YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

HAREKET EDEN NESNELER İÇİN SORGU SİSTEMİ

Bilgisayar Y. Müh. Perihan KİLİMCİ

FBE Bilgisayar Mühendisliği Anabilim Dalında Bilgisayar Mühendisliği Programında Hazırlanan

DOKTORA TEZİ

Tez Savunma Tarihi	**:**	28 EKİM 2009
Tez Danışmanı	**:**	Prof. Dr. Oya KALIPSIZ
Jüri Üyeleri	**:**	Prof. Dr. Nadia ERDOĞAN
		Prof. Dr. Selim AKYOKUŞ
		Doç. Dr. Adem KARAHOCA
		Yrd. Doç. Dr. Yücel SAYGIN

İSTANBUL, 2009

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

$movObj_{id}$	Hareketli nesnenin belirleyici numarası
$<t_s, t_e>$	Hareketli nesnenin <x, y> konumunda kaldığı zaman aralığı
v	Hareketli nesnenin hızı
<x, y>	Konum koordinatları
N	Veri tabanındaki kayıt sayısı
DA	Bir sorguyu cevaplamak için disk G/Ç sayısı
GA	Grid hücre ortalama erişim sayısı
DA_g	Diskteki veri sayısı her erişilen grid için
IA_g	Diskteki endeks sayısı her erişilen grid için
f	B-tree endeksinin fanout'ı
BS	Blok büyüklüğü (bir bloktaki kayıt-tuple sayısı)
q	Uzay boyutunda sorgunun büyüklüğü
q_t	Zaman boyutunda sorgunun büyüklüğü
l	Her boyutta grid'in uzunluğu
l*	Her boyutta grid'in en iyi (optimum) uzunluğu
N_g	Grid'deki toplam hücre sayısı = $(1/l)^2$
N_g^*	Grid'deki toplam optimum hücre sayısı = $(1/l^*)^2$
N_p^*	Bir boyuttaki optimum partition sayısı

KISALTMA LİSTESİ

CASTER	Computer-Aided SpatioTemporal Entity Relationship
CASTR	Computer-Aided SpatioTemporal Relational
ER	Entity-Relationship
GIS	Geographic Information Systems
GPS	Global Positioning Systems
GSTD	Generate Spatio Temporal Data
GTERD	Generator of Time-Evolving Data
FTL	Future Temporal Logic
LRS	Linear Referencing System
MADS	Modeling of Application Data with Spatio-temporal features
MOD	Moving Objects Databases
MOR	Moving-Object Rectangles
MOST	Moving Objects Spatio-Temporal
MBR	Minimum Bounding Rectangle
ODCI	Oracle Data Cartridge Interface
RDBMS	Relational Database Management System
SDOAPI	Oracle Spatial Java Library
SDT	Spatial Data Types
SQL	Structured Query Language
STAM	Spatio-Temporal Access Method
STER	SpatioTemporal Entity-Relationship
STXER	SpatioTemporal eXtended Entity-Relational model
SUMO	Simulation of Urban MObilitiy

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖNSÖZ

Araştırmacılar, hareket eden nesneler konusunda henüz bir standart oluşturulamadığı için bu konuyu bir çok uygulamada farklı açılardan incelemişlerdir. Günümüzde internet ve kablosuz cihazların yaygınlaşması ile hareket eden nesnelerin, gerçek zamanlı olarak bu cihazlarda takibi pek çok uygulamada gerçekleştirilmektedir. Hareket eden nesneler, geleneksel veri tabanı uygulamalarından farklı kavramsal veri modelleme, veri tanımlama, endeksleme ve sorgulama özelliklerine ihtiyaç duymaktadır.

Tez çalışması kapsamında, öncelikle hareket eden nesnelerin kavramsal veri modellerini oluşturma için günümüzde kullanılan kavramsal veri modelleme çalışmaları incelenmiştir. Nesneye yönelik tasarım tekniklerinin kullanıldığı, Parent ve arkadaşları tarafından geliştirilen MADS (Modeling of Application Data with Spatio-temporal features) ile kavramsal veri modeli belirlenip, oluşturulmuştur.

Bir çok araştırmada oluşturulan kavramsal veri modeline göre hareket eden nesne uygulama verileri, veri tabanı yönetim sisteminden bağımsız olarak dosyalarda saklanıp performans ölçümleri gerçekleştirilmiştir. Buna rağmen sınırlı sayıdaki araştırmada, kavramsal veri modelleri veri tabanı yönetim sistemlerinde tanımlanmış ve endeksleme yöntemleri gerçekleştirilmiştir. Hareket eden nesneler, hem uzaya hem de zamana bağlı (spatio-temporal) özellikler içermektedir. Bu iki tip özelliği içeren hazır bir veri tabanı, günümüzde bulunmamaktadır. Buna karşın, sadece uzaya bağlı tanımlamaların eklendiği hazır veri tabanları kullanıcılara sunulmaktadır. Bu çalışmada, uzay eklentisi ile geleneksel veri tabanı olan Oracle Spatial 10g kullanılarak, tanımlanan veri modeli ismini MOORA (Moving Objects within ORAcle) olarak verip, tanımladığımız hareket eden nesneler sorgu sistemi için tanımlanmıştır. Daha sonra, MOORA sorgu sistemi dili ve yetenekleri belirlenmiş ve Oracle'da tanımlanmıştır. Hareket eden nesne verileri, Brinkhoff'un gerçekleştirdiği üreteç ile Eminönü-İstanbul sayısal haritası üzerinde oluşturulmuştur. Farklı endeks yapıları ile veriler veri tabanından okunarak performans ölçümleri elde edilmiştir. Gerçekleştirilen çalışmalarla, hareket eden nesneler için özgün bir sorgu sistemi oluşturmaya gayret edilmiştir.

Tüm bu çalışmalar sırasında, sürekli olarak beni yönlendiren, tezimi düzenlememde yol gösteren ve bugünkü bilgi birikimimi oluşturmama yardımcı olan tez danışmanım Prof. Dr. Oya KALIPSIZ'a sonsuz şükranlarımı sunarım.

Yeşilyurt — Perihan KİLİMCİ

19 Ağustos 2009

ÖZET

Üzerinde GPS cihazları olan çok sayıda hareket eden nesne için gerçekleştirilen uygulamalarda, uygulamadaki tercihe göre hareket eden nesne bilgileri, belirli aralıklarla veya bilgilerinde değişiklik olduğunda sunucuya bildirilmektedir. Bu verilerin çok büyük olmasından dolayı, etkin veri modelleme, veri saklama ve erişim yöntemlerine ihtiyaç duyulmaktadır.

Bu çalışmada hareket eden nesnelerin, nesneye yönelik tasarım tekniklerinin kullanıldığı, Parent ve arkadaşları tarafından geliştirilen MADS (Modeling of Application Data with Spatio-temporal features) ile kavramsal veri modeli oluşturulmuştur. Çalışma sunulurken, hareket eden nesneler için literatürdeki kuramsal endeksleme yapılarından bahsedilmiş ve hareket eden nesneler için sorgu dili özellikleri tanımlanmıştır.

Tanımlanan kavramsal veri modeli ve sorgu dilinin, veri tabanı yönetim sistemlerinde uygulanabilirliğini göstermek amacıyla, Oracle Veri kartuşu ile teklif edilen sorgu sistemi gerçekleştirilmiştir. Hareket eden nesne bilgilerine hızlı ulaşmak için Space Partitioning with Indexes on Time yöntemi, R-tree + zamana bağlı B-tree ve Z-değerleri + B-tree endeks yapıları oluşturularak performans ölçümleri elde edilmiştir. Oluşturulan bu sistemde, gezingesinin bir bölümünde hız yapmış hareketli nesnelerin sayısının az olmasından dolayı, hızlı hareket eden nesneler için ayrı bölümlenme algoritması ortaya konmuştur. Endeks yapısının hızlı nesnelerle bozulmamasından dolayı, performans artışları elde edilmiştir. Deneylerde hareket eden nesne bilgileri, Brinkhoff'un geliştirdiği veri üreteci ile Eminönü sayısal haritası üzerinde oluşturularak kullanılmıştır.

Anahtar Kelimeler: hareketli nesneler, hareketli nesne modelleme, MADS, hareketli nesne erişim yapıları, veri tabanı yönetim sisteminde hareketli nesneler, Oracle Spatial 10g.

QUERY SYSTEM FOR MOVING OBJECTS

ABSTRACT

Moving objects that equipped with GPS, send their data to the server when a threshold value exceeded or a change in their state, according to the application preferences. These huge amounts of data, necessitates using of efficient data modeling, data storage and access methods.

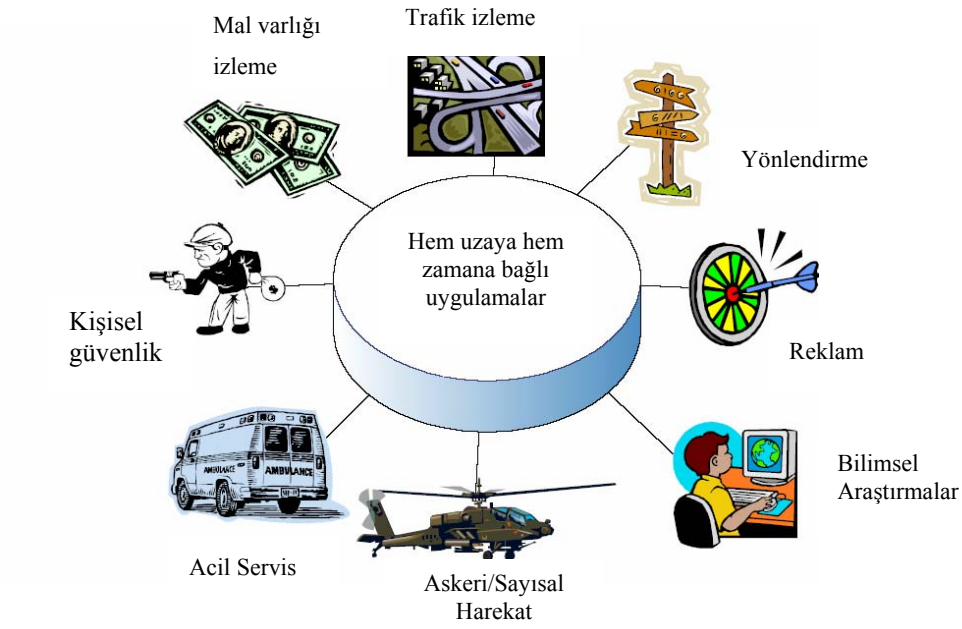
In this work, conceptual data model for moving objects is defined with Parent and et.al. work, MADS (Modeling of Application Data with Spatio-temporal features). This model uses object oriented design techniques. Theoretical indexing structures are investigated, while presenting this work. Then, query language specifications are defined for moving objects.

In order to prove that defined conceptual data model and query language are applicable in database management systems, the proposed query system is implemented in Oracle Data Cartridge. Space-Partitioning with Indexes on Time Approach, R-tree + temporal B-tree and Z-value + B-tree approaches are built in order to access moving objects data fast. The number of moving objects which has high velocity in some time is small in this system. Therefore, we proposed that new velocity-partitioning approach. Performance improvements are obtained while not affecting the index structure. Moving objects data are generated by Brinkhoff's Generator on Eminönü-İstanbul digital map.

Keywords: moving objects, modeling of moving objects, MADS, access structures of moving objects, moving objects in database management systems, Oracle Spatial 10g.

1. GİRİŞ

Son yıllarda hem uzaya hem de zamana bağlı verileri destekleyerek veri tabanını genişletmek için çok sayıda kitap yazılmış ve konferanslar düzenlenmiştir. Kablosuz teknolojilerin gelişmesi ve GPS (Global Positioning Systems) cihazları yardımı ile hem uzaya hem de zamana bağlı veriler üretilmiştir. Geliştirilen uygulamalara örnek olarak istenilen yere, sayısal harita üzerinde en kısa ve en az trafiği olan yol ile yönlendirme verilebilir. Bu gelişmelerle birlikte hem uzaya hem de zamana (spatio-temporal) bağlı veri tabanlarında, pek çok araştırma yapılmaktadır. Bu tip verinin yararları ve kullanımı sınırsızdır. Örneğin sağlık kuruluşları, personelini ve hastaları acil durumlarda izleyebilmektedir. Ticari kuruluşlar, mühendislerini gün içinde izleyerek en yakın servise yönlendirebilmektedir. Trafik izleme (Baars, M. 2004; Brakatsoulas, S.vd. 2004), mal varlığı izleme, suç izleme (Griffiths, T.vd. 2004) gibi pek çok uygulama gerçekleştirilebilmektedir (Şekil 1.1).



Şekil 1.1 Hem uzaya hem zamana bağlı uygulamalar

Her uygulamaya göre izlenen ve görüntülenen hareketli nesneler (moving objects) farklıdır. Hareketli nesneler uygulamalarda; insanlar, hayvanlar, arabalar, gönderilen ürünler, hava durumu sistemleri olabilmektedir. Bilimsel ve eğitim amaçlı uygulamalar da bu tip veri üzerine kurulabilmektedir.

Genel olarak gerçekleştirilen çalışmalar değerlendirildiğinde, hareket eden nesneler konusunda gerçekleştirilen araştırma konuları aşağıda verilmiştir:

- Bir ağ üzerinde veya bir alan içinde hareket eden nesnenin nerede olduğunun hesaplanması,
- Hareket eden nesnelerin ortak bir sunucu üzerinde saklanması,
- Hareket eden nesneleri hızlı olarak sorgulayan veri tabanının tasarlanması,
- Hareket eden nesneler için benchmark'ların gerçekleştirilmesi,
- Hareket eden nesneler veri tabanını etkin olarak depolayan donanım birimlerinin tasarlanması,
- Nesnenin t birim zaman içinde nerede olabileceğinin tahmini,
- Hareket eden nesneye en yakın diğer hareket eden nesne/nesnelerin hesaplanması.

Bu araştırma konularının günlük yaşantıda örnekleri aşağıda sunulmuştur:

- Arabası ile giden sürücüye 15 dakika içinde yolu üzerindeki hastanelerin bildirilmesi,
- 3 dakika zaman dilimi içinde, araba ile gidilirken en yakın benzin istasyonunun belirlenmesi,
- Askeri birliğin mevcut yerine yakın/en yakın tedarik noktalarının belirlenmesi,
- İki arabanın nerede, hangi hızla belirli bir hedefte karşılaşacağının belirlenmesi.

1.1 Hareket Eden Nesne Veri Tabanları

Belirtilen araştırma konularında yapılan çalışmalarda geleneksel veri tabanı sistemlerinden farklı olarak, hareket eden nesneler veri tabanında olması gereken yetenekler aşağıda verilmiştir:

- Veri modelleme,
- Sorgu dili,
- Endeksleme,
- Belirsizlik.

Literatürdeki inceleme çalışmaları sürecinde, bu yetenekler ayrıntılı olarak incelenmiş ve her bir yeteneğe önemli kazanımlar getiren makaleler aşağıdaki bölümlerde özetlenmiştir.

1.1.1 Veri modelleme

Mevcut veri tabanları sürekli olarak değişen veriler (örneğin hareket eden nesnelerin konum bilgileri), ile uğraşmak için yeterli değildir. Bunun nedeni de verinin sabit olduğunun, değiştirme (update) komutu verilmediği sürece kabul edilmesidir. Örneğin maaş bilgisi 1000 TL ise değiştirme komutu verilmediği sürece, maaş bilgisi veri tabanında aynıdır. Bundan dolayı, hareket eden nesneleri (örneğin arabalar) veri tabanında temsil etmek ve konum bilgilerini sorgulamak için konum bilgilerinin sürekli olarak değişmesi gerekmektedir. Bu da performans sorunlarına yol açacağı için, gerçek zamanda tüm hareketli nesnelerden gelen

veriyi işleyebilmek mümkün değildir. Ayrıca kablosuz ağlar üzerinde hareket eden nesneler konum bilgisi değişimi bilgisi kısa zaman aralıklarında verildiğinde, veri tabanında bilgi değiştirme ciddi kablosuz band genişliği yükü oluşturmaktadır. Bu nedenlerden dolayı da hareketli nesnelerin, veri tabanında saklanabilmesi için gerçeklenebilir bir model oluşturulmaktadır (Wolfson vd., 1999; Frentzos vd., 2008).

