışmada, genetik algoritmanın arama optimizasyonu ve makine öğrenmesinde kullanılabileceği gösterilmiştir [40]. Y.E Ioannidis ve Y.C Kang, çok sayıda birleşim gerektiren sorgularda rastsal algoritmaların kullanılabileceğini ortaya atmışlardır [39].

Genetik Algoritmanın ilişkisel sorgu iyileştirmesinde kullanımı

1991 yılında, K.Bennettt ve arkadaşları, ilişkisel veritabanı yönetim sistemleri için çok sayıda birleşim gerektiren sorgularda birleşim sırası tayininde genetik algoritmayı kullanan Left-Deep ve Bushy stratejilerine dayalı alternatif araştırmalar yapmış, özellikle artan sayıda girdiden oluşan birleştirmelerde performans artışı sağlamayı başarmıştır. Bu araştırmada bir tablo sıralamasının gen dönüşümü ve sıralamaları şekil 3.9'da gösterildiği gibi yapılmıştır.



Şekil 3.8. Tabloların birleşim sıralamalarının gen halinde kodlanması [41]

1992 yılında Steinbrunn ve arkadaşları tarafından yapılan çalışmada bugüne kadar yapılan rastsal ve heuristic birleştirme sıra tayin algoritmalarının bir genel değerlendirmesi yapılmış ve diğer heuristic tekniklerin yanı sıra genetik algoritmanın da birleştirme sıra tayininde kullanımının başarılı bir örneğini ortaya koymuşlardır [11] .

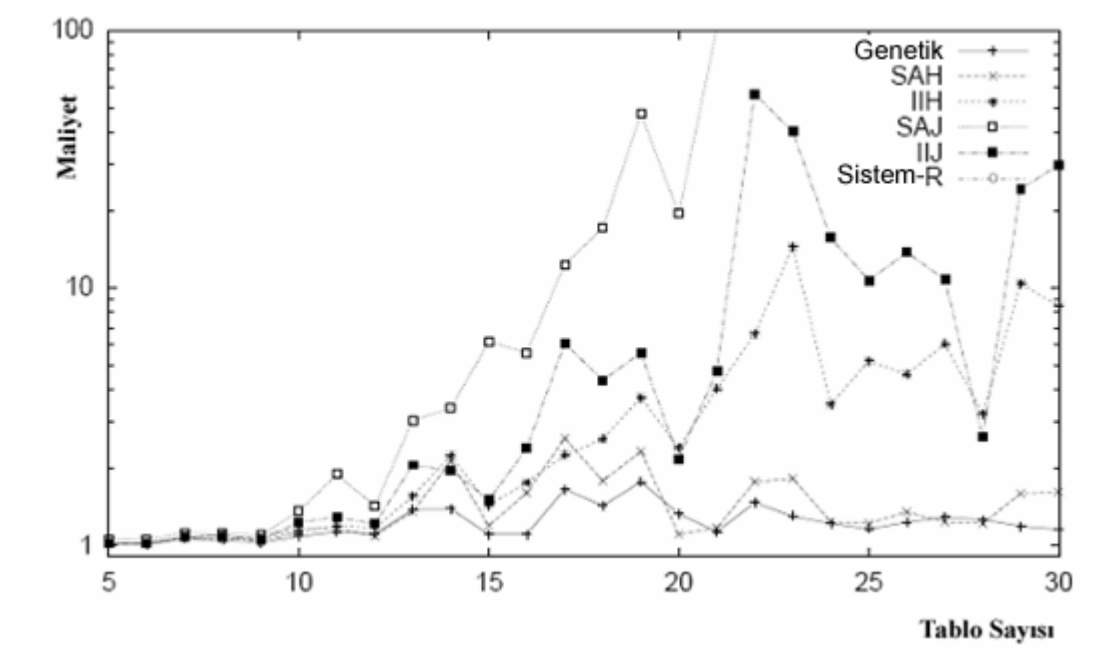
Araştırmanın Genetik Algoritma'nın uygulanması için yapılan düzenlemeler aşağıdaki şekildedir:

Öncelikle, birlikte sorgulanan her bir tabloya bir numara verilerek, Genetik Algoritma'da gen adı verilen birim yapı elde edilmektedir. Bu yapının maliyeti kestirimsel olarak hesaplandıktan sonra, rulet tekerleği kullanılarak, sonuca yatkınlığı doğrultusunda genlere şans verilmekte ve bir seçime tabi tutulmaktadır. Ardından bir dizi çaprazlama ve seçme işlemlerine tabi tutularak, en iyi bireyin elde edilmesi işlemi gerçekleştirilmektedir. Çaprazlama için iki nokta kullanılmakta ve bir mutasyon işleminden sonra değerlendirilmektedir. Steinbrunn ve arkadaşları tarafından yapılan çalışmada, genetik algoritma ile ilgili parametreler şu şekilde ele alınmıştır:

- Çözüm Uzayı (Sistem-R ve Çalı Ağacı)
- Kodlama (Dalların Sıralı Listesi / Sıralı Numara Kodlama)
- Çözüme Yatkınlığa göre seçilme şansı (Rulet Çemberi)
- Sıra Değiştirmeli çaprazlama operatörü
- 128 bireylik popülasyon
- Herhangi bir bireyin çaprazlamaya giriş şansı (%65)
- Mutasyon Oranı (%5)
- Sonlandırma Şartı (İyileşme sağlamayan 30 iterasyon ve iyileşme sağlamayan 50 iterasyon)

Bu çalışmada, Sistem-R stratejisine dayalı genetik algoritma, birleştirme sıra tayininde kullanılan diğer rastsal yöntemler arasında düşük maliyeti ile ön plana

çıkmaktadır. Bu araştırmaların sonuçlarına dayalı grafik Şekil 3.10'da görüldüğü gibidir.



Şekil 3.9. Çeşitli algoritmaların tablo sayısına bağlı maliyet grafiği [41]

3.2.XML Sorgulama Yaklaşımlarının Genel Değerlendirmesi

XML'in ilk yıllarında, OQL, XML-QL, XPath, XQL, ve benzeri gibi genel kabul görmüş sorgulama yöntemleri kullanılmakta idi [5, 6].

XML verinin kullanımının yaygınlaşmasının ardından, bu veriyi sorgulama ihtiyacı bu yöntemler tarafından tam olarak karşılanamaz hale geldi. 1999 yılında W3C tarafından verileri sorgulama için genel kabul görmüş bir standart geliştirme amacıyla oluşturulan çalışma grubu IBM, Microsoft, Oracle ile diğer küçük üreticiler ve bazı saygın akademisyenlerin de desteğiyle XQuery için 2002 yılında bir standart taslağı yayınladı [5]. Bu yayının ardından, XML sorgulama için bu taslak kullanılmaya başlanmış ve bu alanda birçok araştırma projesi başlatılmış veya ek özelliklerle bu standart taslağı ile uyumlu hale getirilmiştir [10, 11, 25].

3.2.1.XML sorgu optimizasyonu yaklaşımları

XQuery sorgularının optimize edilmesi iki farklı eksen etrafında şekillenmektedir. Bunlardan ilki, sorguyu öncelikle SQL ifade domainine çevirip daha sonra da SQL ifadesi olarak optimize etme şeklinde karmaşık bir yapı şeklindedir [42].

Aşağıda bir XQuery ifadesi ve bu ifadenin SQL'e çevrilmiş hali gösterilmiştir.

```
for $d in //yazar, $m in //kitap
where $d/yazarKod = $m/kitapKod
return  <KitapYazar yazarKod="{$d/yazarKod}"
kitapKod="{$m/mad}"/>
```

ifadesi yerine eşdeğer SQL ifadesi olarak

```
SELECT d.yazarKod, m.KitapKod
FROM Yazar d, Kitap m
WHERE d.yazarKod= m.yazarKod
```

kullanılabilir. Daha sonra, bu yeni SQL ifadesi bilindik ilişkisel cebir üstünden optimize edilerek, sorgu için bir çalıştırma planı elde edilebilir. Özellikle zaman içerisinde daha çok kabul gören yöntem, doğrudan XML cebiri çerçevesinde yapılan sorgu iyileştirmesidir. Doğrudan sorgunun XML Cebirine göre planları oluşturulur. XML Cebirine göre operatör eşdeğerler kullanılır ve XML Veriye erişim-işleme maliyeti hesaplanır. Ancak bu noktada birçok ferdi yaklaşımlar mevcuttur, oturmuş bir XML-XQuery cebiri henüz bulunmamaktadır. En çok kabul gören XQuery cebirlerinden biri, XAM(XML Access Module) 'dür. Timber'da kullanılan cebir de bu temele dayanmaktadır [10].

Bu alanda yapılan iki önemli araştırma olarak Galax ve Timber'i, yapılan tez çalışmasının temelini oluşturmaktadır [10, 11].

İlişkisel Sorgu Üstünden XQuery iyileştirmesi

XML ile ilişkisel sorguların optimize edilmesindeki yemel yaklaşımlar birbiri ile parelellik göstermektedir. Öte yandan yapılan bazı çalışmalarda XML cebiri ile yapılan iyileştirmelerin, ilişkisel yöntemlere nazaran daha iyi veya daha kötü sonuçlar doğurabileceği tespit edilmiştir [43].

Şema Bazlı İyileştirme

XQuery ile iyi yapılandırılmış veriler sorgulandığı için, bazen şema üstünden sorgu optimizasyonu yapmak mümkündür [43]. Bu türden bir örnek aşağıda yer almaktadır:

```
<!ELEMENT Students (Student*)>
<!ELEMENT Student (Name, Address, Birthday)>
<!ELEMENT Address (Street, City, Zip,
(Tel|Email))>
```

indirgenmiş ifade

```
//Student[Birthday]/Address[Tel|Email]
//Student/Address
```

Yukarıda bir DTD tanımı ile tipi belirtilen bir XML dokümanı ve bu doküman üstünde verilen bir sorgunun şemadan yararlanılarak basitleştirilmiş halini göstermektedir. Bu basitleştirme, her bir XML elemanının Tel ve Email değerlerinin olmasının zorunluluk şeklinde tanımlanmasına dayanmaktadır. Çünkü şemaya göre her bir öğrencinin adresi, telefonu ve e-mail bilgisi olmak zorundadır. Böylelikle şemaya göre sorgu iyileştirmesi yapılmış oldu.

3.2.2. XAL XML cebir sistemine genel bakış

Buradaki örnekler için iki farklı XML dosyası kullanılmaktadır. Bunlardan ilki kitaplar ile ilgili XML veri, diğeri XML verinin tip ve şema tanımlarını içeren XSD dokümanıdır.

XML veriler aşağıdaki şekilde alınmıştır

```
<bib>
 <book>
   <title> Data on the Web
   </title>
 <year>1999</year>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
</book>
<book>
 <title>XML Query</title>
 <year>2001</year>
 <author>Fernandez</author>
 <author>Suciu</author>
 </book>
</bib>
```

Yukarıda gösterilen XML dokümanının tip tanımlarını belirleyen XSD dokümanı aşağıda gösterilmiştir.

```
<xsd:group name="Bib">
 <xsd:element name="bib">
 <xsd:complexType>
   <xsd:group ref="Book" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:complexType>
 </xsd:element>
</xsd:group>
<xsd:group name="Book">
```

```
 <xsd:element name="book">
  <xsd:complexType>
   <xsd:element name="title" type="xsd:string"/>
   <xsd:element name="year" type="xsd:integer"/>
   <xsd:element name="author" type="xsd:integer"
maxOccurs="unbounded"/>
  </xsd:complexType>
 </xsd:element>
</xsd:group>
```

Bu verilere göre, XML cebirsel ifadeleri aşağıdaki gösterildiği şekilde tanımlanabilir:

```
type Bib = bib[Book{0,*}]
type Book = book[
                title [ String ],
                year [ Integer ],
                author [ String ]{1,*}
          ]
let bib0 : Bib = bib [
          book [
                title [ "Data on the Web" ],
                year [ 1999 ],
                author [ "Abiteboul" ],
                author [ "Buneman" ],
                author [ "Suciu" ]
          ],
          book [
          title [ "XML Query" ],
          year [ 2001 ],
          author [ "Fernandez" ],
          author [ "Suciu" ]
          ]
     ]
```

Bu ifadelerde kitap şeması şu şekilde gösterilmekte:

```
 book0 : Book =
```

```
book [
      title [ "Data on the Web" ],
      year [ 1999 ],
      author [ "Abiteboul" ],
      author [ "Buneman" ],
      author [ "Suciu" ]  ]
```

İzdüşüm(Projection)

Ele alınan çalışmada, izdüşüm operatörünün gerçeklenmesi XPath'deki yol takibine benzer bir ifade ile gerçekleştirilmektedir.

Örneğin, Book elamanının içindeki bütün kitapları getirecek bir yol ifadesi, iz düşürme operasyonudur ve örnekler için aşağıdaki sonucu verir.

```
bib0/book/author

author [ "Abiteboul" ],
       author [ "Buneman" ],
       author [ "Suciu" ],
       author [ "Fernandez" ],
       author [ "Suciu" ]   : author [ String ] {0,*}
```

Sonuç ile ilgili önemli noktalar:

- Doküman sırası sonuçta korunur.
- Birden fazla aynı değer varsa, korunur.
- Her bir eleman birden fazla alt eleman içerebilir veya hiç içermeyebilir.