1.1.2 Sorgu dili

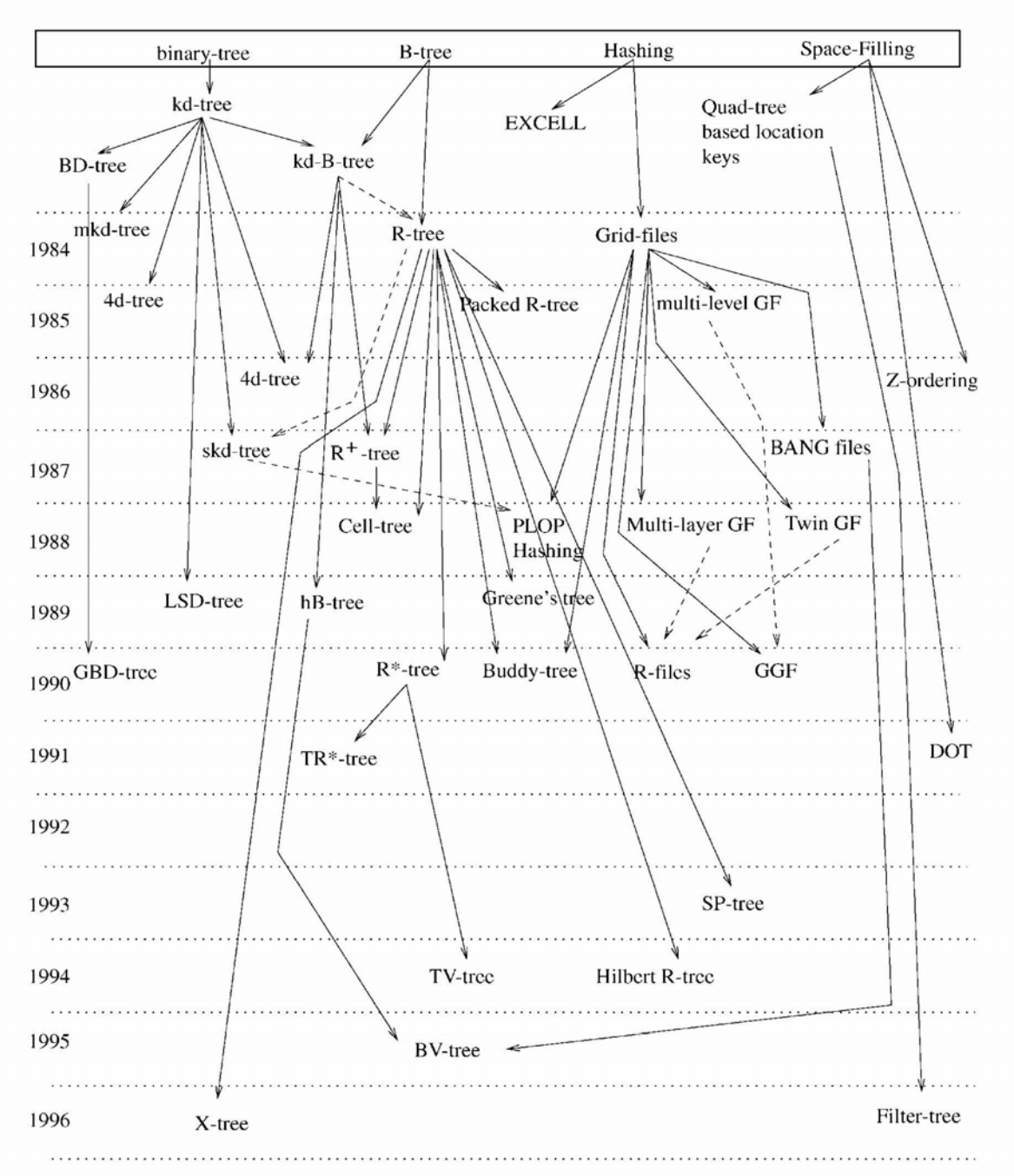
Hareket eden nesneler veri tabanları (Moving Objects Databases - MOD) uygulamalarındaki uzaya bağlı nesneler (noktalar, çizgiler, alanlar, poligonlar, gibi), uzaya bağlı (spatial) ve zamana bağlı (temporal) sınırlar içermektedir. P poligonu ile önümüzdeki üç dakika içinde kesişecek nesnelerin sorgulanması düşünüldüğünde, bu hem uzaya hem de zamana bağlı aralık (range) sorgusunu tanımlamaktadır. Uzaya bağlı alan P poligonu ve zamana bağlı alan da şimdiki an ve üç dakika arasındaki zamandır. Ayrıca, hem uzaya hem zamana bağlı birleştirme sorguları da uygulamalarda kullanılmaktadır. Aralarında 10 km. bulunan dost ve düşman uçak çiftlerinin bulunması ve bunun ne zaman olacağının sorgulanması birleştirme sorgularına örnek olarak verilmektedir.

Bahsedilen bu örnekler, zamana bağlı işlemleri de içerdiklerinden yapısal sorgu dili (SQL) gibi geleneksel sorgu dilleri ile ifade edilememektedir. Bu nedenle de hareket eden nesne veri tabanlarında, kullanıcı ayrıca kendi sorgu dilini tanımlamakta ve bu tanıma göre sorgularını gerçekleştirmektedir (Wolfson, O.vd. 1999).

1.1.3 Endeksleme

Veri tabanında temsil edilen hareket eden nesne sayısı çok fazla olabilmektedir. Bu nedenle, etkin olarak MOD sorgularına cevap verebilmek için endeks oluşturulmaktadır. Oluşturulan uzay bilgisi endeksi de nesneler hareket ettiği için sürekli olarak değişmektedir. Bu da veri tabanı yönetim sistemlerinde kabul edilemez bir durum oluşturmaktadır (Wolfson, O.vd. 1999). Gerçek zamanlı olarak bu problem, hala etkin olarak modellenememiştir.

Uzaya bağlı geliştirilmiş endeksler (Shekhar, S.ve S.C. 2003) Şekil 1.2'deki gibi sınıflandırılabilir (Bertino, E.ve Ooi, B. C.). Bir benchmark ile uzaya bağlı endeksler için performans karşılaştırması gerçekleştirilmiştir (Myllymaki, J.ve Kaufman, J. 2003). Hareketli nesneler söz konusu olduğunda ise uzay boyutuna ek olarak, zaman boyutu da endekse eklenmektedir (Abraham, T.ve Roddick, J. F. 1999).



Şekil 1.2 Uzaya bağlı endeks yapılarının gelişimi

1.1.4 Belirsizlik

Veri tabanındaki konum bilgisi değiştirildikten sonra nesnenin hareketinin devam etmesinden dolayı, veri tabanındaki konum bilgisi nesnenin gerçekteki konum bilgisinden her zaman farklıdır. Bu kalıtsal belirsizlik, veri tabanı modellemede, sorgulamada ve endekslemede göz önüne alınarak çözümler gerçekleştirilmektedir. Bu belirsizlikten dolayı örneğin aralık sorguları, iki farklı cevap oluşturabilmektedir: sorgunun cevabı olabilecek nesneler kümesi veya sorguyu mutlaka cevaplaması gereken nesneler kümesi. Bu soruna bir diğer yaklaşım da

bir nesnenin sorguyu karşılama olasılığının hesaplanmasıdır. Veri tabanlarında belirsizlik çok yönlü olarak araştırılmasına rağmen (Frentzos, E.vd. 2008a), yeni modelleme ve hem uzaya hem zamana bağlı özellikleri olan hareket eden nesneler için varolan çözümlerin tekrar incelenmesi gerekmektedir (Wolfson vd., 1999; Frentzos vd., 2008a).

Ayrıca belirsizlik ile ilgilenen araştırmalar, veri tabanında saklanan ham bilgiye bağlı belirsizlik bilgisi varsaymaktadırlar. Başlangıçtaki belirsizliğin nasıl elde edilebileceği sorusu MOD uygulamalarında, konum belirsizliğini nasıl ölçebiliriz sorusuna dönüşmektedir. Bilgilerin değiştirme yükü ile belirsizlik arasındaki değerlendirme çalışmaları, hareket eden nesnenin ne kadar zaman süresi içinde konum bilgisini değiştireceğinin belirlenmesi ve hareket eden nesnenin bağlantısının kopması durumunda değiştirme bilgisini gönderememesinde yapılması gerekenler bu kapsamda araştırılmaktadır (Trajcevski, G.vd. 2004).

1.2 Tezde Yapılan Çalışmalar

Hem uzaya hem de zamana bağlı (spatio-temporal) verilerin kullanıldığı uygulamalar için, veri modelleme ve bu tip verilerin veri tabanlarında saklanabilmesi için gerekli etkin erişim yöntemlerinin bütünleştirildiği az sayıda çalışma gerçekleştirilmiştir (Mallett, 2004; Mallett vd., 2005; Huibing, 2008). Bu tez çalışmasında hem uzaya hem de zamana bağlı verileri modelleme, kullanıma hazır veri tabanlarında endeksleme ve yeni sorgu dili tanımlama konuları incelenmiştir. İlişkisel Veri tabanı Yönetim Sistemleri (RDBMS - Relational Database Management System) desteğini sağlamak üzere, Oracle 10g veri tabanı çalışmada kullanılmıştır.

RDBMS; eşzamanlılık kontrolü, yedekleme ve verileri kurtarma mekanizmaları, çoklu kullanıcı ve çoklu birim işlemleri (transaction) desteği ve büyük veri tabanlarının yönetimi için çeşitli araçları kullanıcıya sunmaktadır (Botea, V.vd. 2008). Bu veri tabanlarında, basit verileri (tamsayı, ondalık sayı, metin veriler, vb.) tutan ilişkisel tablolar bulunmaktadır (Obalı, M. 2007). Bu verilere, kullanıcının daha hızlı erişmesini sağlamak için endeksleme işlemi gerçekleştirilmektedir. Karmaşık verilerin, veri tabanlarında saklanabilmesinden sonra da bu verilere hızlı bir şekilde ulaşılabilmesi için yeni yöntemler ve araştırmalar geliştirilmektedir. Bu karmaşık verilere örnek olarak, geometrik ve konuma bağlı veriler gösterilebilir (Oracle 2006). Konuma bağlı nesneyi basit veri tipiyle eşleştirerek veri tabanında endeksleme yerine, özel bir uzaya bağlı nesne tipi ve çok-boyutlu endeksleme yapısı ile büyük veri tabanlarında veriye ulaşma performansı daha çok artırmaktadır (Mallett, D.vd. 2005). Aynı şekilde, veri

tabanlarında zamana bağlı verilerin endekslemesi üzerinde çeşitli çalışmalar gerçekleştirilmiştir (Dumas, M.vd. 2004).

Hareket eden nesneler için gerçekleştirilen alan uygulamalarında, RDBMS içinde hem uzaya hem de zamana bağlı verilerin desteklenmesi istenmektedir. Konumunu değiştiren herhangi bir şey, *hem uzaya hem de zamana bağlı veri* olarak isimlendirilmektedir. Günümüzde kablosuz cihazlar, cep telefonu gibi GPS takılı cihazlar, konum bilgisinin elde edilmesine olanak sağlamaktadır (Mallett, D.vd. 2005).

Hem uzaya hem de zamana bağlı verilere erişim (Spatio-Temporal Access Methods – STAMs) yöntemleri hakkında bir çok çalışma gerçekleştirilmesine rağmen (Mokbel vd., 2003; Pelekis vd. 2004; Krassimir Markov 2008), RDBMS içinde bu yöntemlerin nasıl tanımlanacağı hakkında çok az çalışma bulunmaktadır (Kothuri ve Ravada, 2002; Kothuri vd., 2002; Mallett vd., 2005)

1.2.1 Kavramsal veri modelleme konusunda yapılan çalışmalar

Bu tip veriyi modelleme için, geleneksel veri tabanı sistemlerinde varlık ilişki modellerinin (Entity-Relationship) veya nesneye yönelik programlarla oluşturulan sistemlerde UML (Unified Modeling Language) şemalarının yetersiz kalmasından dolayı, tez çalışması kapsamında hareket eden nesneler kavramsal veri modeline ihtiyaç duyulmuştur.

Hem uzaya hem zamana bağlı veriyi kavramsal olarak modellemek için, varlık ilişki (entity relationship) şemalarını genişleterek kullanan iki modelleme çalışması tez çalışmasında incelenmiştir: SpatioTemporal Entity-Relationship (STER) (Tryfona, N.vd. 1999) ve SpatioTemporal eXtended Entity-Relational model (STXER) (Jin, P.vd. 2008).

Nesneye-dayalı modelleme yapısı kullanılarak Parent ve arkadaşları tarafından geliştirilen MADS (Modeling of Application Data with Spatio-temporal features) (Parent, C.vd. 2006) modeli ile sistemler etkin olarak tasarlandığı için tez çalışmasında incelenip, görsel editörde incelenen problemin kavramsal model oluşturulmuştur. Tez çalışması kapsamında gerçekleştirilen kavramsal modelleme deneme çalışmaları (Kilimci, P.ve Kalıpsız, O. 2008a; Kilimci, P.ve Kalıpsız, O. 2008b)'de sunulmuştur.

1.2.2 Veri modelleme konusunda yapılan çalışmalar

Hem uzaya hem de zamana bağlı veya hareket eden nesneler (Guting, R. H.ve Schneider, M. 2005) veri tabanlarında üç ana tipte endeksleme çalışmaları gerçekleştirilmektedir. Bunlar;

geçmişe, mevcut zamana ve geleceğe ait verileri endeksleme yöntemleridir (Krassimir Markov, K. I., Ilia Mitov, Stefan Karastanev 2008). Bu tez çalışmasında ilk yönteme göre, geçmişe yönelik verilerin endekslemesi incelenmiştir. Hareket eden nesnelerin herhangi bir zamanda bilgileri sorgulanabilmektedir. Hareket eden nesnelerin bilgileri, belirli aralıklarla veri tabanında saklanmaktadır. Hareket eden nesneler, verilerini<$movObj_{id}$, x, y, t_s, t_e, v> şeklinde göndermekte olduğu varsayılmaktadır. Burada,

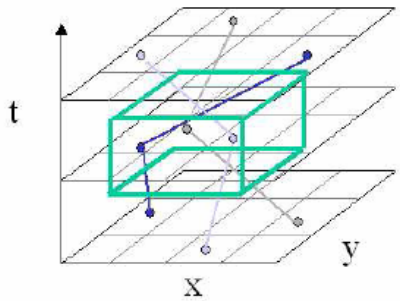
- $movObj_{id}$, hareketli nesnenin belirleyici numarasını,
- <x, y>, konum koordinatlarını,
- <t_s, t_e>, hareketli nesnenin <x, y> konumunda kaldığı zaman aralığını,
- v, hareketli nesnenin hızını göstermektedir.

Sayısal harita olarak, İstanbul Büyükşehir Belediyesinden, Eminönü-İstanbul haritası temin edilmiştir. Bu harita shape dosyalarına dönüştürülerek, Oracle veri tabanında saklanmıştır. Tez çalışması kapsamında gerçekleştirilen veri tabanında veri modelleme deneme çalışmaları (Kilimci, P.ve Kalıpsız, O. 2007a; Kilimci, P.ve Kalıpsız, O. 2007b)'de sunulmuştur.

Hareket eden nesneler için veriler, gerçeğe uygun olarak tez çalışmasında hem uzaya hem zamana bağlı Brinkhoff'un geliştirdiği üreteç (Brinkhoff, T. 2000) ile Eminönü-İstanbul haritası üzerinde üretilmiştir. Üretecin parametreleri değiştirilerek ağ üzerinde hareket eden farklı veriler oluşturularak deneyler gerçekleştirilmiştir.

1.2.3 Endeksleme konusunda yapılan çalışmalar

Bir hem uzaya hem zamana bağlı aralık (range) sorgusu Q, Q = <R ,T> şeklinde gösterilir. Burada R, uzaya bağlı bir alanı ve T de zaman aralığını ifade eder. Q, R içinde bulunan <x, y> noktalarını ve T ile kesişen <t_s, t_e> zamanları içeren kayıtların farklı nesne id'lerini geri göndermektedir (Şekil 1.3). Bu tez çalışmasında incelenen problem, hem uzaya hem zamana bağlı verilerin endekslenmesi ve en kısa zamanda aralık sorgu sonucunun elde edilmesidir.



Şekil 1.3 Hem uzaya hem zamana bağlı aralık sorgusu

Hareket eden nesne bilgilerine hızlı ulaşmak için Space Partititoning with Indexes on Time yöntemi, R-tree + zamana bağlı B-tree ve Z-değerleri + B-tree endeks yapıları oluşturularak performans ölçümleri elde edilmiştir. Bu yöntemlerden Space Partititoning with Indexes on Time, z-değerlerine göre gerçekleştirilmiş ve Z-değerleri + B-tree yöntemi bu tez kapsamında iyileştirilmiştir. Oluşturulan bu sistemde bulunacak hızlı hareket etmiş nesnelerin sayısının az olmasından dolayı, hızlı nesneler için ayrı bölümlenme algoritması ortaya konmuştur. Endeks yapısının hızlı nesnelerle bozulmamasından dolayı, performans artışları elde edilmiştir.

1.2.4 Sorgu dili konusunda yapılan çalışmalar

Oracle Veri Kartuşu kullanılarak, hareket eden nesneler için sorgu dili PL/SQL dilinde tanımlanmıştır (Oracle 2001b; Oracle 2001c; Oracle 2001a). Bunun için zamana, uzaya ve hem uzaya hem zamana bağlı veri tipleri ve işlemleri tanımlanarak SQL cümleleri ile sorgular gerçekleştirilmiştir.

1.3 Tezin İçeriği

Hareket eden nesne veri tabanları kapsamında gerçekleştirilen endeksleme çalışmaları ve veri oluşturma için kullanılan üreteçler Bölüm 2'de verilmiştir. Bu bölüm, bu alanda yapılan en son çalışmaları içererek kapsamlıca incelenmiştir. Bölüm 3'te hem uzaya hem zamana bağlı kavramsal modelleme çalışmaları araştırılmıştır. Bölüm 4'te kavramsal sorgu dili çalışmaları incelenmiştir. Bölüm 5'te uzaya bağlı kavramların veri tabanında kullanımı incelenmiştir. Bölüm 6'da ismini MOORA (Moving Objects within ORAcle) olarak verip tanımladığımız, hareket eden nesneler sorgu sistemi ve algoritmaları tanımlanmıştır. Bölüm 7'de gerçekleştirilen deneyler açıklanmış ve elde edilen sonuçlar tartışılmıştır. Son olarak Bölüm 8'de, tez kapsamında oluşturulan MOORA sisteminden elde edilen tecrübe, deney sonuçları ve bundan sonra yapılabilecek çalışmalar hakkında bilgi verilmiştir.

2. HAREKET EDEN NESNELERİN ENDEKSLEME YAPILARI VE VERİ ÜRETEÇLERİ

Hareket eden nesneler, *zamana bağlı geometrilerdir*. Diğer bir ifade ile geometriler, zamana bağlı fonksiyonlar olarak ifade edilmektedir. Hem uzaya hem zamana bağlı veri tabanlarında yapılan çalışmalardan farklı olarak, geometriler *sürekli* olarak değişmektedir.

Son on yılda hareket eden nesneler konusunda bir çok araştırma yapılmıştır. Bunların çoğu özel endeks yapıları veya özel tipteki sorguları etkin olarak gerçekleştiren algoritmalar konularına odaklanmıştır. Bu alandaki olgunlaşma ancak önemli sorguları yapan tam sistemlerin kullanıma sunulması ile anlaşılabilir. Fakat günümüze kadar, bu tip çok az sistem bulunmaktadır.

Bu bölümde, literatürde yapılan en son çalışmalar incelenerek, hareket eden nesnelerin endekslenmesi incelenmiştir. Ayrıca hareket eden nesne verilerini üreten veri üreteçleri de detaylı olarak araştırılmıştır.

2.1 Endeksleme Yapıları

Hareket eden nesne verisi konusunda geçmiş yıllarda iki farklı görüş oluşmuştur. Bunlardan ilki, hareket eden nesnelerin mevcut ve öngörülmüş zamanda yakın geleceğe ait soruları cevaplamaya odaklanmıştır. Buna örnek olarak kamyon filosunun mevcut ve yakın gelecek zamanda gerçek zamanlı hareketlerini saklama verilebilir. Bu yaklaşım, bazen izleme (tracking) olarak isimlendirilmiştir. Bu yönteme uygun olan sorgularda, Moving Objects Spatio-Temporal (MOST) modeli (Sistla, A. P.vd. 1997) ve Future Temporal Logic (FTL) dili nesnelerin mevcut durumları için (Stojanovic, D.ve Dordevic-Kajan, S. 2003) teklif edilmiştir (Wolfson, O.vd. 1998; Wolfson, O.vd. 2002).

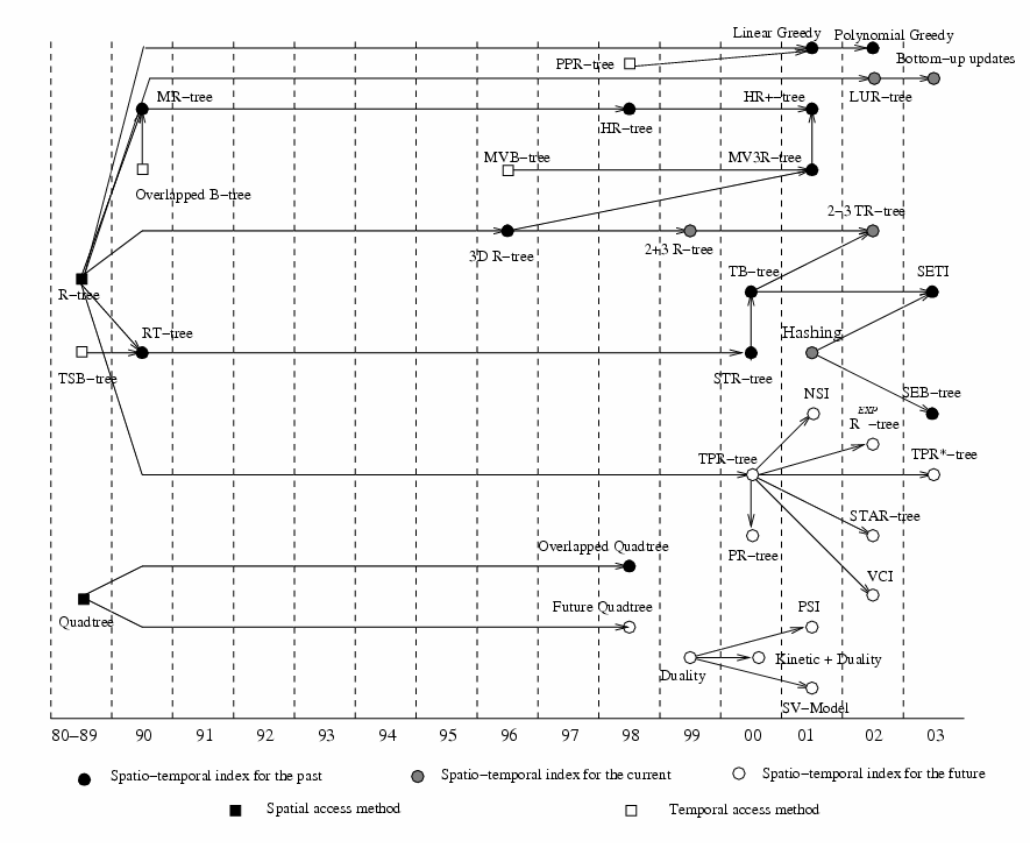
İkinci yaklaşımda ise hareket eden nesnelerin tüm geçmiş bilgileri tutulmaktadır (Güting, Böhlen vd. 2000). Bu veri tabanları, *gezinge* (trajectory) veri tabanları olarak da adlandırılmaktadır (Forlizzi, L.vd. 2000). Bu yaklaşım çeşitli çalışmalarda da araştırılmıştır (Frentzos, E. 2003; Almeida, V. T. D.ve Güting, R. H. 2005; Güting, H.vd. 2006; Alvares, L. O.vd. 2007).

Hareket eden nesneler konumlarını, konum belirleme araçları ile belirleyerek kablosuz ağlar vasıtasıyla sunucuya gönderirler. Daha sonra sunucu, nesnelerin konumlarını değiştirme bilgisini alarak, her nesnenin hem uzaya hem zamana bağlı tarihçesini saklar. Sunucunun desteklediği tipik sorgular; **time slice** ve **window** sorgularıdır. “t zamanında belirli bir alandan

geçerken tüm nesnelerin bulunması" tipik bir *time slice* sorgusudur. "[t_1, t_2] zaman aralığında belirli bir alandan geçerken nesnelerin bulunması" ise tipik bir *window* sorgusudur. İki sorgu tipi için de geçmiş, mevcut ve gelecek zamanlar için sorgulama yapabilir. *Gezinge* sorguları sadece geçmiş ile ilgilenir, örneğin "geçmiş bir saat içinde belirli bir nesnenin en yüksek hızının belirlenmesi", "birbirine benzeyen gezingelerin belirlenmesi" (Frentzos, E.vd. 2007a), vb. Hareket eden *window* sorguları ise sadece gelecek ile ilgilenir, örneğin "belirli zaman aralığında hareket eden alan ile kesişen nesnelerin bulunması" (Mokbel, M. F.vd. 2003).

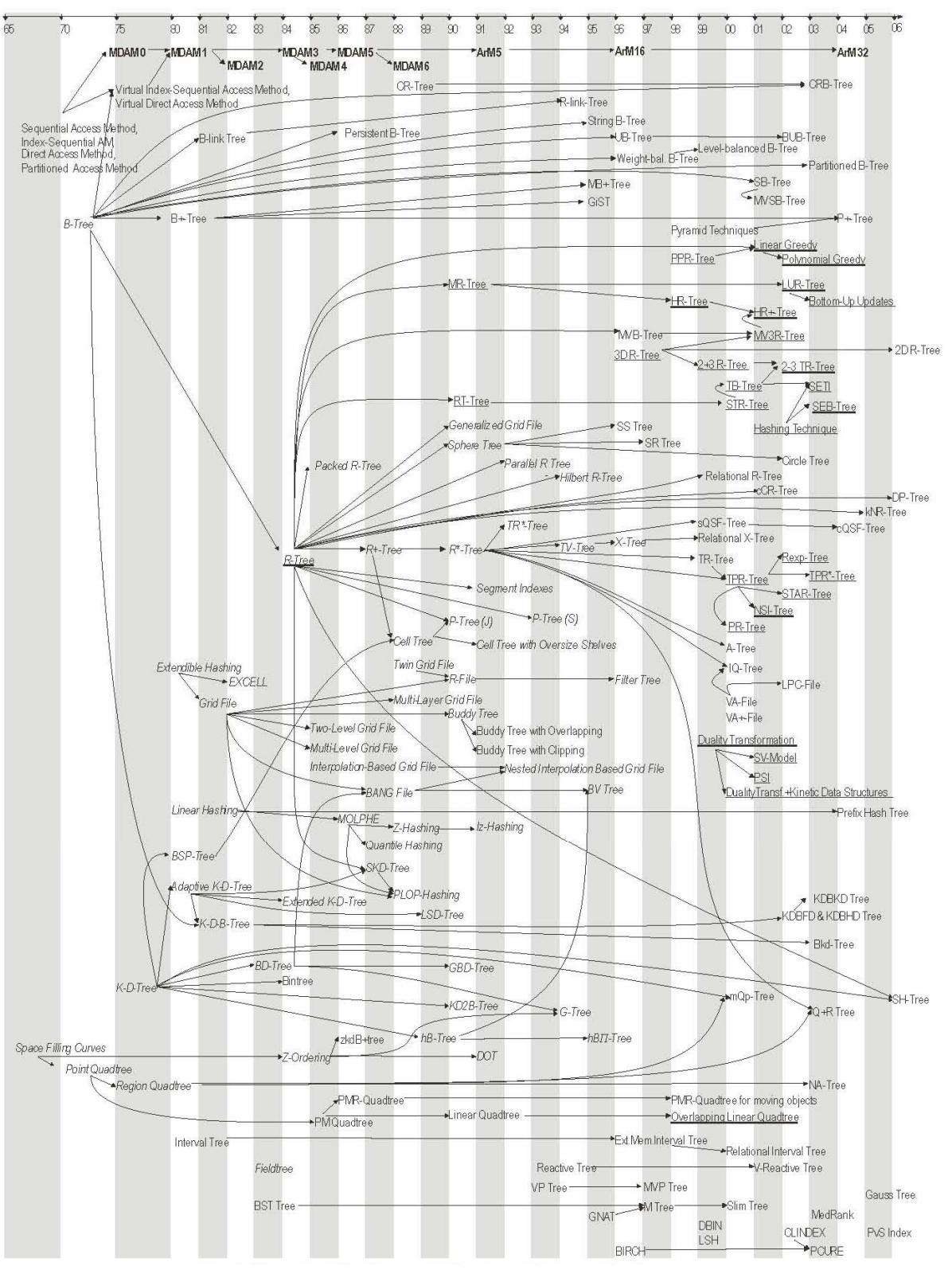
Hem uzaya hem zamana bağlı sorguları desteklemek için, erişim yöntemlerini geliştirme alanında çalışmalar yapılmıştır (Ni, J. 2007). Etkin sorgu işleme için endeksleme yöntemleri, yukarıda açıklanan mevcut, gelecek ve geçmiş zaman yaklaşımları için çok önemlidir (Koubarakis, M.ve T.S.e.a. 2003).

Uzaya, zamana, hem uzaya hem zamana bağlı geliştirilen endeks yapıları Şekil 2.1'de gösterilmiştir (Mokbel, M. F.vd. 2003). Erişim yöntemleri arasındaki çizgiler, hem uzaya hem zamana bağlı endeks yapısının hangi yapıyı temel aldığını göstermektedir.



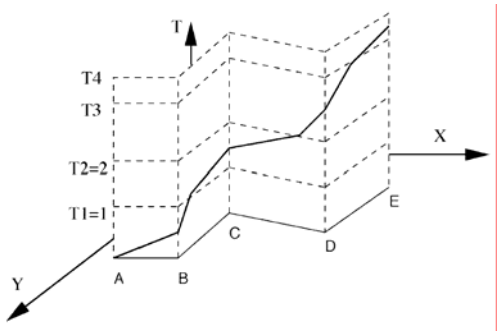
Şekil 2.1 Uzaya ve zamana bağlı erişim yapılarının gelişimi

Diğer özet çalışma, günümüze kadar gerçekleştirilen veri tabanı erişim yöntemlerini Şekil 2.2'de özetlemektedir (Krassimir Markov, K. I., Ilia Mitov, Stefan Karastanev 2008). Altı çizili olan erişim yöntemleri Mokbel'in çalışmasından alınmıştır.