Sorgu, aşağıda gösterilen aşamalardan geçerek neticelendirilmektedir.

```
bib0 : Bib
bib0/book : Book{0,*}
bib0/book/author : author [ String ]{0,*}
```

İterasyon

Doküman içerisindeki elemanlar üstünde gezinilerek elemanların değerlerini yeni içeriklere değiştirebilmek mümkün olur.

Örneğin, her bir kitabın, yazarını başlığından önce gösteren ve yılını göstermeyen bir iterasyon aşağıdaki gibi olabilir.

```
for b <- bib0/book in book [ b/author, b/title ]
```

işlemin sonucu şu şekilde olacaktır:

```
book [
     author [ "Abiteboul" ],
     author [ "Buneman" ],
     author [ "Suciu" ],
     title [ "Data on the Web" ]
],
book [
     author [ "Fernandez" ],
     author [ "Suciu" ],
     title [ "XML Query" ]
]
     : book [
           author[ String ]{1,*},
           title[ String ]
     ]{0,*}
```

Tip sistemi b'nin her zaman kitap olduğunu algılayabilir. Böylece b/author her zaman author[String]{1,*} 'dir ve b/title da title[String] tipindendir.

Sorgunun tip tanımı ile ilgili geçtiği aşamaları aşağıda gösterilmiştir.

```
bib0/book : Book{0,*}
b : Book
```

```
b/author : author [ String ]{1,*}
b/title : title [ String ]
```

Seçme

XQuery'de bir yükleme bağlı olarak bazı değerleri seçmek için WHERE operatörü kullanılır.

Örneğin, 2000 yılından önce yayınlanan, bib0 altında yer alan bütün kitapların listesini alacak bir sorgu aşağıda verilmiştir.

```
for b <- bib0/book in
where value(b/year) <= 2000 then b
```

Where-then dönüşümü ifadesi aşağıda gösterildiği gibi değiştirilir ve gösterilen kural çerçevesinde sorgu şekil değiştirir.

```
where e1 then e2

if e1 then e2 else ()

for b <- bib0/book in
if value(b/year) < 2000 then b else ()
```

Sonuç şu şekilde olacaktır:

```
book [
     title [ "Data on the Web" ],
     year [ 1999 ],
     author [ "Abiteboul" ],
     author [ "Buneman" ],
     author [ "Suciu" ]
] : Book{0,*}
```

Birleştirme

Birleştirme operatörü, bir veya daha fazla dokümandan gelen verileri birleştirmek için kullanılır.

Örneğin, Review0 ve bib0 kaynaklarını birleştirerek Kitap adı, yazar adı ve yorumları tek sonuç olarak göstermek için Şekil 3.27'de gösterildiği gibi verilmiş olsun.

```
for b <- bib0/book in
  for r <- review0/book in
  where value(b/title) = value(r/title) then
   book [ b/title, b/author, r/review ]
```

Kitaplar hakkında verilerin yer aldığı aşağıdaki şema tanımına uygun bir veri kaynağı olduğu var sayılmaktadır.

```
type Reviews =reviews [
                book [
                title [ String ],
                review [ String ]
               ]{0,*}
]
```

Bu şemaya uygun olarak, aşağıda gösterilen verilerin mevcut olduğu var sayılırsa

```
review0 : Reviews =
     reviews [
           book [
           title [ "XML Query" ],
           review [ "A darn fine book." ]
     ],
     book [
           title [ "Data on the Web" ],
           review [ "This is great!" ]
```

```
      ]
]
```

verilen sorgu aşağıda gösterilen sonucu verecektir:

```
book [title [ "Data on the Web" ],
      author [ "Abiteboul" ],
      author [ "Buneman" ],
      author [ "Suciu" ]
      review [ "A darn fine book." ]
],
book [title [ "XML Query" ],
       author [ "Fernandez" ],
       author [ "Suciu" ]
       review [ "This is great!" ]
]
: book [title [ String ],
        author [ String ] {1,*},
        review [ String ]
] {0,*}
```

Görüldüğü gibi XAL, SQL benzeri bir cebir sistemi önermektedir.

3.2.3 TAX XML cebir sistemi

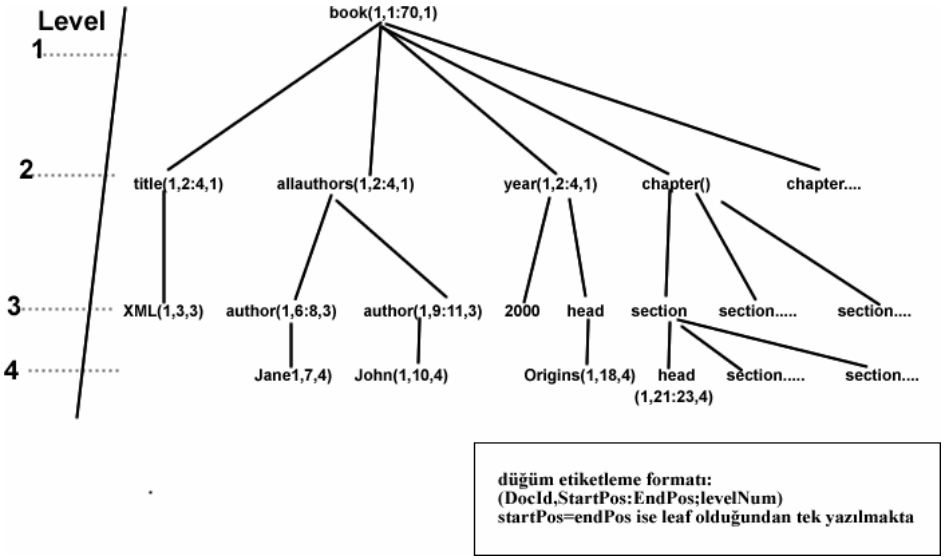
TAX, XAL'a göre ilişkisel veri cebirinden daha bağımsız bir veri modeli sunar ve bu veri modeli üstüne kurulu bir ilişkisel cebir önerir. Bu nedenle, TAX'ın veri modeline bir göz atmak gerekir [42].

TAX veri modeli ve Structural Join

TAX cebirine göre, bir veri ve bu veri üstündeki sorgu ağaç modeli ile Şekil 3.10'da gösterildiği gibi düzenlenmekte. Aşağıdaki gibi bir örnek XML dokümanı ele alınmış olsun.

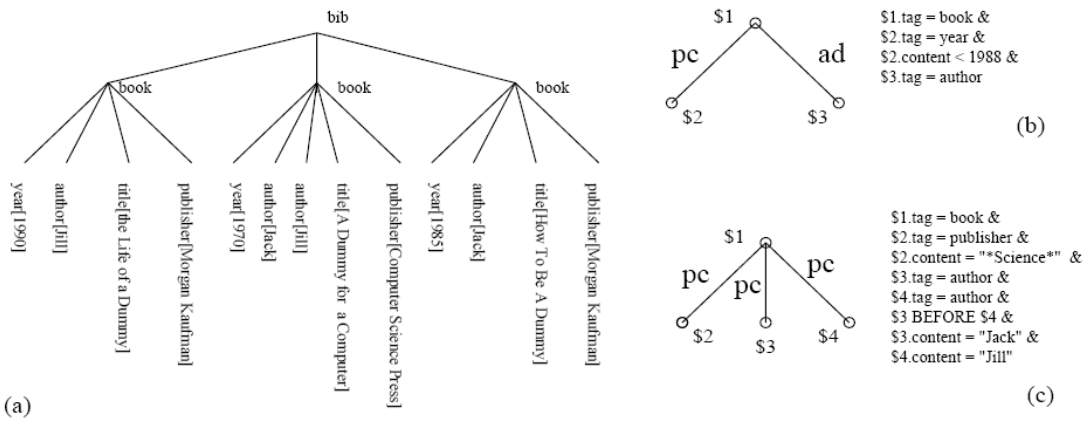
```
<book>
     <title> XML </title>
     <allauthors>
           <author>Jane</author><author>John</author>
     </allauthors><year>2000</year>
     <chapter>
           <head> Origins</head>
           <section>
                 <head>...</head>
                 <section>...</section><section>...</section>
           </section>
     </chapter>
     <chapter>
           <head>Axes</head>
           <section>
                 <head>...</head>
                 <section>...</section>
                 <section>...</section>
           </section>
     </chapter>
...
</book>
```

Yukarıda gösterilen dokümanın veritabanına aktarım formatı Şekil 3.10'da gösterilmiştir. Bu şekle göre her bir düğüme, doküman tekil tanımayıcısı, başlama pozisyonu ve bitiş pozisyonu ile derinliği gibi ek tanımlayıcılara yer verilmektedir.



Şekil 3.10. Bir XML dokümanının veritabanına aktarılma formatı [10]

Verilerin sorgulanması aşamasında ise, pattern tree adı verilen yapılardan yararlanılmaktadır. Bu türden bir örnek Şekil 3.10'da gösterilmiştir. Bu şekle göre pattern tree'ler bir ağaç ve ağaçta yer alan alt-üst ilişkisi tanımından oluşur. Bir alt-üst düğüm ilişkisi şayet ata-soy türünden ise, ara düğüm sayısı önemsizdir ve ad işareti ile veya çift çizgi ile gösterilmektedir. Tersine alt-üst ilişkisi sadece anne-çocuk ilişkisi ise, bu durum sadece bir sonraki düğüm ile bir önceki düğüm şartını aramaktadır ve pc etiketi ile işaretlenmekte veya tek çizgi ile gösterilmektedir.



Şekil 3.11. Bir XML ağacı şeması ile üretilen iki pattern tree [42]

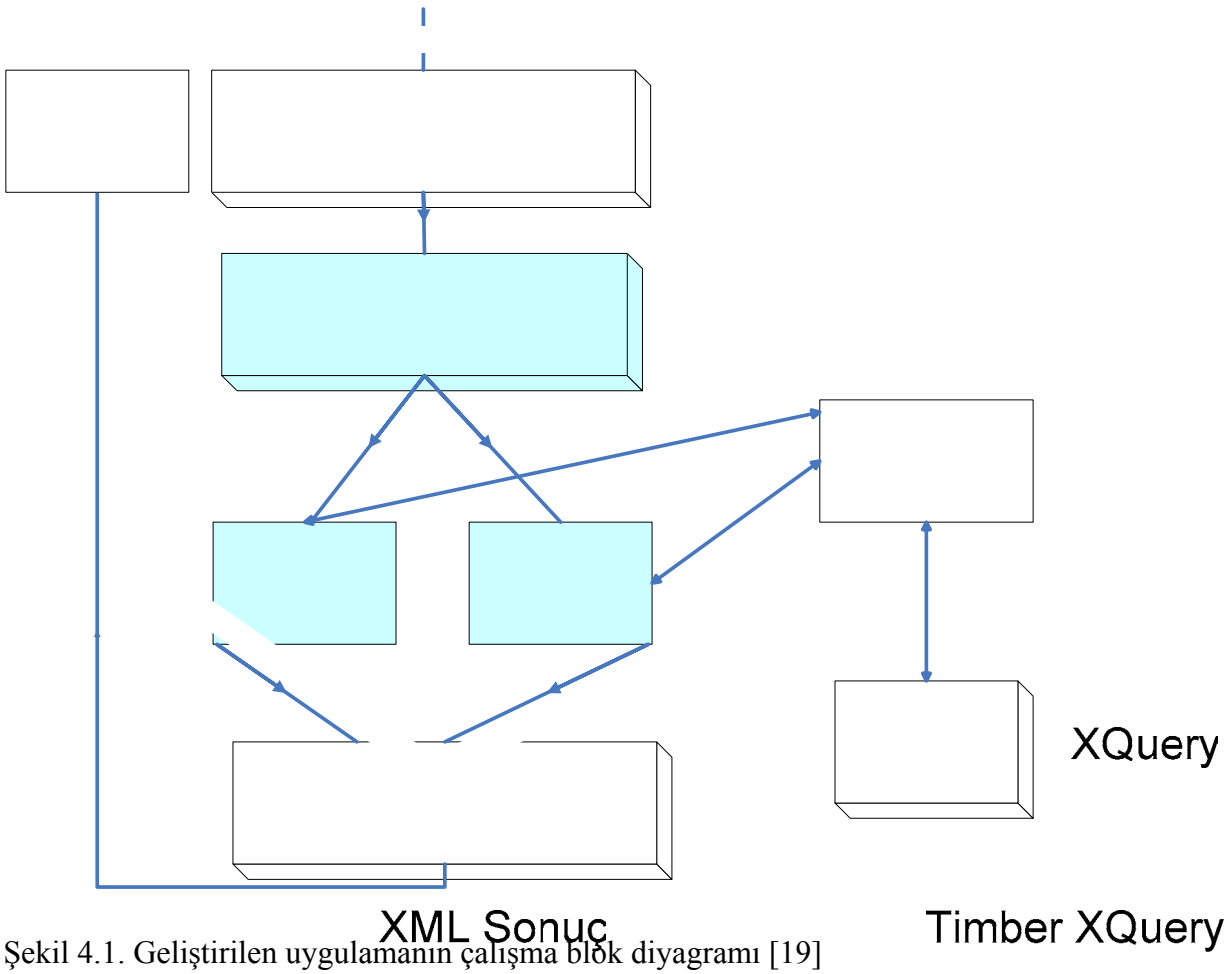
Şekil 3.11'de verilen gösterim, pattern tree'lerde yer alan ad ve pc etiketlerinin eşdeğeri düğümlerin daha kolay sorgulanması amacıyla eklenmiştir. İlgili şartları sağlayan etiket eşleşim formülleri Timber'ın temel cebirini anlatan çalışmada yer almaktadır [44].