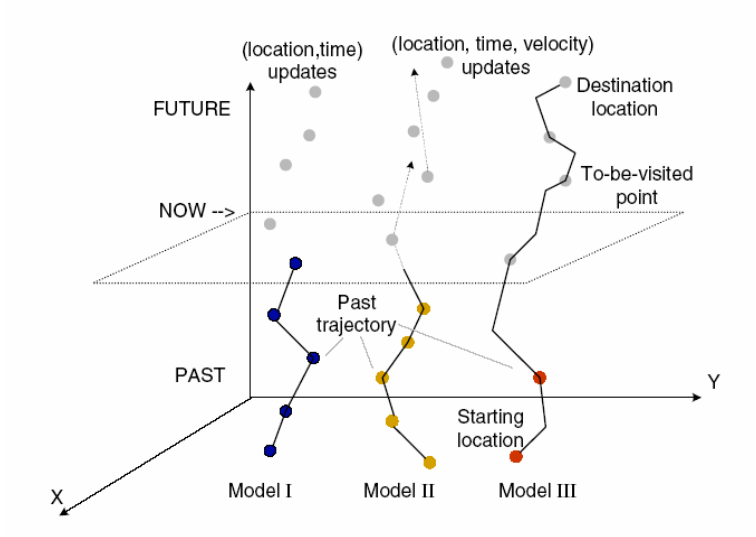
Şekil 2.2 Erişim yöntemleri

Geçmiş zamanla ilgilenen sorgularda hareket eden nesnelerin tarihçe bilgisini küçültmek için ise iki yöntem kullanılmaktadır. Bunlar *örnekleme* ve sadece *nesnenin bilgi değişimidir*. Örneklemede, her zaman diliminde nesne bilgisini saklama yerine belirli aralıklarda nesne bilgisini saklama yapılmaktadır. Bu örnek aralıkların dışında interpolasyon yöntemi ile gezinge noktaları oluşturulmaktadır (Şekil 2.3). İkinci yöntemde ise hareket eden nesne, bilgilerinin değişmesi durumunda sunucuya haber vermektedir. Böylece hızında veya yönünde bir değişiklik olduğunda hareket eden nesne, sunucuya konum bilgisini göndermektedir.



Şekil 2.3 Örnek bir gezinge

Nesneler pek çok çalışmada serbest hareket etmektedir. Bazı çalışmalar da gerçek dünya uygulamalarını göz önünde bulundurarak, kısıtlı hareket eden nesneler için incelemiştir (Vazirgiannis ve Wolfson 2001). Örneğin, ulaşım ağlarında sınırlı hareketler (Güting, H.vd. 2006) ve bunlar için mevcut özel endeks yapıları ile bir çalışmada incelenmiştir (Almeida, V. T. D.ve Güting, R. H. 2005). Aşağıda mevcut, gelecek ve geçmiş zaman için endeks yapıları incelenmiştir (Şekil 2.4).



Şekil 2.4 Hareket eden nesne modelleri (Ding, H.vd. 2008)

2.1.1 Geçmiş zamanda endeksleme yapıları

Geçmiş zamana ait veriler için hem uzaya hem zamana bağlı endeks yapıları üç gruba ayrılmaktadır. Birinci grupta endeksler, varolan uzaya erişim metotlarına zaman boyutunun eklenmesi ile oluşturulur. İkinci grupta uzay ve zaman bilgileri bir yapıda yönetilir. Üçüncü yapıda ise zaman boyutu endekslenir ve uzay boyutuna ise ikincil öncelik verilir.

Geçmişe yönelik hem uzaya hem zamana bağlı endeksleme yapıları, **zaman boyutu, örtüşen (overlapping) ve çok-sürümlü yapılar** ve **gezinge-yönelimli yapılar** olarak üç grupta incelenmektedir.

2.1.1.1 Zaman boyutu

Zaman boyutu ile ilgilenirken, hem uzaya hem zamana bağlı endekslemede temel olarak uzay ile ilgilenilir. Zaman sorgularının ikincil önemi vardır. Bu grupta RT-tree, 3D R-tree ve STR-tree endeksleme yapıları bulunmaktadır (Mokbel, M. F.vd. 2003).

RT-tree'de uzay boyutu için R-tree (Guttman, A. 1984; Manolopoulos, Y.vd. 2000) ve zaman boyutu için TSB-tree yapıları birleştirilerek oluşturulur. Nesnenin başlangıç ve bitiş zamanları R-tree'ye eklenir (Brakatsoulas, S.vd. 2002). R-tree'de veriler, (id, MBR, t_s, t_e) sırasındadır. Burada id, nesnenin belirleyici numarası, MBR ise nesneyi çevreleyen en küçük dikdörtgeni, t_s ve t_e zaman aralığını göstermektedir. RT-tree, R-tree gibi etkin şekilde uzay sorgularını gerçekleştirmektedir. Fakat time slice ve aralık sorguları için tüm ağacın gezilmesi gerekmektedir (Xu, X.vd. 1990).

3D R-tree, zamanı uzayın diğer bir boyutu gibi ele alır. Bu yöntem, uzay ve zaman sorgularını farklı ele almamaya çalışır. 3D R-tree zaman ve uzay sorgularını desteklemesine rağmen, performans sorunları ortaya çıkar. Burada en önemli sorun, time slice sorgularının sorgu zamanında gerçek bilgilere bağlı olmayarak hareketin geçmişindeki toplam bilgiye bağlı olmasıdır (Theodoridis, Y.vd. 1996).

STR-tree yapısı, farklı ekle/böl (split) algoritması ile R-tree'nin genişletilmesidir. Yaprak düğümlerde bilgi (id, t_{id}, MBR, o) şeklinde saklanmaktadır. Burada t_{id}, gezingenin belirleyici numarasını ve o ise MBR'daki gezingenin yönünün göstermektedir. STR'de temel düşünce, konum yakınlığının ve kısmi gezingenin, aynı gezingedeki segment çizgilerinin konum yakınlığını koruyarak birlikte tutmadır. Uzay özellikleri ve gezinge korunması arasındaki dengeleme için p parametresi oluşturulmuştur. Gezinge korunması için ayrılan seviyeleri p

parametresi belirlemektedir (Pfoser, D. 2000). Bir segment çizgisi eklenirken, p seviyedeki önceki gezingeye mümkün olduğunca yakın yerleştirme hedeflenmektedir (Pfoser, D.vd. 2000).

2.1.1.2 Örtüşen ve çok-sürümlü yapılar

Örtüşen (overlapping) ve çok-sürümlü yapılar zaman boyutunu, uzay boyutundan ayırmaktadır. Burada hedef, uzay bilgisini bir zaman instance'ında birlikte ve canlı olarak bir endeks yapısında tutmaktır. Her zaman instance'ı için ayrı bir R-tree oluşturulur. Bu da çok disk alanını kaplamaktadır. Bu grupta geliştirilen endeks yapıları; MR-tree, HR-tree, HR+-tree, MV3R-tree ve PPR-tree'de fırsatçı (greedy) algoritmalardır.

MR-tree yapısı örtüşen B-tree yapısını, R-tree ile oluşturur. Her timestamp için R-tree disk alanı yükünden kaçınmaya çalışır. Bunu da R-tree'deki art arda gelen nesneler arasında ortak nesneleri saklamayarak gerçekleştirir. Farklı ağaç köklerinden farklı zamanlarda aynı değerlere sahip aynı düğümlere bağlantılar (linkler) oluşturarak elde eder. Bu fikir, time slice sorguları için en iyi çözüm oluşturur. Bu ağaçta arama ise, uygun kökten başlayarak R-tree'de uzay araması yapılarak elde edilir. Buna karşılık window sorgularında performans etkin değildir. Ayrıca, bir çok bilgi kopyalanmaktadır. Art arda gelen timestamp'lerde sadece bir düğümün bilgisinin değişmesi durumunda, diğer tüm düğümler iki art arda gelen R-tree de kopyalanmaktadır (Xu vd., 1990; Nascimento ve Silva, 1998).

HR-tree, MR-tree'ye benzemektedir. Örtüşen B-tree detaylarını, R-tree'de kullanan somut bir algoritması bulunmaktadır (Nascimento, M. A.ve Silva, J. R. O. 1998).

HR+-tree, HR-tree'deki kopyalardan kaçınarak oluşturulur. HR+-tree, farklı timestamp'lerin aynı düğümde saklanması ile elde edilir. Fakat, her R-tree'eki ata düğüm, aynı ata'nın timestamp değerine sahipse erişebilmektedir. Diğer bir ifadeyle, bir düğümün birden fazla ata düğümü olabilir. Fakat bu ata düğümler, çocuk düğümün farklı parçalarına erişebilmektedir (Tao, Y.ve Papadias, D. 2001a).

MV3R-tree yapısı da çok versiyonlu B-tree yapısına benzemektedir. İki ağaç oluşturulur. MVR-tree'si timestamp sorguları ve 3D R-tree ise uzun aralık sorguları için oluşturulur. Kısa aralık sorguları hangi ağacın eşik değerini temel alarak kullanıldığının kontrol edilmesi ile en iyilenir (Tao, Y.ve Papadias, D. 2001b).

PPR-tree'de fırsatçı (greedy) algoritmalarda ise R-tree kısmi olarak kalıcıdır (Partially-Persistent R-tree – PPR-tree). Bitemporal veri tabanları için geliştirilerek, hem uzaya hem

zamana bağlı uygulamalar için genişletilmiştir. Fakat, bu da çok fazla ölü alan oluşturmaktadır. Ölü alanları engellemek için suni nesne değişiklikleri yapılmaktadır. Doğrusal hareket eden nesnelerde suni değişim bilgileri için en iyi yeri bulan fırsatçı algoritmalar için çeşitli çalışmalar gerçekleştirilmiştir (Kollios vd., 2001; Hadjieleftheriou vd., 2002).

2.1.1.3 Gezinge-yönelimli erişim yöntemleri

Gezinge-yönelimli sorgulara erişme yapısı oluşturmaktadır. Bu yöntemde nesneleri uzaya bağlı olarak yan yana tutmak ikincil hedeftir. Bu alanda geliştirilen endeks yapıları TB-tree (Trajectory Bundle Tree), SETI (Scalable and Efficient Trajectory Index) ve SEB-tree (Start/End timestamp B-tree)'dir.

TB-tree, gezingeleri koruyarak oluşturulan R-tree'dir. Yaprak düğümler aynı gezingeye ait doğru segmentleri tutmaktadır. Fakat, farklı gezingeler konum olarak yakın ve doğru segmentler olarak farklı düğümlerde saklanır. TB-tree soldan sağa doğru büyümektedir (Pfoser, D. 2002). En soldaki yaprak düğüm ağaca yerleştirilen ilk düğümdür (Frentzos, E. K. 2008). En sağdaki yaprak düğüm de en son yerleştirilen düğümdür. TB-tree, STR-tree'nin gezingeler için uyarlanmış halidir (Pfoser, D.vd. 2000; Frentzos, E. K. 2008).

SETI, uzay boyutunu sabit örtüşmeyen bölümlere (partition) bölmektedir. Burada önemli bulgu, zaman boyutunun sürekli olarak değişirken uzay boyutunun kısıtlı olarak değişmesidir. Bundan dolayı uzay boyutu sabit olarak bölümlere ayrılmıştır. Her bölümde gezinge segment'leri R-tree ile endekslenmiştir. İyi bir bölümleme fonksiyonu, aynı gezingeye ait her line segment'ini aynı bölümde saklamalıdır. Böylece R-tree'de uzay boyutu minimize edilerek gezinge korunması sağlanır. İki bölüm sınırlarında olan bir segment, iki bölümde de saklanmaktadır. Bu da sorgu sonucunda birden fazla aynı bilginin elde edilmesini sağlar (Chakka, V. P.vd. 2003).

SEB-tree, SETI'ye benzemektedir. Fakat burada uzay, örtüşen bölgelere (zone) bölümlenmiştir. Her bölge, hareket eden nesnenin başlangıç ve bitiş timestamp'lerine göre SEB-tree yapısı ile endekslenmiştir. Her hareket eden nesne bölgeye, hash fonksiyonu ile ulaşır. SETI'den farklı olarak, bu yapıda gezingeler yoktur. Sadece iki-boyutlu noktalar endekslenmiştir. Uzay bölgelerini bölümleyerek aynı gezingeye sahip iki-boyutlu noktalar yan yana bulundurulurlar (Song, Z.ve Roussopoulos, N. 2003).

2.1.2 Mevcut zaman için endeksleme yapıları

Önceki bölümde açıklanan endeksleme teknikleri, tüm hareketlerin önceden bilinmesini gerektirmektedir. Mevcut uzay için hareketli nesneler ne saklanmıştır ne de sorgulanmıştır. Şu anki zamanda (NOW) nesnelerin uzaylarını sorgulama, çok boyutlu bir problemdir. NOW hakkındaki sorguları cevaplamayı sağlayan endeks yapıları; 2+3 R-tree, 2-3 TR-tree, LUR-tree, aşağıdan-yukarı değişimler ve hashing'dir.

2+3 R-tree, nesnelerin mevcut ve geçmişe ait bilgilerini endeksler. İki tane R-tree yapısı vardır. Bir R-tree, mevcut iki-boyutlu noktaları tutar. Diğer R-tree de üç-boyutlu (iki boyut uzaya, tek boyut da zamana ait) gezingeleri tutan tarihçe-ağacıdır. Mevcut nesne değişikliği yapıldıktan sonra, nesnenin gezingesi üç boyutlu MBR ile oluşturulur. Daha sonra iki boyutlu R-tree'den silinir ve 3 boyutlu R-tree'ye eklenir. Sorgu zamanına bağlı olarak iki ağaç da aranır (Nascimento, M. A.vd. 1999).

2-3 TR-tree, 2+3 R-tree'ye benzemektedir. Fakat, 2-3 TR-tree'deki 3-boyutlu R-tree gezinge yerine çok-boyutlu noktaları tutarak ölü alanları oluşturmaktan kaçınır. TB-tree yapısını kullanmasından dolayı gezinge-yönelimli sorguları rahatlıkla cevaplar (Abdelguerfi, M.vd. 2002).

LUR-tree (Lazy Update R-tree), hem uzaya hem zamana bağlı nesnelerin mevcut uzayları ile ilgilenir. LUR-tree'de geçmişe ait bilgiler tutulmamaktadır. Nesne yerini değiştirdikten sonra, nesnenin eski bilgileri silinerek, yenisi eklenir. LUR-tree, R-tree endeks yapısının performansını bozmadan, hareket eden nesnelerin sık değişimlerini çözmek için tasarlanmıştır. Hareket eden nesnenin yeni uzayı MBR içinde olduğu sürece, nesnenin uzay bilgisi değiştirilmez. MBR dışına hareket ettiğinde de iki yaklaşım söz konusudur. İlkinde nesne silinir ve tekrar eklenir. Bu da birleştirme ve bölme işlemlerini gerekli kılar. İkinci yaklaşımda nesne MBR'dan çok uzağa gitmediği sürece, MBR yeni uzayı içine alacak şekilde genişletilir (Kwon, D.vd. 2002).

Aşağıdan-yukarı değişimler LUR-tree yapısını genişletir. Birçok aşağıdan-yukarı yaklaşımı, hareket eden nesnelerin bilgilerinin sık değişmesi durumu için önerilmiştir. Bunlara örnek olarak, yeni değeri içerecek şekilde MBR'ın genişletilmesi ve mevcut nesnenin kardeş düğüme hareket ettirilmesi verilebilir. Değişimin olduğu düğümün kardeş ve ata düğümlerini araştırırken yüklü G/Ç işlemlerden kaçınmak için, ana bellek yapıları önerilmiştir (Lee, M.vd. 2003).