Burada gösterilen pattern tree'lerin içerdiği veriler, veritabanından düğümlerin uygun şartlarda birleştirilmeleri ile bulunur. Bu yaklaşıma Structural Join yaklaşımı denmektedir [44]. Bu nedenle, TAX cebiri ile yapılan bir sorguda çok fazla sayıda birleştirme gereksinimi doğmaktadır.

Bu yaklaşımda da sorgulamaya girecek düğümlerin büyüklükleri ile ilgili bir sıralama sorununun mevcut olduğu, Wu ve arkadaşları tarafından yapılan çalışmada alternatif yaklaşımlar ile birlikte ele alınmaktadır [45]. Burada, Sturactural Join algoritmasında birleştirme sıralaması tayini için Rastsal Olmayan birleştirme sıralama tayin algoritmalarından Dinamik Programlama çeşitli ayarlamalarla kullanılmıştır. Yapılan tez çalışmasında da bu probleme genetik algoritma ve demir tavlama benzetimi uygulanarak iki sonuç birbiri ile kıyaslanmıştır.

4. UYGULANAN YÖNTEM

Yapılan çalışmada, Structural Join algoritması ile sorgu birleştirme sıralaması Genetik Algoritma ve Tavlama Benzetimi ile gerçekleştirilmiş ve bu sonuçlar birbiri ile kıyaslanmıştır. Yapılan araştırma için geliştirilen uygulamanın blok diyagramı Şekil 4.1'de gösterilmiştir.



Şekil 4.1. Geliştirilen uygulamanın çalışma blok diyagramı [19]

4.1.Genetik Algoritma ile Sorgu Optimizasyonu

Yapılan çalışmada K.Bennet ve arkadaşları tarafından önerilen düğümlerin farklı sıralamalarının farklı bireyler olarak esas alınması şeklindeki dönüşüm kullanılmıştır [17]. Bir başka ifade ile yapılan çalışma ilişkisel veritabanı sorgulamalarında

Logical Plan Pars

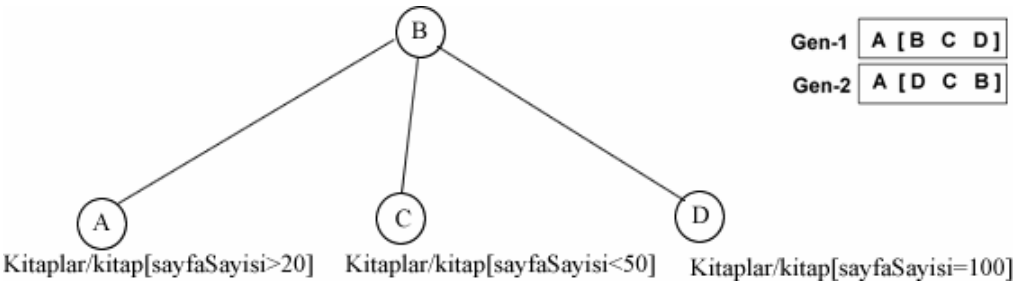
birleştirme sıralamasında genetik algoritma kullanançalışmalarının Structural Joinve XML veri sorgulamaya adapte edilmiş halini kapsamaktadır [17, 18, 44].

4.1.1.Genetik Algoritma'ya kısa bir bakış

Genetik algoritma'da, çözümün her bir parçasına gen denilmektedir. Genetik algoritma, aynı Genlerin çeşitli operatörler yardımıyla farklı dizilişlerini sağlayarak farklı bireyler elde etmekte ve bu bireylerden en iyi olanı seçmek şeklindedir. Birleştirme sıralaması tayini için genetik algoritmanın tercih edilmesindeki neden, taranması gereken seçeneklerin çok geniş olması ve bu alanda iyi bir çözümün yeterli oluşu, en iyinin elde edilme zorunluluğunun bulunmaması.

Gen ve birey

Genetik algoritmayı probleme uygularken, her bir düğüm bir gen olarak alınmıştır. Yine genetik algoritmada, genlerden her birinden eksiksiz-fazlasız sadece bir tane bulunduran her bir yapıya birey (kromozom) denilmektedir. Buna göre her bir düğümün bir defa yer aldığı dizilişlerden her biri bir birey olarak modellenmiştir. 4 genden oluşan örnek düğümler ve bu düğümlerden üretilen iki gen, Şekil 4.2'de gösterilmiştir.

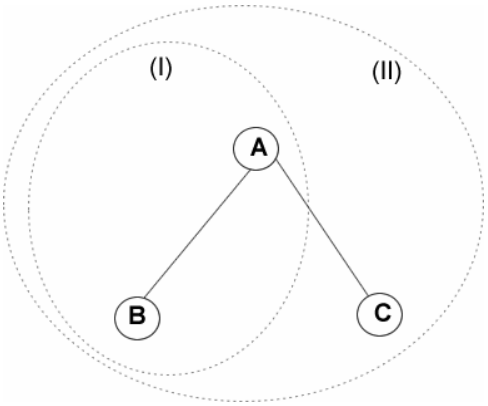


Şekil 4.2.Örnek bir düğüm ve bu düğümden elde edilen iki farklı gen [19]

Uygunluk fonksiyonu

Bireyler arasındaki farklılık genlerin farklı sıralanmasıyla ortaya çıkmaktadır. Her bir bireyin gen dizilişindeki bu farklılıklarının çözüm üzerindeki başarı oranını gösterecek bir fonksiyona gerek vardır. Bu fonksiyon probleme özgüdür ve modeldeki her bir bireyin çözüme uygunluk değerini hesaplamaktadır. Bu fonksiyona, uygunluk fonksiyonu (fitness function) denmektedir. Bu fonksiyondan elde edilen sonuca da bireyin uygunluk değeri denmektedir. Bu çalışmada, bir çalıştırma planının toplam maliyeti ceza puanı olarak kullanılmıştır. Böylece Structural Join yaklaşımında yer alan maliyet fonksiyonu, genetik algoritma için ceza temelli uygunluk fonksiyonu olarak kullanılmıştır [44].

A düğümü B düğümünün atası olmak üzere her seferinde sadece bir düğüm birleştirerek ilerleyen bir sorgu çalıştırma yöntemi kullanılmıştır. Yöntemin çalışması Şekil 4.3'de ayrıntılı olarak gösterilmiştir.



Şekil 4.3. A düğümüne ait B ve C çocuklarının birleştirilme sırası [19]

Şekil 4.3'de gösterilen A ve B şeklindeki iki düğümün maliyet fonksiyonu S. Al-Khalifa ve arkadaşları tarafından yapılan araştırmaya göre maliyet hesabında, işlemin bellekte yapılmasından dolayı I/O işlemine ek olarak CPU işlemlerinin maliyetine yönelik bir hesaplama yapılmıştır [44].

$2*|AB| * f\_io + 2 * |A| * f\_st$ (1)

Burada A düğümü B düğümünün atası olmak üzere,

|A| : A düğümünün kardinalitisi,

|AB| : A-B birleştirmesinin değerinin kardinalitisini,

f_io : IO işlemleri maliyeti

f_st : Sıralama maliyeti

değerlerini ifade etmektedir.

Popülasyon oluşturulması

Birden fazla bireyden oluşan topluluğa popülasyon denilmektedir. Literatürde, ilişkisel veritabanları için birleştirme sıralamasında genetik Algoritma kullanan yaklaşımlarda 128 bireyden oluşan popülasyonlarla uygulamaları görülmektedir[18]. Bu çalışmada problemin çözümünde en iyi birey sayısını bulmak için farklı sayılardaki bireyler için deneysel sonuçlar elde edilmiştir.

Herhangi bir bireyde, genlerin dizilişlerinde bir kurala bağlı olmaksızın, nedensiz ve düşük ihtimallerle meydana gelen değişikliğe mutasyon denmektedir. Düşük seviyede mutasyon oranı genetik çeşitliliği artırmaktadır. Ancak öte yandan artan mutasyon oranlarının iyi sonuçların yok olmasına neden olduğu yapılan çalışmada gözlemlenmiştir. %0,001 - %0,05 arasında farklı değerler alınarak en iyi değerin bulunması hedeflenmiştir.

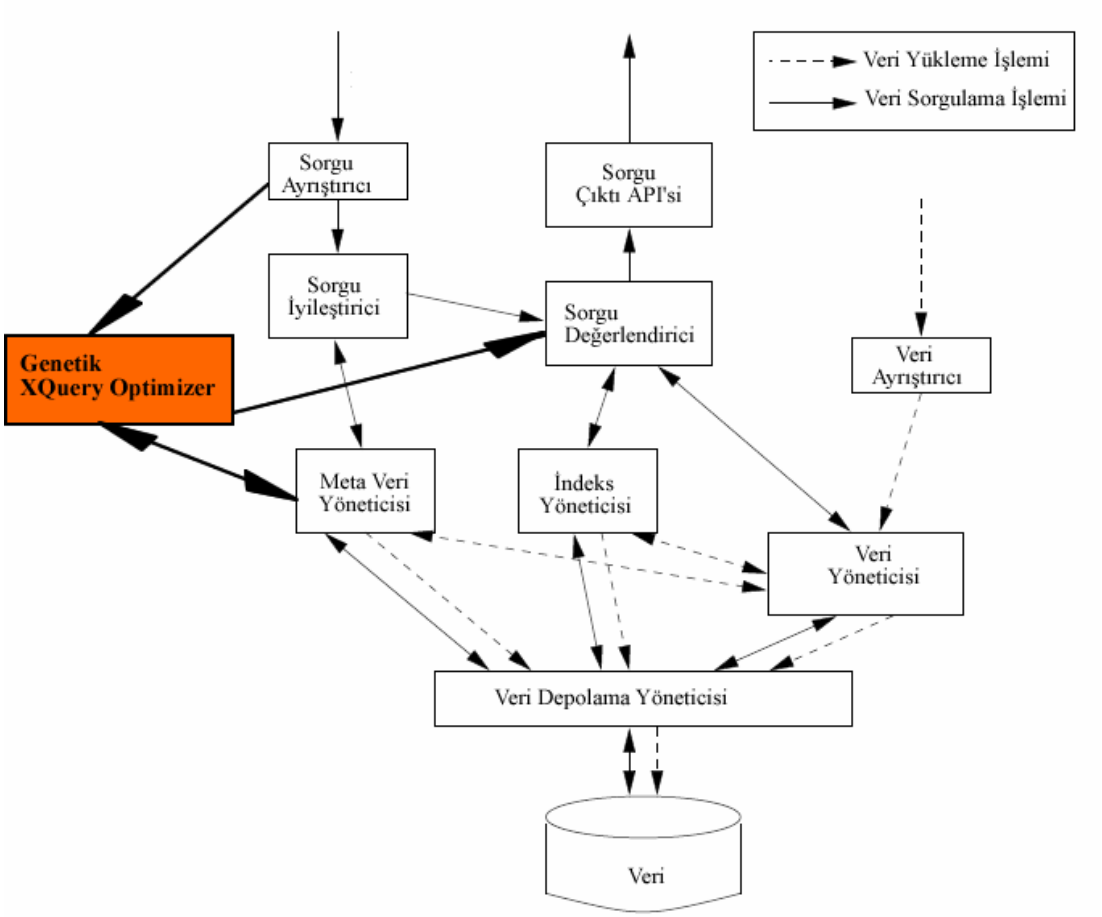
Çaprazlama işlemiyle, popülasyonda yer alan iki birey rasgele bir noktadan bölünüp parçaları karşılıklı değiştirilerek eski iki bireyden farklı iki yeni birey üretilmiştir. Bu işlemin sonucunda, her iki bireyde de tekrarlayan ve dolayısıyla da eksik kalan genler olabileceği gibi sorgunun anlamını değiştirebilecek bireyler olabileceği için çaprazlamanın sonucu bir tamir işlemine tabi tutularak, kromozomların anlamsal bütünlüğü sağlanmıştır. Yeni bir nesil, eldeki popülasyondan çaprazlama, mutasyon ve elitizm gibi kriter ve yöntemlerle elde edilmiştir. Her bir popülasyonda en iyi n birey, yeni popülâsyona hiç bir işleme tabi tutulmadan, elitizm kriteri çerçevesinde

aktarılmıştır. Bu çalışmada elitizm kriteri 3 olarak alınmıştır. Böylece genetik değişimlerle elde edilen iyi bireylerin kaybının önlenmesi sağlanmıştır.

4.1.2. Geliştirilen uygulamanın sisteme entegre edilmesi

Bu çalışmada genetik algoritmayla problemin çözümünün gerçekleştirilmesi için C#2.0 programlama dili ile açık kaynak Timber Veritabanı Motoru program bileşenleri kullanılmıştır. Timber veritabanının çalışma ilkeleri ve bileşenleri özet olarak Ek-1'de anlatılmıştır. Şekil 4.3'de Timber veritabanının, XML veriyi ayrıştırması ve sorgulama esnasında işlemler yapan bileşenlerinin blok diyagramları üstüne eklenerek gösterilmiştir [44]. Bu şekilde yapılan çalışmanın işlevsel konumunun anlaşılabilmesi için koyu renkli dolu dikdörtgen olarak blok diyagramda görülmektedir.

Şekil 4.4'e göre, gelen XML sorguları, Timber projesinde yer alan ayrıştırıcı (parser) tarafından ayrıştırıldıktan sonra, geliştirilen Genetik XQuery Optimizer tarafından ele alınarak yeniden optimize edilmesi şeklinde bir çalışma gerçekleştirilmiştir.



Şekil 4.4. Timber veritabanı bileşenleri ve çalışmanın işlevsel konumu [44, 19]

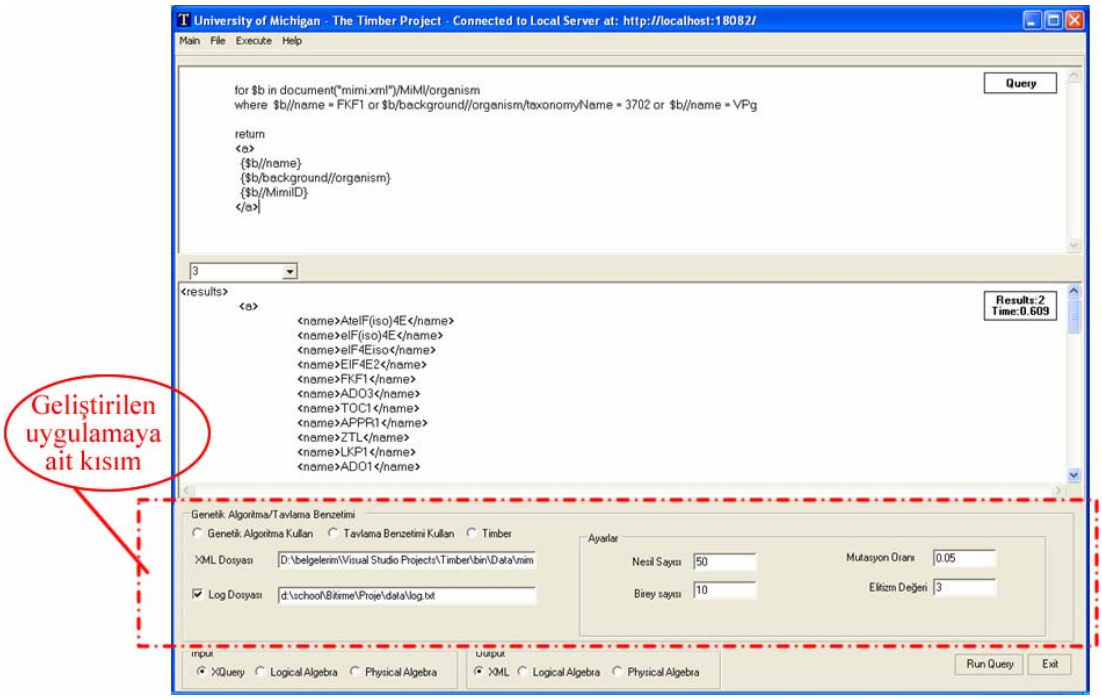
Uygulamanın bileşenleri

Geliştirilen uygulama, Timber tarafından ayrıştırılmış XQuery sorguları almakta ve bu sorgulardan yeni bir ağaç yapısı üretmekte. Arkasından bu ağaç yapısındaki düğümlerden her birini bir gen olarak ele alıp genlerden bir rastsal başlangıç popülasyonu elde edilmekte ve arkasından bu popülâsyondan yeni bireyler elde edilerek yeni bir popülasyona ulaşılmaktadır.

Bütün bu işlemleri gerçekleştiren uygulamanın programatik bileşenleri bu bölümde detaylandırılmaktadır. Geliştirilen uygulama Timber'ın standart arayüzüne ek işlevler olarak monte edilmiş hali Şekil 4.5'de gösterilmiştir. Bu şekle göre, Sorgu penceresine yazılan bir XQuery ifadesi,

- Genetik Algoritma ile
- Tavlama Benzetimi Algoritması ile
- Klasik Timber Yöntemi ile

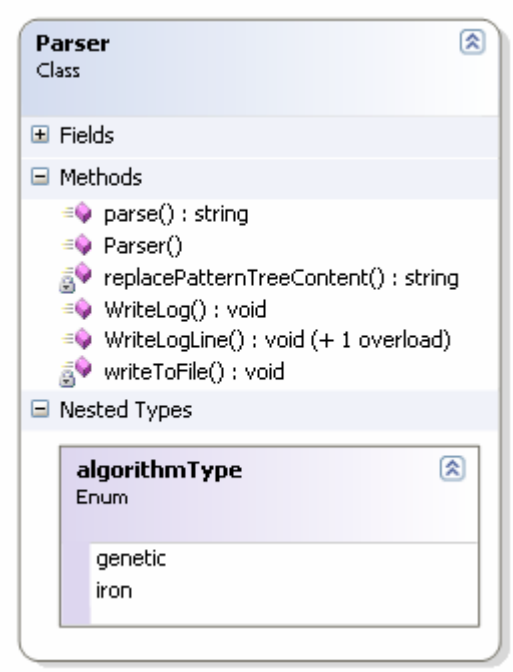
Optimize edilebilmektedir. Her bir algoritma için, parametrik değerler arayüzden ayarlanabilmekte ve herhangi bir işlemin ne kadar sürdüğü de aynı arayüzden okunabilmektedir.



Şekil 4.5. Geliştirilen uygulamanın Timber arayüzü ile entegrasyonu [19]

Parser sınıfı

Parser sınıfının diyagramı Şekil 4.6'de gösterildiği gibidir. Bu sınıf, kullanıcıdan gelen parametrelerin okunması işini yapar. Ayrıca Timber projesinde yer alan XQueryParser uygulamasının ürettiği ve mantıksal sorgu adı verilen(Logical Algebra) dili ifadelerini yorumlayarak yeni mantıksal sorgular üretir.



Şekil 4.6. Parser sınıfı için UML Class diyagramı [19]

Sınıfta yer alan başlıca metodlar ve işlevleri aşağıdaki gibidir:

- *parse*: Ana metod olup mantıksal ifadeleri ayrıştırıp yeni mantıksal ifadeler üreten metodudur. Bu işlem öncelikle girilen mantıksal ifadenin hiyerarşik şekilde bir veri yapısına dönüştürülüp daha sonra bu veri yapısı üzerinde düğümlerin çocuklarının sıraları değiştirilerek yeni hiyerarşik ağaç oluşturulur. Daha sonra bu ağaç, optimizasyon işlemlerinden geçirilir ve seçilen alternatif çalışma planı mantıksal ifadeye geri dönüştürülür. Fiziksel ifadeye dönüşümünün yapılması ve neticesinin döndürülmesi için Timber XML Veritabanına gönderilir. Bu işlem kullanıcı tarafından belirlenen parametrik değerler ve algoritma dâhilinde yapılır.
- *Parser*: Bu sınıfın yapıcı metodudur. Kullanıcı tarafından girilen değerler bu metoda parametre olarak gönderilir ve bu parametreler sınıfın global değişkenlerine aktarılır.
- *replacePatternTreeContent*: Hiyerarşik yapıda bulunan ayrıştırılmış ağacı Timber veritabanının anlayacağı mantıksal ifadeye dönüştüren fonksiyondur.
- *WriteLog ve diğer metodlar*: Hata takip amaçlı metodlardır .

Node sınıfı

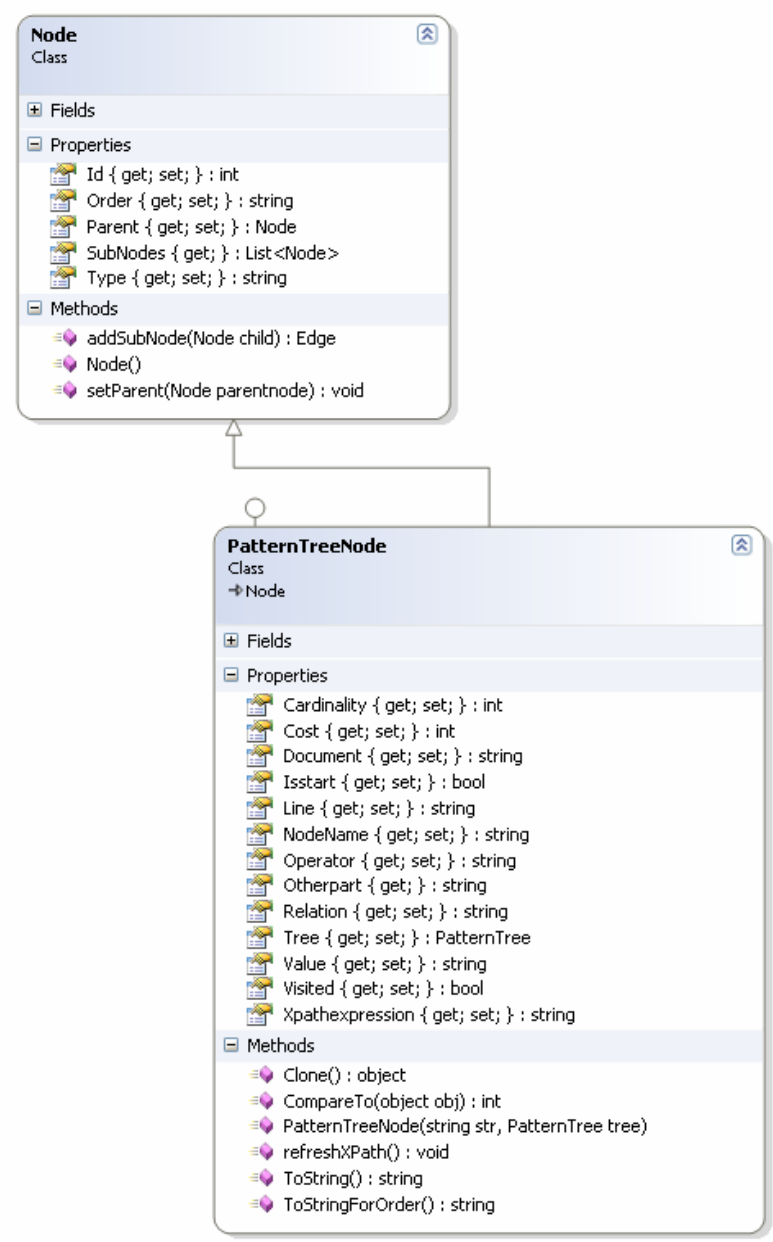
Şekil 4.7'de UML Class diyagramı gösterilen Node sınıfı, bir temel sınıf kütüphanesidir. PatternTreeNode sınıfının bazı temel özelliklerini özet olarak bulundurmaktadır.

Özellikleri aşağıdaki gibidir:

- *Id*: Her bir düğümün tekil tanımlayıcısı olarak kullanılmaktadır.
- *Order*: Her bir düğümün ana düğümü altındaki kaçıncı sırada olduğunu gösteren değer
- *Parent*: Her bir düğümün üst düğümüne referans
- *SubNodes*: Her bir düğümün alt düğümlerine referans
- *Type*: Düğümün Timber mantıksal planındaki türü ile ilgili bilgi olup, Document, element gibi değerler alabilmektedir.

Metodları aşağıdaki gibidir:

- *addSubNode*: Parametre olarak girilen düğümü alt çocuk olarak ekler
- *Node*: Yapıcı metod
- *setParent*: bu düğümün ata düğümünü belirlemek için kullanılır.



Şekil 4.7.Node ve PatternTreeNode sınıflarının UML diyagramları [19]

PatternTreeNode sınıfı

Şekil 4.7'de görülen PatternTreeNode sınıfı, mantıksal ifadede bulunan desen ağacının her bir düğümünü tutmak için kullanılır. Node sınıfından ortak özellikleri miras alır ve gerçekler.

Bu sınıf ICloneable arayüzünü gerçekler. Bu sayede PatternTreeNode'un yeni kopyaları daha pratik oluşturulabilir.

PatternTreeNode sınıfı aynı zamanda IComparable arayüzünü de gerçekler. Bu sayede farklı pattern tree'ler maliyet değerlerine göre kıyaslanabilir hale getirilebilirler ve azalan-artan sıralama işlemi kolaylaştırılmış olur.