Hashing yönteminde uzay örtüşen alanlara bölümlenir (partition). Nesne alanını değiştirmediği sürece, bilgileri veri tabanında aynı kalır. Bu nedenle, veri tabanı hareket eden nesnelerin yaklaşık uzaylarını tutar. Belirsizliği çözmek için hareket eden nesneler ile veri tabanı arasına filtre katmanı oluşturulmuştur. Aralık sorguları, alan kümesine çevrilmektedir. Aralık sorgusunda bir bölge tamamıyla kaplanmışsa, sorgu cevabı olarak tüm alandaki bilgiler döndürülür. Fakat aralık sorguları ile bir bölge kesişirse, alandaki tüm bilgiler filtre katmanına gönderilerek aralık sorgusunu cevaplayıp cevaplamadığı kontrol edilir (Song, Z.ve Roussopoulos, N. 2001).

2.1.3 Mevcut ve gelecek zaman endeksleme yapıları

Hareket eden nesnelerin gelecekteki yerlerini tahmin etmek için, *hız* ve *hedef* gibi ek bilgilerin de saklanması gerekmektedir. Nesnelerin doğrusal hareketlerini modellemek için $x_t = at + b$ denkleminden faydalanılarak tek boyutlu olarak modellenmektedir. Bu tip endeksleme yapısı olarak; PMR-quadtree, RUM-tree, B^x-tree, B^{dual}-tree ve STRIPES verilebilir.

PMR-quadtree, gelecek gezinge noktalarını endeksler. Hareket eden nesnenin bilgisi değiştiğinde, tüm endeks yapısı bozularak yeni bilgiye göre yeni endeks oluşturulur. Çok sayıdaki değişim işlemlerinde, endeks ΔT zaman sonra tekrar oluşturulur. Kavramsal düzeyde, sonsuz zaman, eşit zaman dilimlerine bölünmektedir. Her bir dilim için, hareket denklemine göre PMR-quadtree oluşturulmaktadır. Bellek sınırlarından dolayı da sadece mevcut zaman için PMR-quadtree yaratılır (Tayeb, J.vd. 1998).

Geleceğe ait gezingeleri endeksleme metotlarında iki sakınca oluşmaktadır. Birincisinde en küçük dikdörtgende gezingeyi tutmak için, çok büyük ölü alanlar oluşmaktadır. İkincisinde ise zamanın bitiş değerleri hep aynı olduğu için gezingelerde, tüm veri bir tarafa yönelmektedir (skewed).

Hem uzaya hem zamana bağlı endeksleme yapısında oluşan sorunları engellemek için zaman-uzay alanı, başka bir uzaya dönüştürülür (duality transformation). Burada daha kolay bilginin temsil edilmesi ve sorgulanması temel olarak ele alınmaktadır. Bu konuda gerçekleştirilen çalışmalar, devimsel veri yapısı ile duality transformation, SV-Model, PSI (Parametric Space Indexing Technique)'dir. Genel olarak ele alındığında dönüştürme teknikleri, üç sorunla karşılaşmaktadır. İlk olarak, ana uzaydaki tüm hareketleri dual uzay yakalayamamaktadır. İkinci olarak, ana uzayda yan yana olan nesnelerin dual uzayda da hala yan yana olması belirsizdir. Üçüncü olarak ana uzaydaki dikdörtgen şeklindeki aralık sorguları, dual uzayda

poligon aralık sorgularına her zaman dönüştürülmektedir. Bu da algoritmaların geliştirilmesini karmaşıklaştırmaktadır (Mokbel, M. F.vd. 2003).

RUM-tree (R-Tree with Update Memo) yapısında, nesnenin bilgisinin değişmesi sırasında eski bilginin silinmesinde disk erişimlerini azaltmak için ana-belek memo'nun kullanmasını teklif etmektedir. Bu nedenle bilginin değişim maliyeti, bilgiyi eklemeye denktir. Özellikle, nesne bilgilerinin değişimi işleme (processing) zamanına göre sıralıdır. Değişim memo'sunu sağlayarak, bir nesne için birden fazla öğe bulunabilmektedir. Kullanılmayan öğeler tembelce arka planda (batch mode)'da silinirler. Kullanılmayan öğelerin ağaçta bulunma oranını sınırlayan çöp toplayıcı bulunmaktadır ve değişim memosunun büyüklüğü kontrol edilebilmektedir (Chen, S.vd. 2008). RUM-tree, R-tree'yi genişlettiği için zamanın doğrusal fonksiyonları ile nokta yerlerini endeksler. (Xiong, X.ve Aref, W. G. 2006).

B^x-tree yapısı, özel bir veri dönüştürümü kullanarak doğrusal fonksiyonlara eşleştirir. Ayrıca, B^+-tree endeks yapısını (Jensen, C. S.vd. 2004) kullanır. Böylece dört-boyutlu uzaydaki noktalar, tek-boyutlu uzaya dönüştürülürler. Bunun için ilk olarak, yakın zaman bilgisi argüman olarak alınarak, doğrusal fonksiyona eşleştirilir. Daha sonra space-filling eğrisi (Peano veya Hilbert eğrisi) uygulanarak, tek boyutlu nokta elde edilir. B^+-tree bilgisi değiştirilen noktaların aynı zamanda olanları aynı bölümde (partition) saklanırlar. Sorgulamaları sağlamak için orijinal sorgular, veri dönüşümlerinden dolayı dönüştürülmek zorundadırlar. Bu dönüşümler, endekslenen nesnelerin hızlarına window genişletilmesine, bölümleme sayısına, B^+-tree'deki her bölüme çevrilmiş sorguların sorgu zamanına bağlıdır. Hızın sapmasının ve aykırı değer almasının zararlı etkilerini azaltmak için B^x-tree, dikkatli olarak ayarlanmış sorgu genişletme algoritmaları teklif edilmiştir (Jensen, C. S.vd. 2006).

B^{dual}-tree, hız bilgisini en etkin olarak sorgu performansını artırmak için teklif edilmiştir. B^{dual}-tree, uzay ve hız vektörlerini tek-boyutlu noktaya indirgemek için dört-boyutlu Hilbert eğrisi kullanır. Bir B^{dual}-tree, iki tane B^+-tree yapısından oluşur ve bu iki ağaç en yüksek değişim aralığı (herhangi bir nesne tarafından verilen değişim çiftleri arasındaki en fazla zaman) olarak adlandırılan durumlar arasında geçiş yapar. B^{dual}-tree'deki her giriş, hareket eden nesne dikdörtgenler (MORs) kümesi ile ilişkilendirilmiştir. Her MOR, kare büyüklüğündedir ve alt ağaçlarda eşleştirilen sürekli Hilbert değerleri saklanır. MOR'lar, TPR-tree'deki BR'ler (sınırlayan dikdörtgenler) gibi düşünülebilir. Böylece TPR-tree'deki algoritmalar B^{dual}-tree'ye küçük değişiklikler yapılarak eklenebilir. Fakat, bu sorgu algoritması B^+-tree temelli olmadığı için varolan veri tabanı sistemlerine kolaylıkla entegre edilememektedir (Yiu, M. L.vd. 2008).

STRIPES da B^{dual}-tree gibi uzay ve hız verisini dual endeksler. STRIPES, iki boyutlu hareket eden nesneleri, dört-boyutlu noktalara çevirir ve PR bucket quad-tree ile endeksler. Bu şekilde düzenleme, etkin değişimleri ve daha az etkin sorguları destekler. Bunun nedeni de, STRIPES yapısında her düğümde az sayıda giriş olması, bundan dolayı da sorgu sonucunu oluşturmak için çok fazla sayfaya erişilmek zorunda kalınmasıdır. Quad-tree temelli endeks kullanır (Jensen, C.vd. 2006). Düşük sayfa kullanımı da çok fazla bellek alanı tüketmektedir. Tüm bu problemlerin etkisini azaltmak için, yaprak düğümlerin sayfaların %50'sini işgal etmesi ve bir sayfanın %50'sinin dolu olması tavsiye edilmektedir. Fakat, bu da eşzamanlılık kontrolü mekanizmasını karmaşıklaştırarak DBMS'e entegre edilmesini zorlaştırmaktadır (Patel, J. M.vd. 2004).

2.1.3.1 Parametrik uzay erişim metotları

Orijinal zaman-uzayı parametrik dikdörtgenlerle endekslenmektedir. En küçük sınırlayan dikdörtgenler zamana bağlı fonksiyonlarla gösterilerek, kapsanan hareket eden nesnenin aynı dikdörtgen içinde yer alması sağlanmaktadır. Bu durumda, herhangi bir t zamanında, endeks yapısı hesaplanabilir ve sorgu için değerlendirilebilir. Bu şekilde geliştirilen erişim yöntemleri; TPR-tree, TPR^{*}-tree, PR-tree, NSI, VCI R-tree, STAR-tree, R^{EXP}-tree ve TPR*-tree'dir.

TPR-tree (Time Parameterized R-tree)'de parametrik dikdörtgenler R-tree'de bulunur. TPR-tree yapısı, doğrusal zaman fonksiyonuna göre hızları endeksleyen R^{*}-tree yapısına eklenmiştir. Bir nesne, uzayına referans veren zaman ve hız vektörü ile gösterilmektedir. Endeksin oluşturulmasında, tutucu olarak hareket eden nesneyi sınırlayan dikdörtgenlerle oluşturulur. Tutucu dikdörtgenin alt sınırı içerdiği noktaların en düşük hızla hareket etmesi ve üst sınırı da en yüksek hızla hareket etmesi durumuna göre ayarlanır. Bu durumda tutucu dikdörtgen ne daralır ve de daima hareketli nesneleri içerir. Sınırlayan dikdörtgenlerin çok büyük olmasını engellemek için o nesnesi konumunu değiştirdiğinde, tüm sınırlayan dikdörtgende saklanan düğümler tekrar hesaplanır. TPR-tree'de değişimler olmadığında, sınırlayan dikdörtgenler genişleyeceği için sorgu ve değişim performansı düşer (Saltenis, S.vd. 2000).

TPR^{*}-tree yapısı, TPR-tree ile aynı veri yapısını kullanır fakat, endeks yapısını yönetme için farklı veri yapıları kullanır. TPR-tree yapısında time-slice sorgusunu en iyileştirme yerine zaman-aralık sorgularını en iyiler (Tao, Y.vd. 2003).

PR-tree, TPR-tree'ye benzemektedir. Fakat her parametrik dikdörtgen hareketin başlangıç ve bitiş zamanlarını da tutar. TPR-tree'de nesnelerin sonsuza kadar hareket edeceği düşünülmesine rağmen, PR-tree'de hareketin ne zaman sonlanacağı bilgisi de tutulur. Böylece hareket eden nesne gezinge yerine poligon olarak temsil edilebilir (Cai, M.ve Revesz, P. 2000).

NSI-tree, TPR-tree'ye benzemektedir. Fakat, sınırlayan dikdörtgenler iki yöntemde de farklı olarak tanımlanmaktadır (Porkaew, K.vd. 2001).

VCI R-tree (Velocity Constrained Indexing)'de R-tree düğümüne her nesnenin yapabileceği en yüksek hız v_{max} eklenir. Herhangi t sorgu zamanında, sınırlayan dikdörtgenler v_{max} kadar genişletilir. Bu endeks yapısında nesnenin gerçek hızını bilmeye gerek yoktur. Bunun yerine nesneler belirli en yüksek hız değerini aşamazlar. TPR-tree'de ise nesnelerin gerçek hızlarına gerek duyulmaktadır (Prabhakar, S.vd. 2002).

STAR-tree (Spatio-temporal Self Adjusting R-tree) yapısı, TPR-tree'ye benzemektedir. Bu yapıya ek olarak kendini-ayarlama bilgisi eklenmiştir. Sorgu performansı düşer düşmez, STAR-tree kullanıcıdan herhangi bir bilgi almadan kendisini ayarlar (Procopiuc, C. M.vd. 2002).

R^{EXP}-tree yapısı, hareket eden nesnelerin yaşlanma zamanları ile uğraşmak için TPR-tree'nin geliştirilmiş halidir. Burada oluşan en önemli sorun, hareketli nesnelerin uzun süre hareketlerini değiştirmemesidir. Yaşlanma zamanı bilgisinden faydalanarak, R^{EXP}-tree yeni bir tip sınırlama alanlarını tanımlamaktadır. Bunlara ek olarak R^{EXP}-tree, sınırlayan dikdörtgenlerin tekrar hesaplanmasına kadar yaşlanan nesneleri tembelce endeksten siler (Saltenis, S.ve Jensen, C. S. 2002).

TPR^{*}-tree yapısı, TPR-tree yapısını ve varsayımlarını kullanır. TPR-tree'den farklı olarak, farklı ekleme ve silme algoritmaları kullanmaktadır (Tao, Y.vd. 2003).

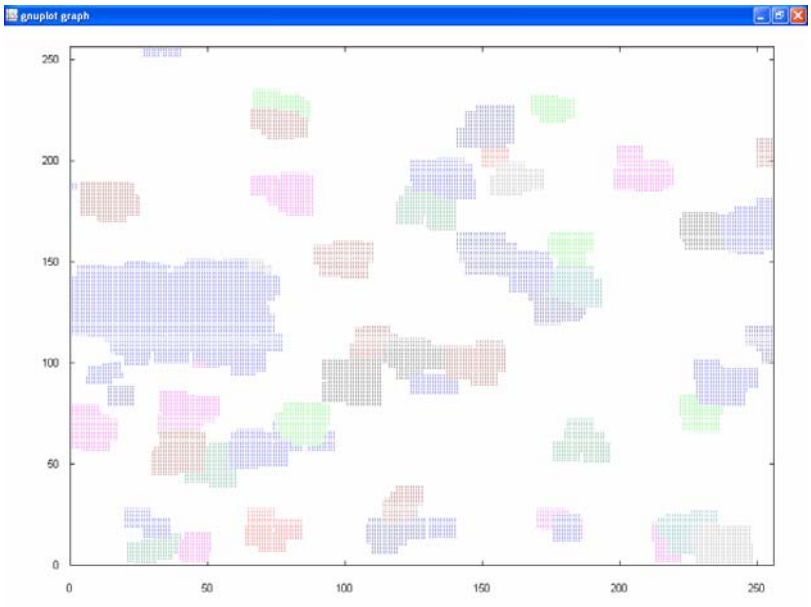
2.2 Hem Uzaya Hem Zamana Bağlı Veri Üreteçleri

Pratikte veri tabanı sistemlerinde, en iyi veri kümeleri gerçek verilerdir. Fakat, hareket eden nesnelere gerçek veriler sağlandığında, çeşitli sorunlar oluşmaktadır. İlk olarak, gerçek veriyi yakalama için ayrı bir çalışma gerekmektedir. Örneğin 1000 veya daha fazla aracın aynı zamanda izlenmesi. Bu araçların hepsinde de GPS alıcısı bulunmalıdır. Diğer bir problem de gerçek zamanlı veri ölçeklenebilir (scalable) değildir. Başka bir ifadeyle eldeki örnek,

geçmişe bakılarak genişletilememektedir. Tüm bu problemlerle uğraşmak yerine, veri üreteçleri ile veriyi oluşturmak en iyi çözüm yoludur. Bu üreteçlerin çıktıları çeşitli parametrelere değer vererek değişmektedir. Bazı veri üreteçleri, nesneleri benzetimli senaryolarda izleyerek oluşturmaktadır. Bu yöntem, gerçekçi senaryolar için iyi temsili veriyi üretmeyi vaat ederler.