Sınıfın özellikleri aşağıdaki gibidir:

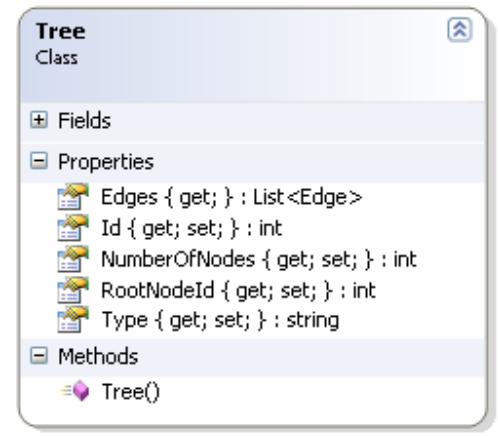
- *Cardinality*: XML dokümanında bu düğüm ile eşdeğer kaç düğüm olduğunu tutar.
- *Cost*: Bu düğümü elde etmek için gerekli maliyettir. Altındaki düğümlerin maliyeti ve kendi ile altındaki düğümlerin çaprazlama maliyetinin toplamıdır. Çaprazlama maliyeti hesaplanırken eşitlik(I)'de geçen ifade kullanılır.
- *Document*: XML dokümanına bir referans olup bir elemanın cardinality ve cost değerinin bulunmasını sağlamak maksatlıdır.
- *IsStart*: Bir düğümün kök düğüm olup olmadığını kök ise 1 değilse 0 tutaraf ifade eder.
- *Line*: Mantıksal plandaki düğümü ifade eden satırı içer.
- *NodeName*: Her bir düğüme verilen adı tutar.
- *Operator*: Sorguda geçen kıyaslayıcı aritmetik işlem varsa belirtir. (<,=, <>… gibi)
- *OtherPart*: İşlemi etkilemeyen ama düğümde geçen, mantıksal plandan gelen diğer bilgileri tutar.
- *Relation*: Üst düğüm ile olan ilişkisini belirler. Anne-Çocuk veya Ata-Soy ilişkisi.
- *Tree*: İçinde bulunduğu pattern tree'ye bir referanstır.
- *Value ve XPathExpression*: cardinality ve cost hesabında kullanılan geçici değişkenlerdir.

Metodları:

- *Clone*: Düğümün sıfırdan bir kopyasını oluşturur. IClonoble arayüzünü sağlamak için eklenmiştir.
- *CompareTo*: İki düğümün Cost değerlerini kıyaslamak ve ICompareble arayüzünü sağlamak için eklenmiştir.
- *PatternTreeNode*: yapıcı medoddur. Mantıksal ifade içerisinde yer alan bu düğüme ait satırı alır ve ayrıştırarak global değişkenlere aktarır.
- *refreshXPath*: XPath ifadesini bulup günceller. Cost ve cardinality değerleri hesabı için kullanılan dahili bir bir metoddur.
- *ToString ve ToStringForOrder*: hata takip amaçlı eklenmiş metodlardır.

Tree sınıfı

Düğümleri hiyerarşik yapıda tutmak için kullanılan sınıftır. UML sınıf diyagramı Şekil 4.8'de gösterildiği gibidir.



Şekil.4.8.Tree Sınıfı UML Class diyagramı [19]

Özellikleri aşağıdaki gibidir:

- *Edges*: Tüm düğümler arasında bulunan kenarları tutan bir dizindir.

- *Id*: Her bir ağacın tekil tanımlayıcısıdır.
- *NumberOfNodes*: Her bir ağaçta bulunan düğüm sayısıdır.
- *RootNodeId*: Her bir ağacın kök düğümünün tekil tanımlayıcısının değeri
- *Type*: Ağacın tipini belirler. Timber projesi mantıksal planında geçen ağaç türlerini belirtmek için kullanılır.

Bu sınıfın yapıcı dışında bir metodu bulunmamaktadır.

PatternTree sınıfı

Mantıksal ifadede bulunan PatternTree'yi ayrıştırma işleminde ve sonrasında tutmak için kullanılır. İçerisinde pattern tree düğümleri hiyerarşik sırada bulunur. Mantıksal ifadede bir pattern tree tüm sorguyu ifade etmek için kullanılır. Sorguda geçen ifadeler pattern tree'de bir düğüm olarak bulunur.

Özellikleri aşağıdaki gibidir:

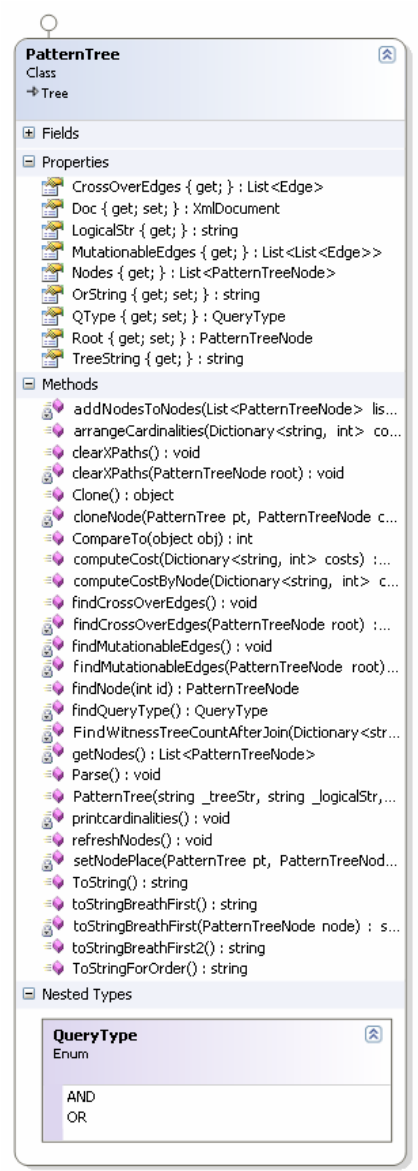
- *CrossOverEdge*: çaprazlanabilir kenarları tutan dizindir. Eğer bir düğümün birden fazla çocuğu varsa bu düğümler çaprazlanabilir düğümlerdir ve çocuk düğümler kendi aralarında çaprazlanabilirler. Bu dizin, çocuklarla ana düğüm arasındaki kenarları tutar.
- *Doc*: XML dokümanına referanstır. Maliyet hesabı gibi kestirimsel değerler elde edilirken kullanılır.
- *LogicalStr*: Timber mantıksal ifadesinin tamamını tutar.
- *MutationableEdges*: Bu ağaçta mutasyona uğrayabilecek kenarlar topluluğunu tutan bir dizidir. Bir dizin elemanı iki tane kenar sınıfı tutar. Eğer bireyde mutasyon yapılacaksa, bu iki kenar yer değiştirilir.
- *Nodes*: Hiyerarşik düzende barındırdığı düğümlere sırasal hızlı erişim için tutulduğu bir dizidir.
- *OrString*: Eğer sorgu OR ifadesi içeriyorsa, OR'lanan ifalerin iki tarafını tutan metindir.

- *QType*: Sorgunun AND veya OR'lardan oluştuğunu tutan bir özelliktir.
- *Root*: Kök düğümüne referanstır.
- *TreeString*: Mantıksal ifadede bu ağacın ifade edildiği kısmın metnini tutar.

Metodları aşağıdaki gibidir:

- *AddNodesToNodes*: Nodes dizinine bir düğümü ve rekürsif olarak alt düğümleriyle beraber ekler.
- *arrangeCardinalities***:** Ağaç düğümlerinin *Cardinalities* değerlerini hesaplar ve bu değerleri düğümlere atar.
- *clearXPaths*: Düğümlerde Cardinality ve cost değerlerini hesaplamak için kullanılan xpath değerlerini sıfırlar.
- *Clone*: Aynı ağaçtan yeni bir ağaç üretmek için kullanılır.
- *cloneNode*: Bir düğümü kopyalamak için kullanılır. Clone metodu tarafından kullanılır.
- *CompareTo*: 2 farklı ağacı karşılaştırır.
- *computeCost*: Ağacın maliyetini hesaplar. Yani sorgunun yapılma maliyetidir.
- *computeCostByNode*: Rekürsif olarak her düğümün maliyetini hesaplar. Düğüm maliyeti alt düğümlerin maliyetlerinin toplamıyla alt düğümlerle ata düğümü çaprazlama maliyetinin toplamıdır. Kök düğümünün maliyeti toplam ağacın maliyetidir.
- *findCrossOverEdges*: Ağaçta çaprazlanabilinir kenarları bulan metottur. Bulduğu kenarları CrossOverEdge dizininine ekler.
- *findMotationableEdges*: Ağaçta mutasyona uğrayabilir kenarları bulan metoddur. Bulunan kenarları MutationableEdges dizisine ekler.
- *findNode*: Ağaçta bir düğümü tekil belirleyiciden bulmaya yarayan metoddur.
- *findQueryType*: Sorgunun and'li veya or ile yazıldığını bulmaya yarar.
- *FindWitnessTreeCountAfterJoin*: İki düğümün çaprazlaması sonucunda elde edilen XML elemanının xml dokümanında kaç tane geçtiğini bulur.
- *getNodes*: Tüm düğümleri dizin olarak elde etmeye yarar.

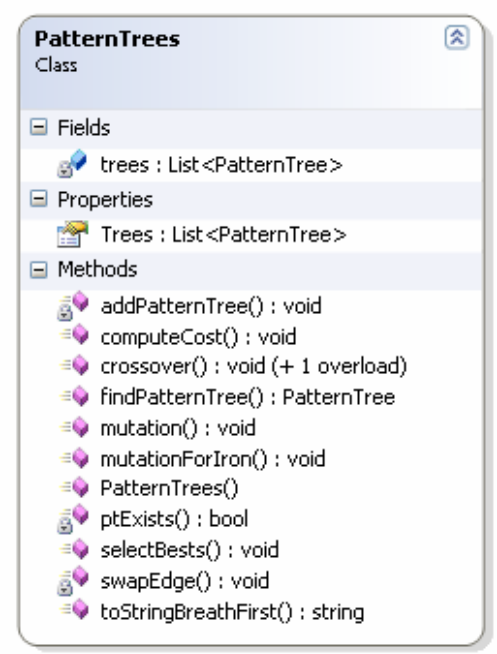
- *Parse*: Logical algebra da ağacı ifade eden metni ayrıştırır ve ayrıştırtırılan değerleri değişkenlere aktarır.
- *PatternTree*: Yapıcı metodtur. İçerisine mantıksal ifadeyi parametre olarak alır.
- *PrintCardinalities*: Hata ayıklama için oluşturulan metoddur
- *refreshNodes*: nodes dizinini tazelemek için kullanılır.
- *setNodePlace*: Düğümlerin ata ve çocuk düğümlerini atayan metoddur.



Şekil 4.9. PatternTree sınıfı UML Class diyagramı [19]

PatternTrees sınıfı

Mantıksal ifadede bulunan tüm pattern ağaçlarını tutan sınıftır. İçerisinde genel olarak sorgunun şekline göre 1 veya daha fazla pattern tree içerir. PatternTrees sınıfı UML Class diyagramı Şekil 4.10'da gösterilmiştir.



Şekil 4.10.Genetik sorgu optimizasyon uygulaması UML Class diyagramı [19]

Özellikleri aşağıdaki gibidir:

- *Trees*: Ağaçların tutulduğu dizidiri.

Metodları aşağıdaki gibidir:

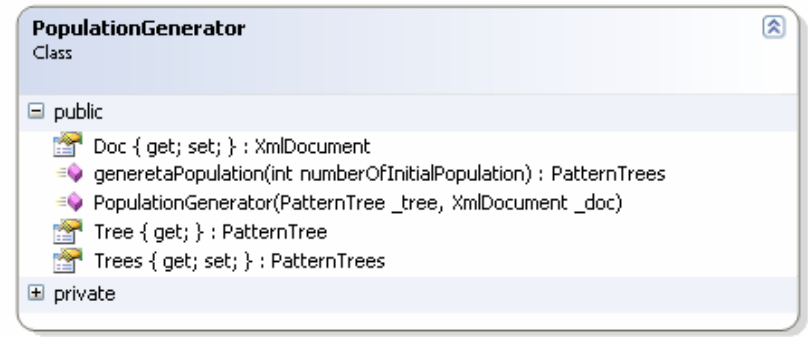
- *addParentTree*: Trees dizinine yeni bir pattern tree ekler.
- *computeCost*: Popülasyondaki tüm pattern ağaçlarının maliyetlerini hesaplar
- *crossover*: Popülasyon içerisinde bireyler arasında çaprazlama yapar. Çaprazlama iki tane rastgele seçilen bireyin rasgele tek noktadan bölünüp elde edilen parçaların birleştirilmesi ile yeni iki pattern tree oluşturulur.

- *findPatternTree*: tekil tanımlayıcı alıp bu tanımlayıcının ifade ettiği ağacı bulur. Yoksa null döndürür.
- *mutation*: bir birey içerisinde mutasyon oluşumunu gerçekleştirir.
- *mutationForIron*: Tavlama benzetimi için farklı yeni bireyler türetme amaçlı kullanılır. Bir çözümün komşu çözümlerini bulmak üzere tasarlanmış bir metoddur.
- *PatternTrees*: Yapıcı metod
- *ptExists*: desenin zaten var olup olmadığını bulur.
- *selectBests*: Popülasyon içindeki en iyi n bireyi seçer. Elitizim kriterini sağlamak için kullanılır.
- *swapEdges*: Çaprazlamaya yardımcı bir metod olup iç kullanım amaçlıdır. İki bireyi çaprazlama işlemini gerçekleştirir.

PopulationGenerator sınıfı

PopulationGenerator sınıfı ilk populasyonu üretmek için kullanılır. Yaptığı iş bir pattern ağacından rastgele istenilen sayıda ağaç üretmektir. PopulationGenerator sınıfına ait UML Class diyagramı Şekil 4.11'de gösterilmiştir. Metod ve özelliklerinin işlevleri sırayla aşağıda verilmiştir.

- *Tree*: Esas pattern ağacıdır.
- *Trees*: Üretilen pattern ağaçlarıdır.
- *Doc*: Xml dukümanına referanstır.
- *genereta**Population***: metodu parametre olarak belirtilen sayı kadar rastgele birey üretir.
- *PopulationGenerator***:** Yapıcı metottur. Üzerinde rasgele değişiklik yapılacak pattern tree'yi parametre olarak alır.



Şekil 4.11.PopulationGenerator sınıfı [19]

4.2.Tavlama Benzetimi ile Sorgu Optimizasyonu

Tavlama benzetimi de genetik algoritma gibi ilişkisel veritabanı sorgularında sıralama tayininde kullanılmış bir yöntemdir [14, 17, 18]. Tavlama benzetimi, genetik algoritmaya göre biraz daha karmaşıklığı az ve uygulaması basit bir yöntemdir. Ancak genetik algoritma, tavlama benzetimine göre daha iyi sonuçları yakalayabilmektedir.

4.2.1 Tavlama Benzetimi Algoritması'na genel bakış

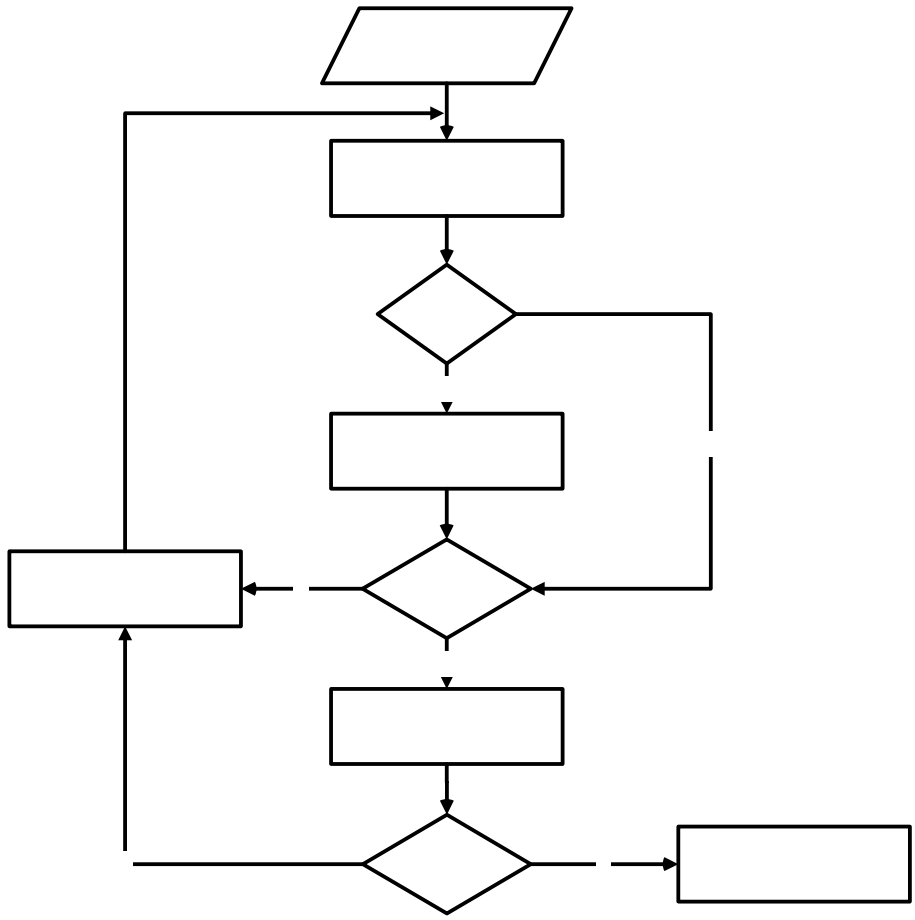
1983 yılında Kirkpatrick ve arkadaşları tarafından önerilen Tavlama Benzetimi, iyileştirme problemleri için iyi çözümler veren bir tarama tekniği olarak kabul görmüştür [46].

Tavlama Benzetimi, katıların ısıtılması ve sonra kristalleşmeye kadar yavaş yavaş soğutulması esasına dayalı bir yaklaşımla çözüm alanını tarar. Bu benzetime göre, sıcaklık değeri, elde edilen en iyi çözümden daha kötü çözümlerin kabul edilme olasılığını belirlemede kullanılır. Yüksek bir sıcaklık değeri ile başlatılır ve her bir adımda sıcaklık değeri düşürülmeden önce belli sayıda çözüm üretilir. Yeni çözümler belirlenen kriterlere göre kabul edilir veya reddedilir. Düşen her sıcaklık, eldeki çözümün bırakılıp yeni bir çözüme geçme ihtimalinin azalmasına etki eder. Sıcaklık minimum değere ulaştığında veya algoritma istenen iterasyon kadar çalıştığında

yahutta istenen çözüme ulaşıldığında algoritma sonlandırılır. Algoritma için sözde aşağıda yer almaktadır.

```
Begin
Başlangıç çözümü seç
Başlangıç sıcaklığı seç (t=100)
Sıcaklık azaltma fonksiyonu belirle
repeat
 repeat
   Yeni bir komşu çözüm üret
   if (yeni - eski) < 0 then yeni çözümü seç
   else begin
     [0,1] aralığında rassal sayı üret (r)
     if exp(-(yeni-eski)/t) > r then yeni çözümü seç
   end
  until iterasyon sayısına kadar
  t = f(t);
until (t<0) veya uygun çözüm bulununcaya kadar
End
```

Veri erişimi iyileştirmesinde, bazen sorgu optimizasyonu, sorgu çalıştırma süresinden daha fazla süre gerektirebilir. Bu istenmyen bir durumdur. Demir tavlama benzetimi, çözüm sahasını tarama işlemini belli bir iterasyon ile kısıtladığından, sorgu optimizasyonu için sadece belli bir süre çalışıp elde ettiğinin en iyisini döndürmesi yönü ile sorgu optimizasyonuna uygulanabilirliği yüksek bir algoritmadır.



Şekil 4.12. Tavlama Benzetimi Algoritması akış şeması [19]

4.2.2 Geliştirilen uygulama

Geliştirilen uygulama genetik ile bütünleşik olarak gerçeklenmiştir. Böylece, Düğüm ve ağaç yapılarının tekrar kullanımı amaçlanmıştır.

Genetik algoritmanın başlangıç sınıfı olan Parser tavlama benzetimi kullanıldığında da ilk olarak devreye girmekte ve uygun olan algoritma seçeneğini ilgili alt bileşenlere aktarmakta.

Her iki algoritma için başlangıç çözümü rastsal olarak üretilir. Tavlama benzetiminde komşu çözümleri tarama işlevi için Şekil 4.9'da gösterilen PatternTree sınıfının mutationForIron metodu kodlanmıştır.

5. DENEYSEL SONUÇLAR

Bu bölümde, incelenen ve gerçekleştirilen sorgu iyileştirme yöntemlerinin deneysel sonuçlarına yer verilmektedir. Bütün deneyler, bir çift çekirdekli AMD Turion64 1.6GHz mobile işlemci, 896GB RAM ve 100GB'lık 5400rpm SATA mobil disk'den oluşan sistem üstünde gerçekleştirilmiştir. Bütün deneylerde sorgu çalıştırma ortamı olarak Timber kullanılmıştır [10].

5.1.Veritabanı ve Sorgular

Deneylerde, verileri test etmek üzere oluşturulan deneme.xml verileri kullanılmıştır. Dosyanın veri boyutu 288KB'dır. İçerisinde 6400 adet XML düğümü içermektedir.

Sorgulama yönteminin etkinliğini test etmek için farklı karmaşıklık seviyesinde sorgular kullanılmıştır. Sorguların zorlukları ve kolaylıkları, mantıksal seviyede Timber tarafından elde edilen sorgu işleme ara desen (pattern tree)'lerin aşağıdaki karakteristiklerine göre belirlenmiştir:

- Her bir yükleme ait eşdeğer düğüm sayısı
- Ara desen ağacı derinliği
- Her bir düğüm altında, yer alan alt düğüm sayısı

Deneylerde kullanılan sorgular aşağıda verilmiştir. Bu sorgulardan Şekil 5.1'de yer alan Q1 bir basit seviye XQuery sorgusu olarak seçilmiştir. Şekil 5.2'de yer alan Q2 orta zorlukta bir sorgu ve Şekil 5.3'de yer alan Q3 sorgusu da ileri seviyede zor bir sorgu olarak seçilmiştir.

Q1 sorgusu aşağıda verildiği gibi seçilmiştir:

```
for $b in document("deneme.xml")/countries/country
      for $c in $b//location
      for $d in $c//employee
```

```
      for $e in $c//job
where
$b//COUNTRY_ID = UK and
$d/JOB_ID = SA_REP and
$d/MANAGER_ID = 146
         return
         <a>{$d}</a>
```

Q2 sorgusu aşağıda verildiği gibi seçilmiştir:

```
for $b in document("deneme.xml")/countries/country
        for $c in $b//location
        for $d in $c//employee
where
$b//COUNTRY_ID = UK and
$d/JOB_ID = SA_REP and
$d/DEPARTMENT_ID = 80 and
$d/SALARY < 9000 and
$d/SALARY > 6000 and
         return
         <a>{$d}</a>
```

Q3 sorgusu aşağıda verildiği gibi seçilmiştir:

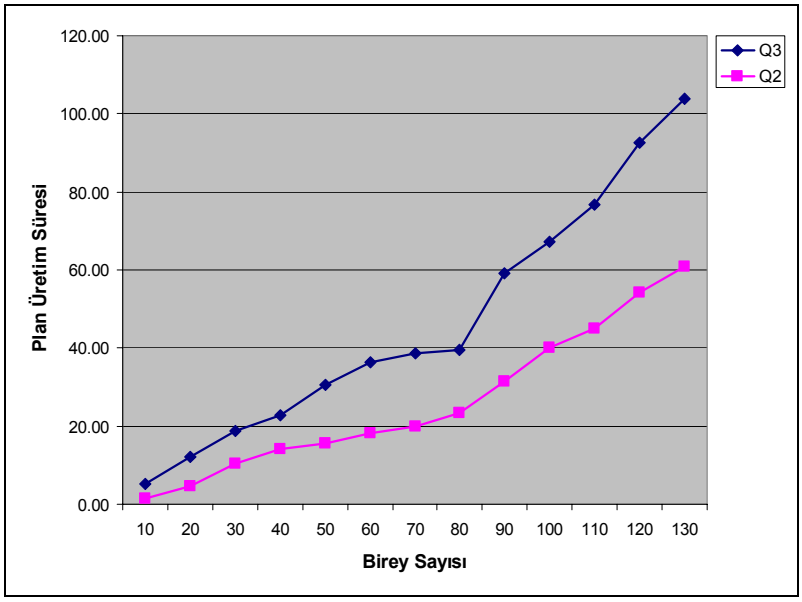
```
for $b in document("deneme.xml")/countries/country
        for $c in $b//location
        for $d in $c//employee
       for $e in $c//job
where
$b//COUNTRY_ID = UK and
$d/JOB_ID = SA_REP and
$d/MANAGER_ID = 146 and
$d/DEPARTMENT_ID = 80 and
$d/SALARY < 9000 and
$d/SALARY > 6000 and
$c/LOCATION_ID > 1000 and
```

```
$c/LOCATION_ID < 4000 and
$c/COUNTRY_ID = UK and
$e/MIN_SALARY>6000
        return  <a>{$e}</a>
```

5.2.Sorgu Planı Oluşturulma Süreleri

Yukarıda yer alan sorgular, Timber üstünde Genetik Algoritmanın farklı parametreleri ile ve Tavlama Benzetimi ile birleştirme sıralaması tayini işleminde kullanılmış ve optimizasyon için harcanan zamanlar aşağıda verilmiştir:

Genetik algoritma ile yapılan deneylerde, elitizm kriteri 3 olarak alınmıştır. Her bir sorgu planı 10 defa çalıştırılmış ve elde edilen sürelerin ortalamaları alınmıştır. Q1 sorgusunda, birey çeşitliliğinin yeterli olmaması nedeniyle genetik algoritma kullanılarak birleştirme sıralaması optimizasyonu yapılamamıştır. Q2 ve Q3 sorguları için yapılan deneylerde, daha iyi sonuçlar elde etmek için 10-130 arasında bireyden oluşan farklı popülasyonlar kullanılmıştır. Algoritmanın çalışma zamanı, sorgunun ayrıştırılması gibi diğer etkenler göz ardı edilerek Şekil 5.1'de görüldüğü gibi bulunmuştur.

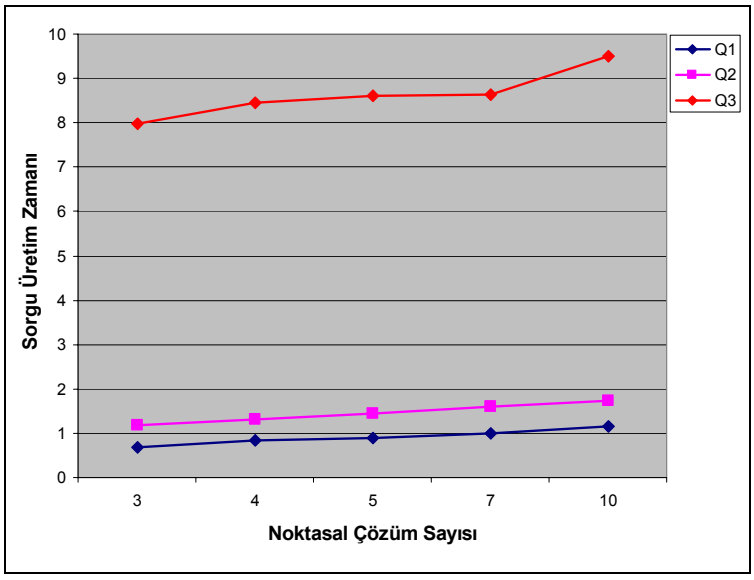


Şekil 5.1. Q2 ve Q3 sorguların genetik algoritma ile oluşturulma süreleri [19]

Şekil 5.1'e göre aradesen ağacı karmaşıklaştıkça, plan üretim zamanında da artım gözlemlenmektedir. Sorgu iyileştirmede, genetik algoritmanın çalışma zamanına doğrudan etki eden faktörlerden birey sayısının değişimi literatürde sıkça izlenen bir durumdur. Deneysel sonuçlara göre, birey sayısının artımı birleştirme plan üretim zamanını artıran bir faktör olarak göze çarpmaktadır.