Hareket eden nesneler ilk veri üreteçlerinden biri Generate_Spatio_Temporal_Data (GSTD)'dir (Theodoridis, Y.vd. 1999). Bu üreteç, kısıtsız hareket eden nokta veya dikdörtgen verisini yaratmaktadır. Bu çalışmaya iyileştirme olarak, daha gerçekçi gezingede daha öbeklenmiş, gezgin ve engelli hareket sağlayan çalışma da gerçekleştirilmiştir (Pfoser, D.ve Theodoridis, Y. 2003). Çalışmada iyileştirilmiş algoritma diğer veri üreteçleri ile de kullanılmıştır, örneğin cep telefonları için hücresel ağda konumlama verisini yaratma (Giannotti, F.vd. 2005). Gerçek dünyada, nesneler önceden tanımlanmış bir ağda hareket ettikleri için erişim yöntemleri de bu gerçekten etkilenirler. Dolayısıyla, GSTD'de üretilen verinin, hareket eden nesneler verilerinin bir ağda hareket etmeleri durumunda performans ölçümlerinde kullanılması uygun değildir.

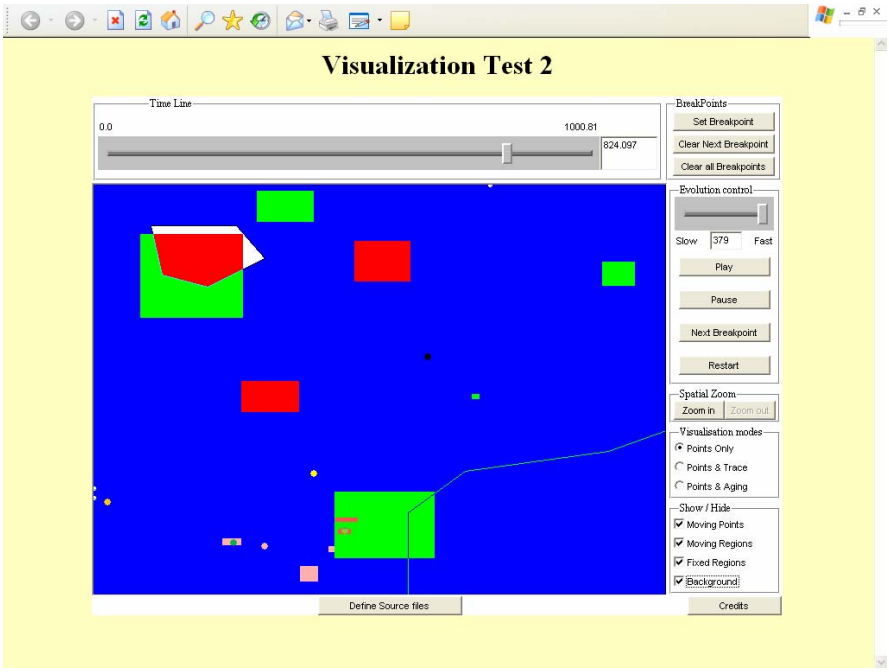
G-TERD (Generator of Time-Evolving Data) kavramsal sistemi (framework), sürekli olarak değişen ve büyük miktarda iki boyutlu bölge (region) verisi oluşturmaktadır (Tzouramanis, T.vd. 2002). Bölgelerin konumu, şekli, büyüklüğü ve değişen özellikler için rengi vardır. En yüksek hızı, dönme açısı, nesne etkileşimleri, başlangıç noktası ve ekrandaki gözlemci için hareketler, istatistik dağılımlar gibi parametreleri, kullanıcı tanımlayabilmektedir. G-TERD çıktısı, Şekil 2.5'te verilmiştir.



Şekil 2.5 G-TERD üreteci çıktısı

GTERD, çıktı şekli raster resimlerdir. Nesneleri izlemek için üreteç, iyi oluşturulmuş verileri üretmektedir. Bunlara rağmen, nesnelerin hareketi sınırsız ve stotastiksel olduğu için bir yol ağı üzerinde araçları benzetilmemek imkansızdır (Düntgen, C.vd. 2008).

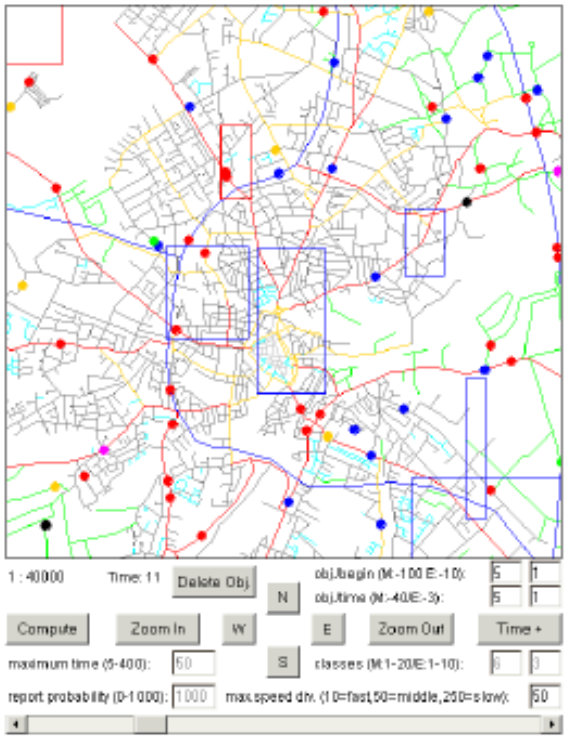
Diğer bir hem uzaya hem zamana bağlı veri üreteci Oporto'dur (Saglio, J. M.ve Moreira, J. 2001). Bu üreteç, balık tutma senaryosunu benzeterek (simüle) ölçeklenebilir hareketli nesneleri yaratmaktadır. Balık sürüleri dalgalanan noktalardır. Gemiler, balık sürülerini bulmak için rüzgarlı alanlardan kaçınarak limanlara ve limanların dışına doğru hareket ederler. Oporto, kısıtsız hareket eden nokta ve alan verisini oluşturur. Gemileri ve balık sürülerini uzun zaman gözlemleme mümkündür. Nesnelerin hareketleri hemen hemen kısıtsız olduğu için, bu üreteçten bir ağda hareket eden nesneler için uygun veri elde edilememektedir (Düntgen, C.vd. 2008). Şekil 2.6'da Oporto grafik arayüzü verilmiştir.



Şekil 2.6 Oporto veri üreteci grafik arayüzü

Bir ağda hareket eden nesneler için bir üreteç ve görsel arayüz bir diğer araştırmada teklif edilmiştir (Brinkhoff, T. 2002). Bir çok bilimsel çalışmanın kullandığı bu ortamda bir ağ üzerinde bulunan yüzlerce nesne modellenmiştir. Sistem geliştirmeye açıktır. Almanya'nın Oldenburg şehri üzerinde bu ortam oluşturulmuştur (Şekil 2.7). Bu üreteçte nesneler başlangıç noktasından, varış noktasına kadar hareket etmektedir. Nesne, hedefine vardıktan sonra gözden kaybolmaktadır. Hız ve hareket eden nesnenin izlediği yol, ağda bulunan kenarların yoğunluğuna (o anda yolu kullanan araçların sayısına bağlı olarak) göre değişmektedir. Bu nesneler harici (external) nesneler olarak adlandırılmaktadır. Ağ; shape, tiger line ve diğer

formatta bulunan dosyalardan yaratılabilmektedir. Üretecin davranışı, çeşitli parametrelerle kontrol edilebilmektedir. Nesne başlangıç noktasından, hedefe vardığından uzun süreli gözlemler mümkün olmamaktadır (Brinkhoff, T. 2003).

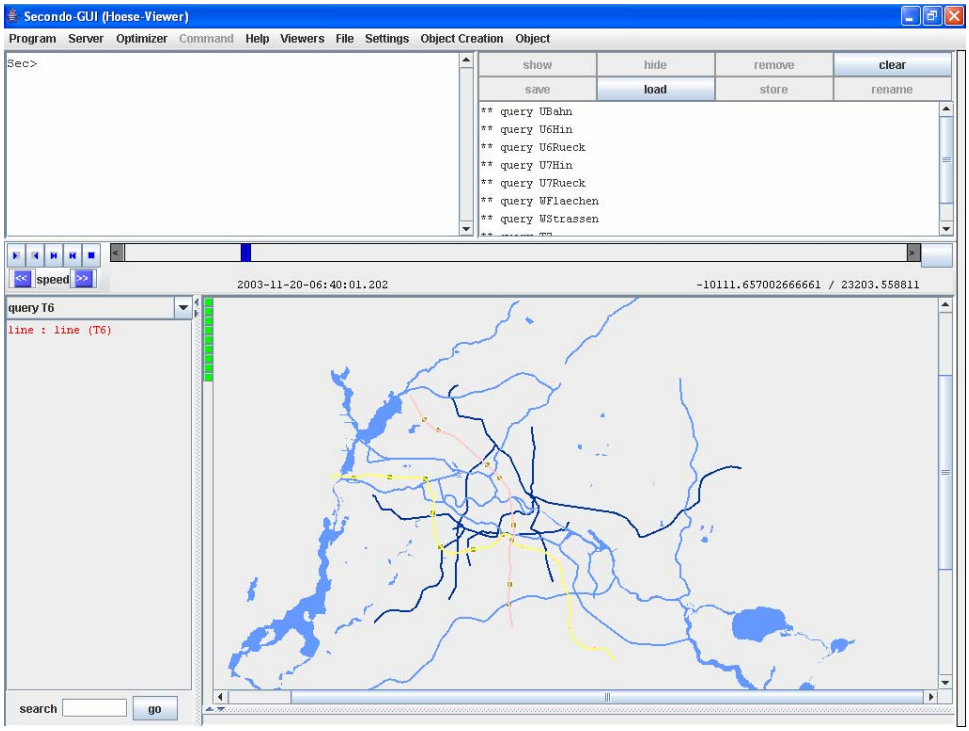


Şekil 2.7 Brinkhoff'un trafik üreteç grafik arayüzü

Bir diğer mikroskobik trafik üreteci de SUMO (Simulation of Urban MObilitiy)'dur (Krajzewicz, D.vd. 2002). Kesikli zamanda, sürekli-uzayda bu üreteç özel ve halka ait ulaşım verilerini üretmektedir. Araçlar çarpışma olmadan hareket ettirilmektedir. Çeşitli parametrelerle kullanıcı, seyahat planının başlangıç ve bitiş noktalarını ancak dikkatlice belirleyerek oluşturabilmektedir.

Bir ağda hareket eden nokta verilerini üreten bir diğer veri üreteci de BerlinMOD'dur. Bazı sorgu kümeleri için nokta verisi üreten bir araç da kullanıcılara sunmaktadır. Veri kümelerini yaratmak için, belirli zaman (ay) aralığında her iki saniyede bir nesnelerin bulundukları yer bilgileri elde edilerek Berlin yol ağında arabaların benzetimlemesi ile oluşturulmaktadır. Bunlara ek olarak BerlinMOD, nesnelerin uzun zamanda gözlemlenerek hareket eden ağ-tabanlı nokta verileri ile benzetimlenerek ilk ölçeklenebilir veri üretecidir. Uzun zamanlı gözlemleme, hareket eden nesneler için büyük ölçekte tarihçe verisinin oluşmasını sağlamaktadır. Bu da ilginç endeksleme ve hem uzaya hem zamana bağlı işlemlerin gerçekleştirilmesi için deneme ortamı oluşturmaktadır (Almeida, V. T. D. 2006). BerlinMOD SECONDO sisteminin içinde değerlendirilmiştir (Guting, R. H.vd. 2005; Guting, R. H.vd.

2007a; Guting, R. H.vd. 2007b). Sorguların ve verinin yaratılma değerlendirmeleri, hareket eden nesneler konusunda en son geliştirilen tekniği göstermektedir (Düntgen, C.vd. 2008). Secondo sisteminde (Almeida, V. T. d.vd. 2007) çalışan BerlinMOD simülasyonu Şekil 2.8'de verilmiştir.



Şekil 2.8 SECONDO Java grafik arayüzü

Bunlara ek olarak, BerlinMOD seyahat tabanlı olarak ve belirli zaman aralığında verileri oluşturabilmektedir. Bu üreteçte yolun yoğunluğu dikkate alınmamaktadır. Nesneler için yolda durak noktaları, yavaşlama ve hızlanma noktaları yolun bölümlerinde geçiş noktaları belirlenebilmektedir (Düntgen, C.vd. 2008).

2.3 Değerlendirme

Bu bölümde hareket eden nesneler için gerçekleştirilen araştırmalarda oluşturulmuş endeks yapıları incelenmiştir. Genel olarak değerlendirildiğinde; gelecek, mevcut ve geçmiş zamanları içeren endeks yapıları ayrı ayrı oluşturulmuştur. Tüm zamanları (gelecek, mevcut ve geçmiş) içeren endeks yapıları da az sayıda araştırmada gerçekleştirilmiştir (Sun, J.vd. 2004; Lin, D.vd. 2005; Pelanis, M.vd. 2006).

Sun vd. gerçekleştirdiği çalışma, her nesnenin detaylı hareketini tutmak yerine çok boyutlu histogramda arttırarak mevcut zamanda sorguları cevaplamaktadır. Bellekte saklanan endeks

yapıları ile yakın geçmiş zamana ait hareketler sorgulanabilmektedir. Endeksin en eski bölümü sabit diske bloklar şeklinde yazılmaktadır. Bu şekilde G/Ç işlemleri azaltılmaktadır. Gelecek zamana ait sorgular ise, yakın geçmiş zamandaki hareketler kullanılarak gelecek stokastik metotlarla tahmin edilmektedir. Burada nesnelerin hızı konusunda herhangi bir varsayım yapılmamaktadır.

Lin vd. yaptıkları çalışmada, BB^x endeks yapısı ile hareket eden nesnelerin uzayları doğrusal zaman fonksiyonuna göre B^+-tree'ler ile oluşturulmaktadır. Bu endeks ile zamana ve uzaya bağlı sınırlamalara göre sorgular gerçekleştirilebilmektedir.

Pelanis vd. çalışmasında ise, endeks kısmi olarak kalıcıdır. Diğer bir ifadeyle endeksin bir bölümü sabit diskte bir bölümü de bellekte saklanmaktadır. Hem birim zaman hem de işlemin gerçekleştiği zaman verilerine göre işlemleri gerçekleştirmektedir.

Endeks yapılarının bütünleştirilebildiği bir kütüphane oluşturma çalışması ile uzaya bağlı endeksler seçilerek istenen uygulamada kullanılabilmektedir (Hadjieleftheriou, M.vd. 2005). Kullanıcının da tanımlayabildiği hem uzaya hem zamana bağlı endeksler ile yapı genişletilebilmektedir.

Hareketli nesneler için gerçekleştirilmiş veri üreteçleri de bu bölümde incelenmiştir. Grafik arayüzü verilen üreteçler, tez çalışması sırasında bilgisayara kurularak denenmiştir. Bahsedilen üreteçler arasında, Brinkhoff'un geliştirdiği üreteç (Brinkhoff, T. 2003), istenen sayısal haritanın kolaylıkla yüklenip hareketli nesnelere ait verileri oluşturabildiği için tez çalışması kapsamında kullanılmıştır.

3. HAREKET EDEN NESNELER İÇİN KAVRAMSAL VERİ MODELLEME

Kavramsal veri modellerinde, gerçek dünya ile bilgisayar sistemlerinde temsil edilen gösterim arasında tutarlılık olmalıdır. Kavramsal modellerin gücü, kavramlarının sayısı ve bunlar arasındaki ilişkiler yaratabilme yeteneği ile ifade edilmektedir. Veri modeli kullanıcıları, oluşturulan kavramsal modelin çok karmaşık olmamasını ve kolay kullanılmasını talep etmektedir. Bu kavram, basitlik ve anlamlılık gücü **ortogonallik** (orthogonality) olarak isimlendirilir. Basitliğin sağlanabilmesi için çok boyutlu problemlerin çözümünde problemin farklı parçalara bölünmesi ve her bir parçanın birbirinden bağımsız olarak çözülmesi gerekir. Bu nedenlerle oluşturulan kavramların görsel gösterimi kullanıcılar tarafından kolayca anlaşılabilmelidir. Veri modelleme; veri yapıları, uzay, zaman gösterimi gibi çok boyutlu problem olarak görülmektedir.

Kavramsal veri işleme dilinin, sorgulama ve değiştirme işlemleri için uygunluğu önemlidir. Böylece, kullanıcıların tüm veri tabanı etkileşimlerinin tek şekilde yapmasına olanak sağlanır. Fakat, günümüzde durum belirtilenden daha farklıdır; kullanıcılar varlık-ilişki veya UML şema tanımlarına bağlı olarak düşünmekte ve veri işlemek için ilişkisel SQL'e göre konuşmaktadır.

Hem uzaya hem zamana bağlı veri tabanı modelleri için bunlara ek olarak birkaç gereksinim daha tanımlanmalıdır. Uzay modelleme düşünüldüğünde **kesikli** ve **sürekli** uzay için kavramların birbiriyle uyumlu şekilde entegre edilmesi gerekmektedir. Bu iki görünüm de uygulamalar için önemlidir, aynı anda ikisi bir arada uygulamalarda kullanılabilmektedir. Zaman modelleme düşünüldüğünde de **işlem** ve **geçerlilik zamanlarının** tanımlanması gerekmektedir.

Hem uzaya hem zamana bağlı veriyi modellemek için, geleneksel veri tabanı sistemlerinde varlık ilişki şemaları (Entity-Relationship) veya nesneye yönelik programlarla oluşturulan sistemlerde UML (Unified Modeling Language) şemaları bu tip veri için geliştirilerek kullanılmıştır. Bu bölümde, hareketli nesneler için gerçekleştirilmiş kavramsal veri modelleri incelenmiştir.

3.1 Varlık-İlişki Şemaları ile Hareketli Nesneleri Modelleme

Hem uzaya hem zamana bağlı veriyi kavramsal olarak modellemek için, varlık ilişki (entity relationship) şemalarını genişleterek kullanan iki modelleme çalışması araştırmalar sırasında elde edilmiştir: SpatioTemporal Entity-Relationship (STER) (Tryfona, N.vd. 1999) ve

SpatioTemporal eXtended Entity-Relational model (STXER) (Jin, P.vd. 2008).

SpatioTemporal Entity-Relationship (STER) modelleme aracı ile herhangi bir kavramsal model uzaya bağlı veri tabanlarındaki (Oracle) veri modeline kolaylıkla dönüştürülebilmektedir. Bu araçta bulunan CASTER (Computer-Aided SpatioTemporal Entity Relationship) ile kavramsal modelleme gerçekleştirilmekte ve CASTR (Computer-Aided SpatioTemporal Relational) ile de mantıksal tasarım yapılmaktadır.

SpatioTemporal eXtended Entity-Relational modeli (STXER) bir CASE aracıdır. Hareket eden nesneler için, tipik nesne-ilişkisel veri tabanı yönetim sistemlerinde veri tabanı modellemeyi gerçekleştirmek için kullanılır. Kavramsal varlık-ilişki şemalarını genişleterek kullanmaktadır. Hem uzaya hem zamana bağlı olmayan veriler ile hem uzaya hem zamana bağlı verileri desteklemektedir. Hareket eden nesnelerin; sürekli, kesikli, sürekli tematik, kesikli tematik, kesikli nesne ve kesikli topolojik değişiklikleri modelde gösterebilmektedir. Bu CASE aracı; grafik modelleme aracı, XML şema editörü ve SQL Script üreteci ile desteklenmektedir. Java programlama dili ile geliştirilmiştir.

3.2 UML Şemaları ile Hareketli Nesneleri Modelleme

UML şemaları geliştirilerek, hareketli nesnelerin modellenmesi Modeling of Application Data with Spatio-temporal features (MADS) modeli ile gerçekleştirilmektedir. Aşağıda Parent ve arkadaşları tarafından geliştirilen MADS modeli verilmiştir.

3.2.1 MADS modeli

MADS modelleme yapıları; nesne tipleri, özellikleri, metotları, ilişki tipleri, çeşitli gösterimler, kısıtlar olarak sınıflandırılmıştır. MADS, çoklu algılama ve çoklu gösterimlere olanak sağlamaktadır. Ayrıca, veri işleme ve sorgulama dili ile ilişkilendirilmiştir (Parent, C.vd. 2006).

MADS modeli elemanları, uzay ve zamana bağlı veri tiplerinin tanımlanmasına olanak sağlamaktadır. Bu veri tipleri aşağıda verilmiştir.

3.2.1.1 Veri tipleri

Hareket eden nesneler için ayrı ayrı uzaya ve zamana bağlı veri tipleri tanımlanabilmektedir. Aşağıda bu veri tipleri sunulmuştur.

3.2.1.1.1 Uzay veri tipleri

Nesneye yönelik yaklaşımdan dolayı, haritalardaki veriler belirli bir hiyerarşiye uygun olarak gruplanır. Haritadaki yerler, diğer bir ifade ile uzaya ait bilgiler aynı tipteki veya farklı tipteki veri grupları ile gösterilebilir. Örneğin şehirler, noktalar grubu ile gösterilebilir. Çizelge 3.1'de MADS uzay veri tipleri verilmiştir. Aynı tipteki veri tipleri, bag ile biten tanımlayıcılarla tanımlanabilmektedir (Parent, C.vd. 2006).

Çizelge 3.1 Uzay veri tipleri

Uzay Veri Tipleri	Sembol	Uzay Veri Tipleri	Sembol
Geo		SimpleGeo	
Point		Line	
OrientedLine		Surface	
SimpleSurface		ComplexGeo	
PointBag		LineBag	
OrientedLineBag		SurfaceBag	
SimpleSurfaceBag			

3.2.1.1.2 Zaman veri tipleri

Benzer olarak, Çizelge 3.2'de MADS ile tanımlanan zaman veri tipleri gösterilmiştir. Belirli bir an, aralık veya bunların gruplanmasından zaman veri tipleri oluşturulabilir. Nesnelerin etkin olduğu zamanı gösterebilmek için, geçerli zaman aralığı MADS ile belirtilmektedir. Ayrıca, instance'ların durumu da scheduled, active, disabled ve suspended durumlarından biri ile ifade edilir (Parent, C.vd. 2006).

Çizelge 3.2 Zaman veri tipleri

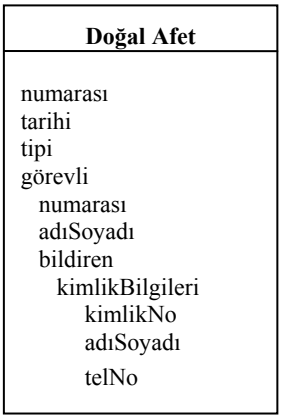
Zaman Veri Tipleri	Sembol	Zaman Veri Tipleri	Sembol
Time		SimpleTime	
Instant		Interval	
ComplexTime		InnstantBag	
IntervalBag		TimeSpan	

Uzay ve zamana bağlı olarak, bu özellikler sürekli veya ayrık (discrete) olarak tanımlanabilir. Sürekli olan özellikler fonksiyonlarla gösterilmektedir.

3.2.1.2 Nesneler

MADS modelinde kavramlar, nesneler veya o tipin örneği (instance) olarak isimlendirilir. Aynı özelliklere sahip sınıf örneklerini birbirinden ayırmak için sistem, her nesneye bir nesne tanımlayıcısı (oid) atamaktadır.

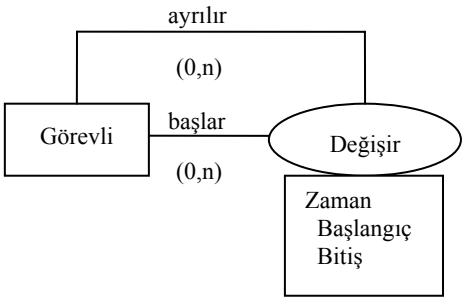
Bir nesnenin özellikleri basit ve karmaşık olabilir. Karmaşık özellikler, birden fazla basit özelliğin bir arada kullanılması ile elde edilmektedir. Örneğin yangın veya deprem gibi bir doğal afeti bildirenin bilgilerini saklayan bir nesne yapısı Şekil 3.1'de verilmiştir. Buna göre numarası, tarihi ve tipi basit özellikler; görevli ve bildiren de karmaşık özellikler olarak tanımlanmaktadır. Şekil 3.1'de bildiren ve kimlikBilgileri karmaşık özelliklerdir (Parent, C.vd. 2006).



Şekil 3.1 Basit ve karmaşık özelliklerden oluşan bir nesne şeması

3.2.1.3 İlişkiler

Veri tabanındaki nesneler modellendikten sonra, bunlar arasındaki ilişkiler belirlenmektedir. Her ilişkinin özellik sayısı (cardinality) en küçük ve en büyük sayı şeklinde iki sayı ile ifade edilir. Örneğin (0,n) sayıları; özelliğin seçimlik olduğunu, diğer bir ifade ile özelliğin hiç veya en fazla n tane olabileceğini, (1,n) sayıları da; özelliğin zorunlu olduğunu ve en az bir, en çok n tane olduğunu göstermektedir. Dairesel ilişkileri tanımlama imkanı sunan MADS ile Şekil 3.2'deki gibi bir gösterim gerçekleştirilebilir. Buna göre doğal afet merkezinde bulunan ve belirli saatler arasında görevli tarafından sisteme kayıt girilebilmektedir. Dairesel ilişkiler, mutlaka bir rol ile ilişkilendirilmelidir (Parent, C.vd. 2006).



Şekil 3.2 Bir dairesel ilişki

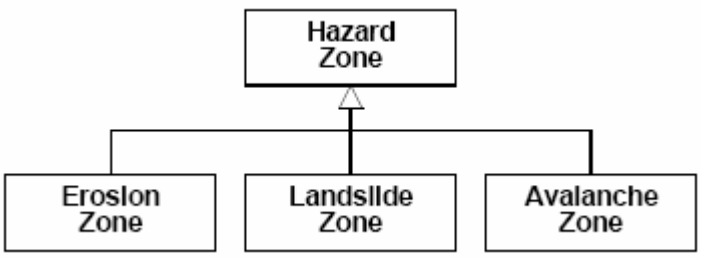
İlişkiler, gerçek dünyadaki bağlantıları göstermektedir. MADS ilişkiler, n'lidir (n≥2) ve iki çeşittir; **bağımlılık** (association) ve **çok-bağımlılık** (multi-association). Bağımlılıklar da ikili olarak tanımlanmalıdır. Her ilişkinin bir numarası (rid) vardır. Nesne tipleri gibi ilişki tipleri de özelliklerle tanımlanmaktadır.

Bazı durumlarda, haritanın ölçeğinden dolayı küçük ölçekte görülen beş bina, büyük ölçekte görülen üç bina ile eşleştirilebilmektedir. Bu durumda, çok-bağımlılık ilişkisi, iç içe iki elips ile gösterilmektedir (Şekil 3.3).

Şekil 3.3 Bir çok-bağımlılık ilişkisi

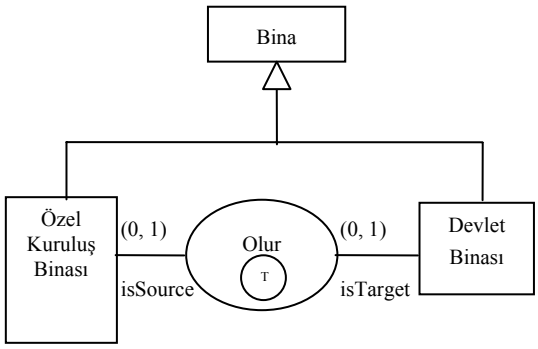
Bunlara ek olarak model, nesnelere dayandırıldığı için genelleştirme/özelliştirme ilişkilerini sağlayan, is-a bağlantısı, çoklu kalıtım, yeniden tanımlama, gruplama (aggregation) ilişkileri de MADS ile tanımlanmaktadır.

Is-a bağlantısı genelleştirme/özelleştime ilişkisidir ve nesne sınıfına özgü (generic) tipi süper tip (supertype) ve alt tipi (subtype) ilişkilendirir (Şekil 3.4). Is-a ilişkisi kalıtımsallığı gösterir. Bu da yerine kullanımı sağlar (Parent, C.vd. 2006).



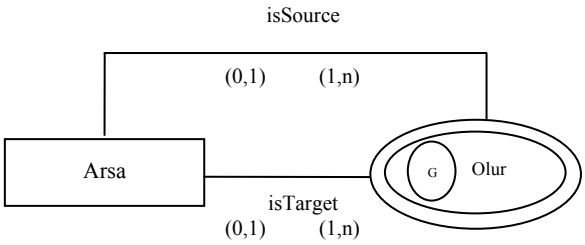
Şekil 3.4 Is-a ilişkisi

MADS modelinin zaman içinde kolaylıkla değiştirilebilmesi için çeşitli anlamları olan iki yapı (semantics), **geçiş** ve **yaratma** yapıları eklenmiştir. Örneğin, bir ülkenin bir ilçesinin zamanla gelişmesi sonucu, ülke yönetimi ilçeyi şehir olarak tanımlayabilir. Bu değişiklik modele, geçiş yapısı ile aktarılabilir. Böylece zamanla değişen roller model tarafından saklanabilir. Şekil 3.5'te özel bir kuruluş binasının, devlet binası olma durumuna geçiş durumu gösterilmiştir. (0,1) sayıları ile, devlet binasına dönüşen özel kuruluş binasının mutlaka önceden olması gerektiği belirtilmiştir. T(transition) ile geçiş yapısı gösterilmektedir. Geçiş yapısı, ikili ilişkilerdir. Bunlar isSource ve isTarget ile tanımlanmışlardır. Bu ilişki aynı kimliğe sahip çalışan sınıflar arasında ilişki kurar.



Şekil 3.5 Geçiş yapısı

Yaratma yapısı ile varolan bir sınıf, zamanla iki sınıfa dönüşebilmektedir. Örneğin bir arsa, mirasçılar arasında paylaşıldıktan sonra daha küçük arsalara bölünmektedir. isSource ile yaratıcı rolü, isTarget ile yaratılan rolü gösterilmektedir. Şekil 3.6'da bu durum, dairesel çok-bağımlılık ile gösterilmektedir. G(generation) ile yaratma yapısı gösterilmektedir.



Şekil 3.6 Yaratma yapısı

3.2.1.4 Uzay-zaman modelleme

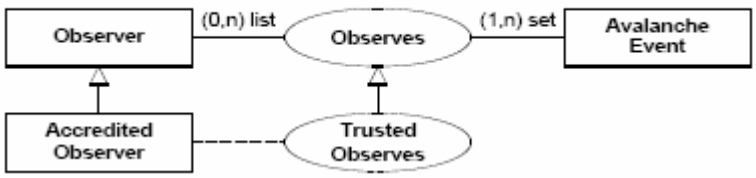
Karmaşık geometrik şekillerden oluşan bir şehir nesne modeli, Şekil 3.7'de verilmiştir. Buna göre, rakım konuma ve nüfus da zamana göre değişmektedir. Ayrıca havadurumu hem uzaya hem de zamana bağlı olarak değişmektedir.