Tavlama benzetimi kullanılarak birleştirme sıralaması tayinini için yapılan çalışmaların sonuçları Şekil 5.2'de gösterilmiştir. Q1, Q2 ve Q3 sorguları için birleştirme sıralaması için geçen sürenin, tavlama benzetiminin bir noktadaki artan sayıda çözüm üretilmesine bağlı incelenmiştir.

Tavlama benzetiminde, sıcaklık 100 dereceden başlatılarak her iterasyonda 1 derece düşürülmek sureti ile lineer olarak her noktada sabit sayıda çözüm denenmiştir. Farklı deneylerde bir noktadaki denenen çözüm sayısı artırılarak yapılan inceleme Şekil 5.2'de gösterilmiştir.



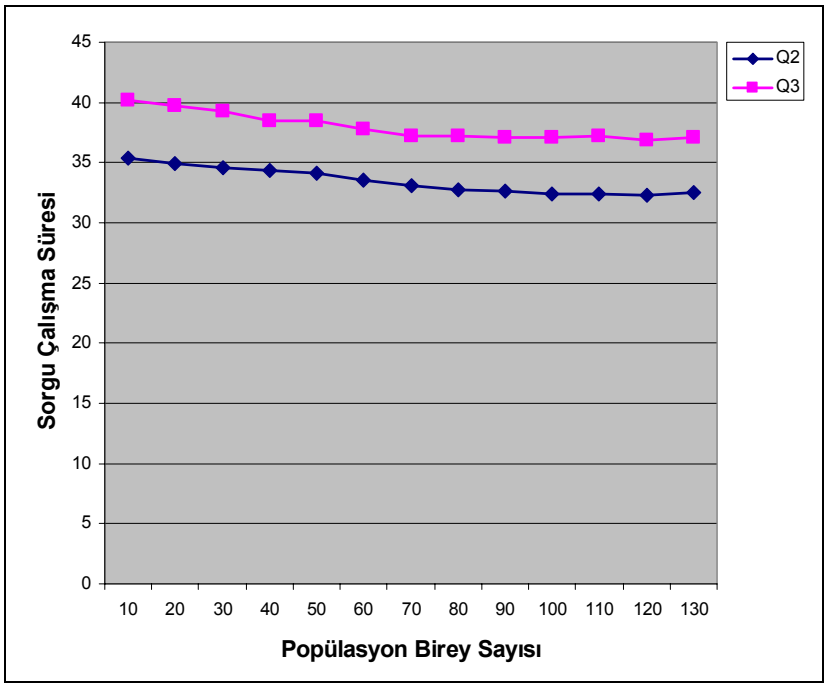
Şekil 5.2. Tavlama benzetimi ile Q1, Q2 ve Q3 sorgularının oluşturulma süreleri [19]

Tavlama benzetimi ile yapılan deneylerin sonucuna göre sorgu karmaşıklığı arttıkça çözüm için harcanan zaman da artmaktadır. Basit seviye bir sorgu için gerekli üretim

zamanına nazaran sorgu karmaşıklaştıkça çok daha fazla üretim zamanı gerekmektedir. Bu fark, birim çözümlerin üretim maliyeti ile doğrudan ilişkilidir.

5.3.Sorgu Çalıştırma Süreleri

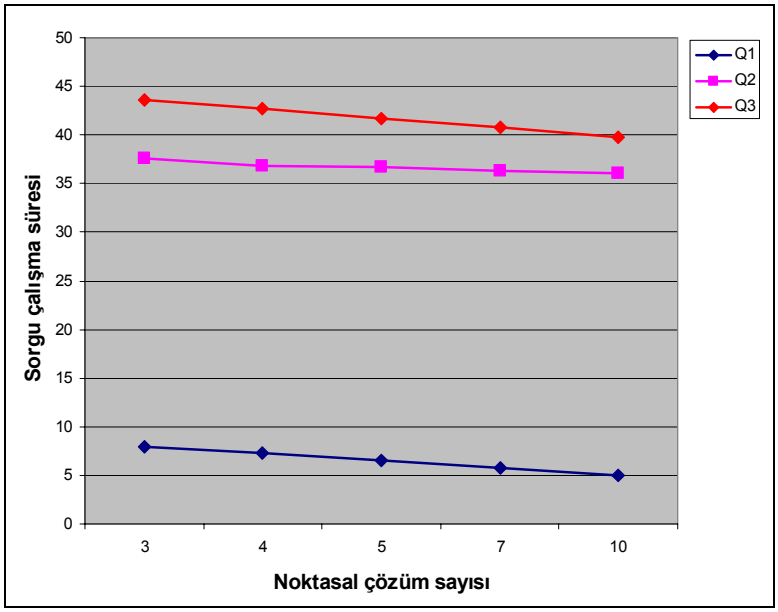
Artan sayıda popülasyon için sorgu süresindeki azalma 130 popülasyondan sonra azalmamaya başlamıştır. Öte yandan, 130 popülasyonluk denemelere kadar, popülasyon sayısı artırıldıkça başarılı sonuçlar alınmıştır. Artan popülasyon sayılarında mutasyon oranının en iyi sonuçları kaybetmeye neden olmasından dolayı, 0.001 olarak alınmıştır. Genetik algoritma ile yapılan iyileştirmelerden sonra Q2 ve Q3 sorgularının çalışma süreleri Şekil 5.3'de gösterilmiştir. Q1 sorgusu yeterli birey oluşumuna uygun olmadığı için bu deneylerde kullanılamamıştır.



Şekil 5.3. Q2 ve Q3 için Genetik Algoritma'da değişen birey sayısına karşılık sorgu süresi [19]

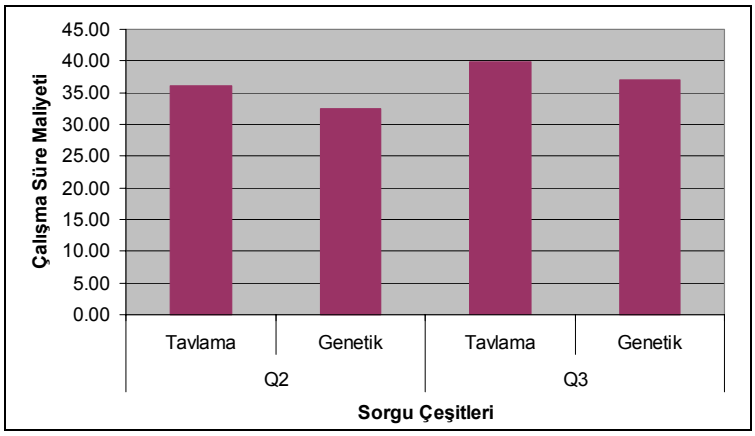
Tavlama Benzetimi Algoritması ile yapılan eşdeğer deneylerin sonuçları Şekil 5.4'de gösterilmiştir. Bu sonuçlara göre, Tavlama Benzetimi daha az çalışma süresi ve Genetik Algoritma'ya nazaran basit sorgulara da uygulanabilirliği ile ön plana çıksa

da karmaşık sorgularda, sonuçta meydana gelen iyileşme miktarı Genetik Algoritmaya nazaran daha azdır.



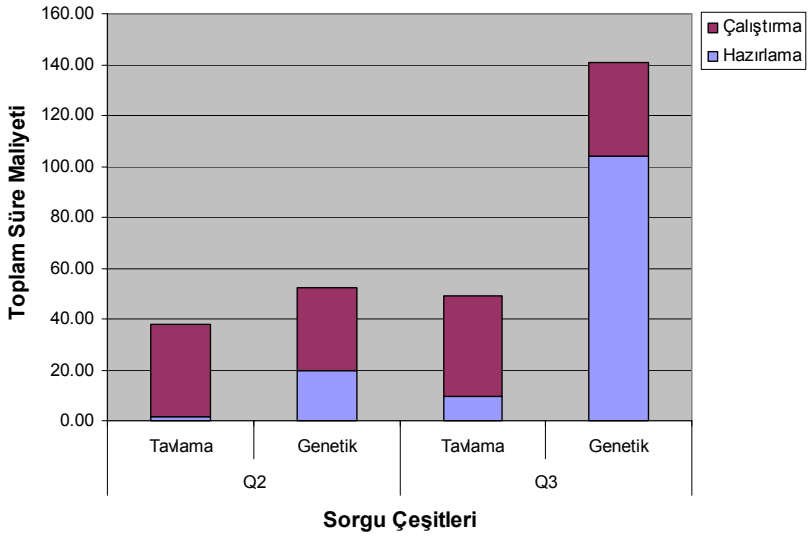
Şekil 5.4.Q1, Q2 ve Q3 için Tavlama Benzetimi ile her iterasyondaki değişen birey sayısına karşılık sorgu süresi [19]

Q2 ve Q3 sorgularının, bir noktada 10 farklı çözüm deneyerek uygulanan Tavlama Benzetimi ve Genetik Algoritma'nın 130 bireyli 0.001 mutasyon oranı ile kullanımlarının sonuçta meydana getirdiği iyileşmeler Şekil 5.5'de gösterilmiştir.



Şekil 5.5. Farklı zorluktaki sorguların GA ve TB ile çalışma süresi maliyetlerinin kıyaslanması [19]

Aynı deneylere ait çalışma sürelerini de hesaba katan toplam maliyet grafiği Şekil 5.6'da görüldüğü gibidir. Buna göre, genetik algoritma daha fazla hazırlama süresi gerektirse de elde edilen çalıştırma planı daha kısa sürede çalıştırılabilmektedir.



Şekil 5.6. Farklı zorluktaki sorguların GA ve TB ile toplam maliyet kıyaslaması [19]

6. SONUÇ VE ÖNERİLER

Bu çalışmada, genetik algoritma ve tavlama benzetimi algoritması İVTYS ortamında olduğu gibi XVTYS ortamında da bir birleştirme sıralaması tayin yöntemi olarak başarıyla kullanılmıştır. Deneysel çalışmalarda basit seviye XML sorgularda genetik algoritma ve tavlama benzetimi algoritmalarının uygulanabilirliğinin azaldığı, buna karşın özellikle karmaşık sorgularda genetik algoritma ve tavlama benzetiminin, daha iyi çalıştırma planları elde edebilmek için güçlü birer seçenek olarak değerlendirilebilecek performansa sahip olduğu görülmüştür.

Özellikle mantıksal sorgu ağacının karmaşık olduğu durumlarda genetik algoritma, tavlama benzetimine göre daha iyi çalışma süresi ile sonuçlar vermektedir. Artan popülasyonlarda genetik algoritma ile yapılan sorgu iyileştirme süresinin bazen çalıştırma süresini aşabildiği görülmüştür. Bunun en büyük nedeni, artan birey sayısı ile genetik algoritmanın daha fazla çözüm uzayını tarama eğilimine girmesidir. Bu türden çözümler tek seferde yorumlanıp sonuçlandırılan sorgular için bir dezavantaj iken, İVTYS ortamında hali hazırda kullanılan ve XVTYS'lerde de kullanılması muhtemel saklı yordam ve tetikleyici gibi bir kere çalışma planı hazırlanıp, bu plan ile çok kere çalıştırılan türden sorgular için daha iyi bir yaklaşım olarak değerlendirilebilir. Uygun parametrelerle çalıştırıldığında genetik algoritma bir birleşim sıralaması seçme yöntemi olarak başarıyla kullanılmıştır.

KAYNAKLAR

1. Internet: World Wide Web Consortium, “Extensible Markup Language (XML)”, http://www.w3.org/TR/REC-xml/ (2004).

2. Internet: World Wide Web Consortium, “Standardized General Markup Language (SGML)”, http://www.w3.org/MarkUp/SGML/ (1996).

3. Pal, S., Cseri, I., Seeliger, O., Schaller, G., Giakoumakis, L., Zolotov, V., “Indexing XML Data Stored in a Relational Database “, ***VLDB Conference*** 1147-1157(2004).

4. Florescu, D., Kossmann, D., “Storing and Querying XML Data using an RDMBS”, ***IEEE Data Engineering Bulletin,*** 22(1): 27-34(1999).

5. Internet: World Wide Web Consortium, “XPath”, http://www.w3.org/TR/xpath (1999).

6. Internet: World Wide Web Consortium, “XML Query Language (XQL)”, http://www.w3.org/TandS/QL/QL98/pp/xql.html (1998).

7. Deutsch, A., Fernandez, M.F., Florescu, D., Levy, A.Y., Maier, D., Suciu, D., “Querying XML Data”, ***IEEE Data Engineering Bulletin***, 22(3):10-18 (1999).

8. Ceri, S., Comai, S., Damiani, E., Fraternali P, “A Graphical Language for Querying and Restructuring XML Documents (XML-GL)”, ***Computer Networks: The International Journal of Computer and Telecommunications Networking***, 31(11-16):1171-1187 (1999).

9. Internet: World Wide Web Consortium, "XQuery (Proposed Recommendation)”, http://www.w3.org/TR/xquery/ (2006).

10. Paparizos, S., Al-Khalifa, S., Chapman, A., Jagadish, H. V., Lakshmanan, V. S., Nierman A., Patel, J. M., Srivastava, D., Wiwatwattana, N., Wu, Y., Yu, C.: “TIMBER: A Native System for Querying XML”, ***SIGMOD Conference*****,** California, 672-672(2003).

11. Vyas, A., Fernández, M. F., Siméon, J., “The Simplest XML Storage Manager Ever”, ***Very Large Data Bases 32nd international conference*****,** Seoul-Korea, 1215 – 1218 (2006).

12. Internet:Wikipedia, “BerkeleyDB”, http://en.wikipedia.org/wiki/Berkeley_db (2006)

13. Codd, E.F., "A relational model of data for large shared data banks", ***ACM***, 13(6):377-378 (1970).

14. Chaudhuri, S. "An Overview of Query Optimization in Relational Systems", ***Symposium on Principles of Database Systems***, Seattle-Washington-ABD, 34 - 43 (1998).

15. Wu, Y., Patel, J. Jagadish, H.V., "Structural Join Order Selection for XML Query Optimization", ***ICDE Conference***, Bangalore-Hindistan, 443-454 (2003)

16. Fernandez, M., Simeon, J., Suciu, C., Wadler, P., "A Data Model and Algebra for XML Query", **ACM SIGMOD Record**, 29(2):141-152 (2000).

17. Bennett, K., Ferris, M.C., Ioannidis, Y.E., "A Genetic Algorithm for Database Query Optimization", ***4th International Conference on Genetic Algorithms***, San Mateo-Kanada, 400-407 (1991).

18. "Heuristic and randomized optimization for the join ordering problem", ***The VLDB Journal The International Journal on Very Large Data Bases***, 6(3): 191-208 (1997).

19. Gözüdeli, Y., arşivinden, (2007).

20. D.Florescu ve M.Kossmann, "Storing and Querying XML data using RDBMS", ***IEEE Data Engineering Bulettin***, ABD, 22(3):27-34 (1999).

21. M.Fernandez, W.C.Tan ve D.Sucio, "SilkRoute: Trading between relational and XML", ***ACM Transactions on Database Systems***, New York-ABD, 27(4):438-493 (2002).

22. Shanmugasundaram, J., Kiernan, J., Shekita, E., Fan, Funderburk, C.J., "Querying XML Views of Relational Data", ***26th Very Large Data Bases Conference***, Kahire-Mısır, 65-76 (2000).

23. Gözüdeli, Y., "Yazılımcılar için SQL Server 2005 ve Veritabanı Programlama", ***Seçkin Yayınları***, Ankara, 392-292 (2006).

24. Meng, X., Wang, Y., Luo, D., Lu, S., An, J., Chen, Y., Ou, J., Jiangi, Y., "OrientX : A Schemabased Native XML Database System", ***Proceedings of the VLDB Conference***, Berlin-Almanya , 1057–1060 (2003).

25. Schoning, H., "Tamino - A DBMS Designed for XML", ***17th ICDE Conference*****,** Heidelberg-Almanya, 149-154 (2001).

26. Internet: World Wide Web Consortium, "XQueryX", http://www.w3.org/TR/xqueryx/ (2006).

27. Internet: World Wide Web Consortium, "XQuery Requirements" http://www.w3.org/TR/xquery-requirements/ (2005).

28. Internet: World Wide Web Consortium, "XQuery 1.0 and XPath 2.0 Functions and Operators", http://www.w3.org/TR/xpath-functions/ (2006).

29. Internet: World Wide Web Consortium, "XML Schema 2.0", http://www.w3.org/XML/Schema (2004).

30. Chaudhuri, S., , "An Overview of Query Optimization in Relational Systems", ***Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems***, Seattle-Washington, 34-43 (1998).

31. Ramakrishnan,R, Gehrke, J, "Database Management Systems", 3th Edition, ***McGraw-HILL***, New York, 100-126 (2003)

32. Astarahan, M.M., "System-R: A relational approach to data management", ***ACM Transactions on Database Systems***, 1(2):97-137 (1976).

33. Ono, K., Lohman, G.M., "Measuring the complexity of join enumaration in query optimization", ***VLDB Conference***, Brisbane-Avustralya, 314-325 (1990).

34. Vance, B., Maier, D., "Rapid Bush Join-order Optimization with Cartesian Products", ***SIGMOD Konferansı Veri Yönetimi***,35-46 (1996).

35. Sellinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G., "Access path selection in a relational database management system", ***ACM-SIGMOD Konferansı Veri Yönetimi***, Boston-ABD, 23-34 (1979).

36. E. Wong, K. Youssefi, "A strategy for query proccessing", ***ACM Transactions on Database Systems***, 1(3):223-241 (1976).

37. Krisnamurth, R., Boral, H., Zaniolo, C., "Optimization of non-recursive queries", ***VLDB Konferansı***, Japonya, 128-137 (1986).

38. Abaraki, T., Kameda, T., "Optimal nesting for computing N relational joins", ***ACM Transactions on Relational Systems***, 9(3):482-502 (1984).

39. Y.E. Ionnidis ve E.Wong, "Query optimization by simulated anneling", ***In Proc. of the ACM SIGMOD Conf. on Management of Data***, San Fransisco-ABD, 9-22 (1987).

40. Goldberg, D.E., "Genetic Algorithms in Search Optimization and Machine Learning", ***Addison Wesley***, Alabama, 217-223 (1989).

41. Ioannidis, Y.E, Kang, Y.C, "Randomized algorithms for optimizing large join queries", ***In Proc. of the ACM SIGMOD Conf. on Management of Data***, Denver-ABD, 168-177 (1990).

42. Jagadish, H. V., L. Lakshmanan , Srivastava, V. S., D., Thompson, K., "TAX: A Tree Algebra for XML", ***The 8th International Workshop on Database Programming Languages***, Frascati, Italya,149-164 (2002)

43. Philippe, M., "XQuery optimization", ***VLDB Konferansı*****,** Berlin-Almanya, 12-13 (2003).

44. Al-Khalifa, S., Jagadish, H.V., Patel, J.M., Wu, Y., Koudas, N., Srivastava, D., "Structural Joins: A Primitive for Efficient XML Query Pattern Matching", ***ICDE Conf***, 141-152 (2002).

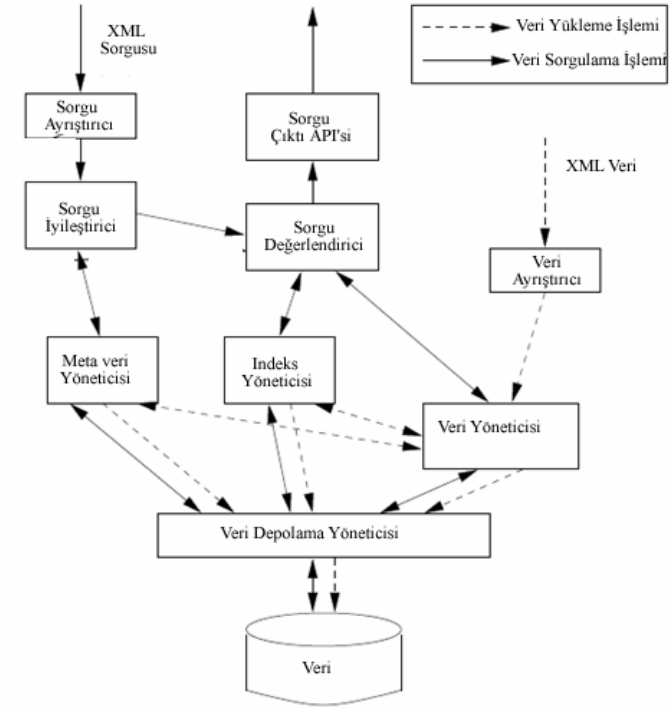
45. Wu, Y., Jagadish, H., "Structural Join Order Selection for XML Query Optimization", ***In Proc. ICDE Conf*****.,** Bangalore-Hindistan, 443-454 (2003).

46. Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P, "Optimization by Simulated Annealing", ***Science***, 220 (4598):671-680 (1983).

EKLER

EK-1 Timber veritabanı yönetim sistemi ile ilgili ek bilgiler

Timber İVTYS sistemine en yakın paralellikte bir XVTYS geliştirme amacıyla University of Michigan bünyesinde geliştirilmiş bir XVTYS'dir.



Şekil 1.1. Timber XVTYS işlevsel blok diyagramı [10]

İVTYS'deki ilişkisel cebire eşdeğer olarak, Timber da TAX adı verilen bir cebire dayanarak sorguları ele alır. Bu cebir sisteminde, XML veriler ilişkisel sistemdeki satır temel birimi yerine sıralı etiketli ağaç düğümleri olarak ele alınır. Böylelikle XML'in karmaşık ve değişken yapısı en temel seviye bir birime kavuşturulmuş olur.

Bir sorgu bir defa cebirsel gösterime ayrıştırıldıktan sonra, ifade bir sorgu iyileştiriciye gönderilir. Sorgu iyileştirici, mantıksal planı ele alarak fiziksel işlemleri kendisi en iyileştirilmiş olarak gerçekleştirir. Maliyete dayalı sorgu iyileştirme, en son veriler ve ara sonuçlar için iyi bir boyut kestirim yaklaşımı gerektirir. XML sorgunun işlenmesindeki en önemli noktalardan biri structural join'in yapılandırılmasıdır [10].

EK-2 Örneklerde kullanılan XML dokümanı

```
<?xml version="1.0" ?>
<kitaplar>
  <kitap numara="1">
    <isim>Visual Basic.NET</isim>
    <ISBN>0-672-32203-X</ISBN>
    <ozet>VB.NET konusunda  geçis bilgilerini içerien kitap.</ozet>
    <sayfaSayisi>204</sayfaSayisi>
  </kitap>
  <kitap numara="2">
    <isim>Telkin ve Hipnoz ile Ögrenme Teknikleri</isim>
    <ISBN>975-6700-15-7</ISBN>
    <ozet>ögrenme ve sinav konusunda hipnoz teknigi</ozet>
    <sayfaSayisi>274</sayfaSayisi>
  </kitap>
  <kitap numara="3">
    <isim>Yatirim Plani Yapmak</isim>
    <ISBN>975-381-263-9</ISBN>
    <ozet>Yatirim yapmak için nereden baslamali, neler yapmali</ozet>
    <sayfaSayisi>74</sayfaSayisi>
  </kitap>
  <kitap numara="4">
    <isim>Is Basinda Duygusal Zeka</isim>
    <ISBN>975-434-224-5</ISBN>
    <ozet>Duygusal zekanin is ve yasam konusundaki etkileri</ozet>
    <sayfaSayisi>447</sayfaSayisi>
  </kitap>
  <kitap numara="5">
    <isim>Is Hayatinda Motivasyon</isim>
    <ISBN>975-8243-98-5</ISBN>
    <ozet>Is basinda Motivasyon </ozet>
    <sayfaSayisi>140</sayfaSayisi>
  </kitap>
  <kitap numara="6">
    <isim>Hayat Yolunda zorluklarla Mücadele</isim>
    <ISBN>975-362-135-3</ISBN>
    <ozet>Kişisel motivasyon ile ilgili bir kitap </ozet>
    <sayfaSayisi>119</sayfaSayisi>
  </kitap>
</kitaplar>
```

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı	: GÖZÜDELİ, Yaşar
Uyruğu	: T.C.
Doğum tarihi ve yeri	: 01.01.1978 Kahramanmaraş
Medeni hali	: Evli
Telefon	: 0 (312) 583 35 44
Faks	: 0 (312) 583 36 36
e-mail	: yasar.gozudeli@tib.gov.tr

Eğitim

Derece	Eğitim Birimi	MezuniyetTarihi
Lisans	İstanbul Üniversitesi/ Bilg. Müh. Bölümü	2002
Lise	Anadolu Öğretmen Lisesi	1998

İş Deneyimi

Yıl	Yer	Görev
2001-2002	Estore A.Ş	Yazılım Uzmanı
2002-2006	Maliye Bakanlığı	Bilg. Mühendisi
2006	Telekomünikasyon İletişim Başk.	Uzman

Yabancı Dil

İngilizce

Yayınlar

1. Gözüdeli, Y., Akcayol, M.A., "Genetik Algoritma ile Web Sayfası Düzeninin Gerçek Zamanlı Optimizasyonu", Gazi Üniversitesi Mühendislik-Mimarlık Fakültesi Dergisi

2. Gözüdeli, Y. , "SQL Server ile Temel Veritabanı Programlama", Seçkin Yayınları-2004, Ankara

3. Gözüdeli, Y., "Yazılımcılar için SQL Server 2005 ve Veritabanı Programlama", Seçkin Yayınları–2006, Ankara