Şekil 3.7 Hem uzaya hem zamana bağlı gösterim

Bunlara ek olarak MADS modeline, nesnelerin uzaya ve zamana ait topolojik ilişkileri eklenerek model tamamlanmaktadır. Zamana ait topolojik ilişkiler, senkronizasyon gösterimleri olarak tanımlanmaktadır. Nesneler uzay topolojik ilişkileri olarak; birbirinden ayrı, çakışıyor, içinde, yaratıyor, değiyor, kesiyor ve eşit olarak uzayda görülebilir (Wolfson, O.vd. 1999; Mokbel, M. F. 2004). Zaman topolojik ilişkileri olarak da önce, esnasında, aynı anda başlıyor, eşit, biri biterken diğeri başlıyor, kesişiyor, aynı anda bitiyor ve ayrı tanımlanmıştır.

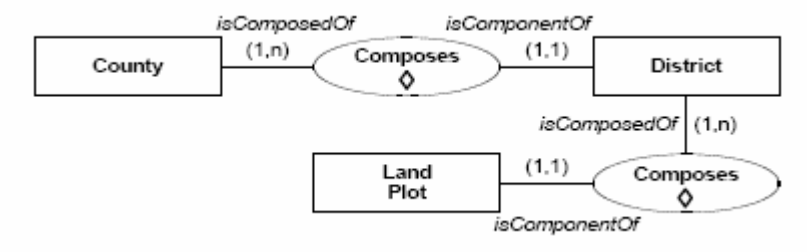
MADS modelinin çok kullanıcı tarafından kullanılması için çoklu algılama ve gösterimde farklı görünümlere bir stamp (pul) atanır. Örneğin bir veri üç farklı şekilde algılanıyorsa, bu s1, s2 ve s3 ile gösterilmektedir. Son olarak, her özelliğin hangi pul tarafından algılandığı modele eklenmiştir (Parent, C.vd. 2006).

Bazı uygulamalarda standart kalıtımsallık ilişkisinin saptırılması gerekir. Ayrıca bazen supertype ve subtype'da kalıtsal özellik aynı kalabilir veya farklı olabilir (Şekil 3.8). Yeniden tanımlama (redefinition), dinamik ilişkilendirmeyi (binding) sağlar. Overloading ise bu durumu rahatlatır.



Şekil 3.8 Nesne tipinden bir rolü ayrıştıran subtype ilişkisi

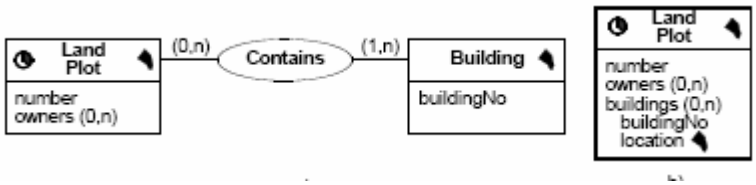
Aggregation ilişkisinde nesne diğer nesnelerin tamamından oluşmaktadır (Şekil 3.9). MADS isComponentOf ve isComposedOf tanımlarıyla bu ilişkiyi belirtir. Aggregation ikili ilişkidir ve dairesel olabilir.



Şekil 3.9 Aggregation ilişkisi

3.2.1.5 Hem uzaya hem zamana bağlı veri yapıları

MADS ile uzaya ve zamana bağlı olarak ilgilenilen bir olayı belirli veri tipleri kullanarak tanımlamak kesikli görünüm veya nesne görünümü (object view) olarak adlandırılmaktadır. Kesikli görünümde bölgeler (regions) veya zaman aralıkları otonom varlıklar olarak yer alır veya boştur. MADS'ta uzay ve zaman tanımları veri yapılarına göre ortogonaldir. Bu da bir olayın, veri yapıları ne olursa olsun uzay özelliklerine genişletilebileceğini ifade eder. Şekil 3.10'da hem uzaya hem zamana bağlı gösterim verilmiştir.



Şekil 3.10 Hem uzaya hem zamana bağlı gösterim

Nesneler veya ilişkiler için zamana bağlı işlemlere iki farklı anlam yüklenebilir. İşlem (transaction) ile bilginin ne zaman veri tabanında saklandığı veya veri tabanından silindiği ifade edilir. Buna karşın geçerli zaman ile uygulamaya göre bir olayın ne zaman geçerli olduğu verisi tutulmaktadır. MADS geçerli zaman bilgisini uygulamalara sağlamaktadır.

Zamana bağlı nesne veya ilişki tipleri için kullanıcılara bu nesnelerin (instance) yaşam döngüsü bilgisi verilir. Yaşam döngüsünde dört durum bulunmaktadır: scheduled, active, suspended ve disabled. Şekil 3.11'de zamana bağlı within ilişkisi gösterilmiştir.

Şekil 3.11 Bir senkronizasyon ilişkisi

3.2.1.6 İlişkileri uzay ve zaman yüklemleri ile kısıtlama

Hareket eden nesneler birbirlerinin konumuna göre, kendi konumlarını topolojik ilişkilerle göstermektedirler. Çizelge 3.3'te uzaya bağlı topolojik ilişkiler verilmiştir.

Çizelge 3.3 Uzaya bağlı topolojik ilişkiler

Topological kind	Icon	Topological kind	Icon
TopoDisjoint		TopoTouch	
TopoOverlap		TopoCross	
TopoWithin		TopoEqual	
TopoGeneric			

Aynı şekilde hareketli nesnelerin, zaman boyutunda birbirlerine göre zaman ilişkileri Çizelge 3.4'te verilmiştir.

Çizelge 3.4 Zamana bağlı senkronizasyon ilişkileri

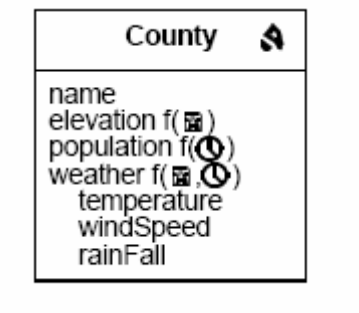
Synchronization kind	Icon	Synchronization kind	Icon
SyncPrecede		SyncMeet	
SyncWithin		SyncOverlap	
SyncStart		SyncFinish	
SyncEqual		SyncDisjoint	
SyncGeneric			

Uzaya ve zamana bağlı bir topolojik ilişki örneği Şekil 3.12'de verilmiştir.

Şekil 3.12 Hem uzaya hem zamana bağlı topolojik ilişki

Kesikli görünümün dışında uzayın ve zamanın sürekli olarak görünümü de uygulamalarda önemlidir. Bu nedenle de nesnelerin bir özelliği uzayda değişiyorsa uzay-değişken özellik olarak adlandırılır. Buna arazinin yüksekliği örnek olarak verilebilir. Bazı özellikler hem

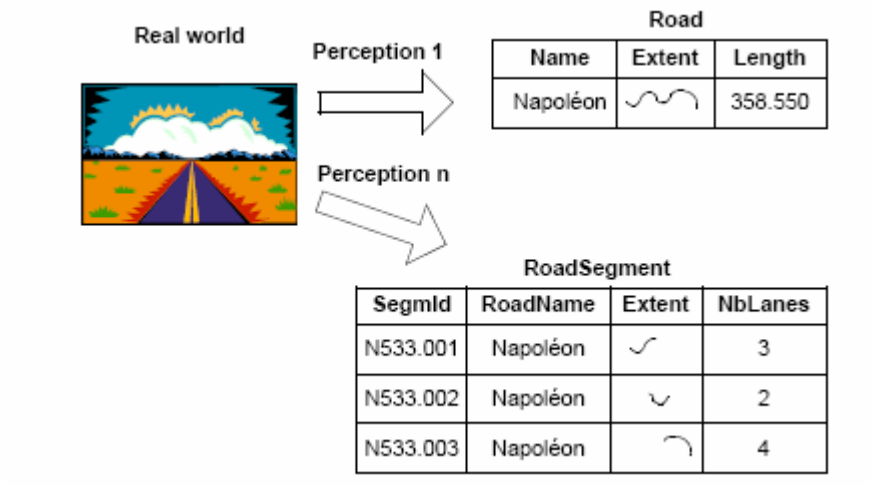
zaman-değişken hem de uzay-değişken olabilir. Şekil 3.13'te değişken özelliklere bir örnek verilmiştir.



Şekil 3.13 Sürekli görünüme bir örnek

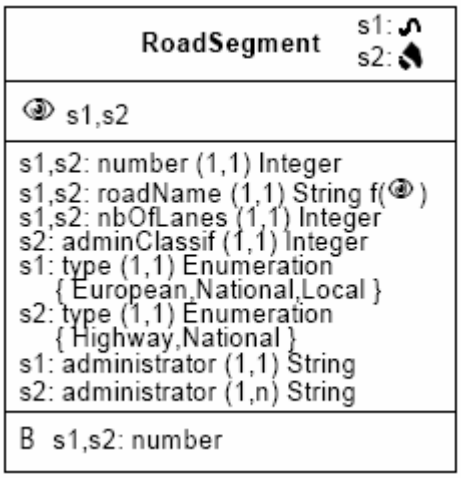
3.2.1.7 Çoklu-algılama ve çoklu-gösterim

Coğrafi sistemlerde çoklu-gösterimler kullanılmaktadır. Bunlardan biri çözünürlüktür. Bir çok sistem farkı çözünürlüklerde oluşturulmuştur. Ayrıca farklı algılamalar da veri tabanlarında saklanabilmelidir. MADS bunu algılama pulları (stamps) ile sağlamaktadır. Şekil 3.14'te farklı algılamaların veri tabanında nasıl saklandığı gösterilmektedir.



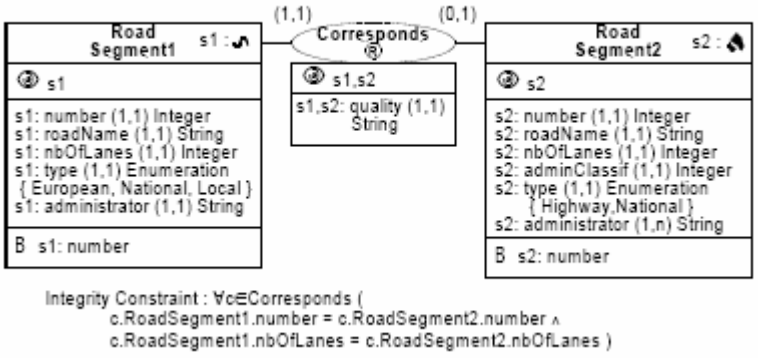
Şekil 3.14 Farklı algılamaların veri tabanında saklanması

Farklı algılamaların MADS ile gösterimi Şekil 3.15'te verilmiştir. Çoklu algılamalar ve çoklu gösterimler için her özelliğin önüne s1, s2, … sn şeklinde pullar yerleştirilerek gerçekleştirilir. Şekil 3.15'te roadName özelliği, farklı algılamalarda değişmektedir. Bu da f( ) ile gösterilmiştir.



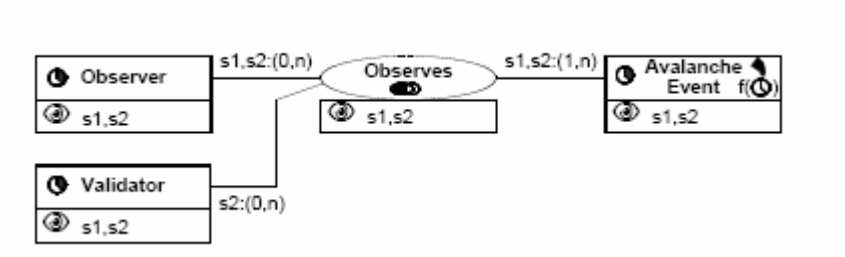
Şekil 3.15 Çoklu-algılama MADS gösterimi

Şekil 3.15'teki çoklu-algılama gösterimini tek bir algılayan için tekrar çizilirse Şekil 3.16 elde edilir.



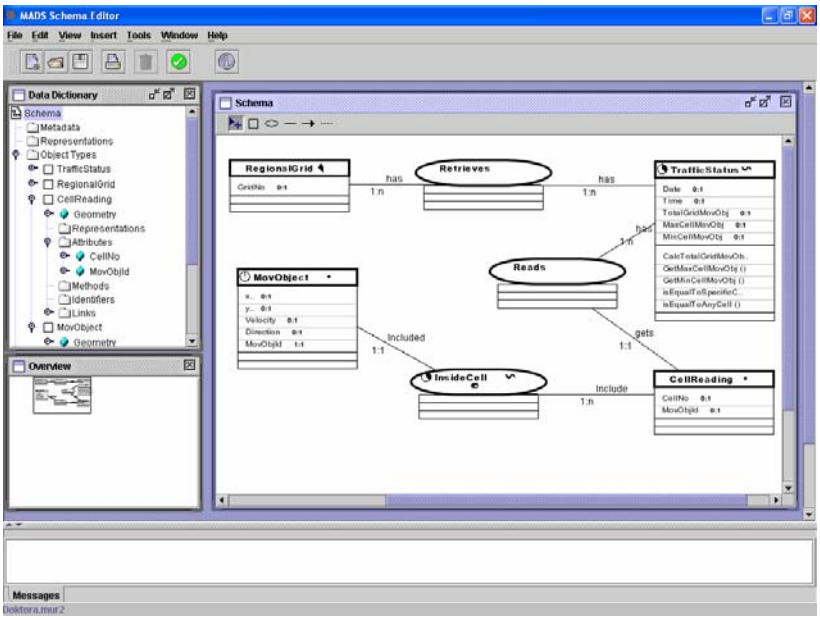
Şekil 3.16 Tek-algılamalı MADS gösterimi

İlişkiler de farklı olarak algılanabilir (Şekil 3.17).



Şekil 3.17 İlişkilerin farklı olarak algılanması

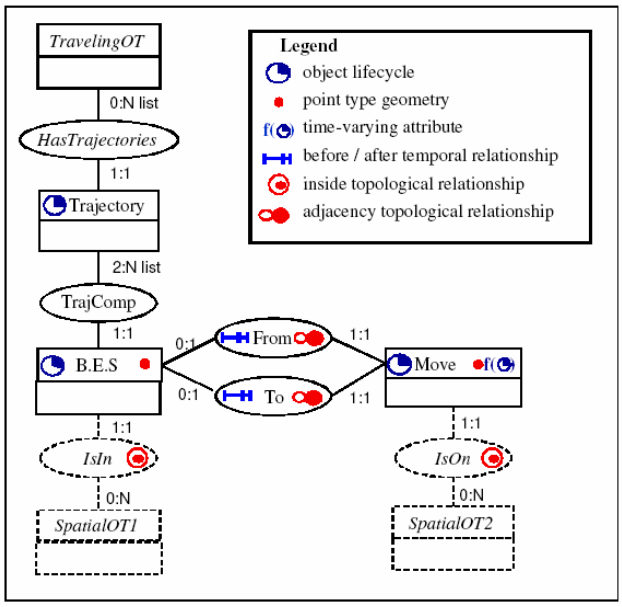
MADS kavramsal modelinin oluşturulması için Şekil 3.18'de gösterilen Murmur Schema Editörü kullanılmaktadır.



Şekil 3.18 Murmur schema editor

3.2.1.8 Gezinge modelleme

Gezingeler ile bir yerden diğer bir yere hareket eden nesnenin izlediği yol tanımlanır. Bu bilgiler modele hareket eden nesnenin başlangıç, bitiş ve durak yerleri belirtilerek aktarılır (B.E.S.-Begin/End/Stop). *TrajComp* ilişkisi, parçalarla gezingeleri birleştirir. Gezinge modelleme, Şekil 3.19'da verilmiştir (Spaccapietra, S.vd. 2008).



Şekil 3.19 Gezinge için teklif edilen model

3.3 Değerlendirme

Hareket eden nesneler için kavramsal modeller varlık-ilişki ve UML şemaları geliştirilerek oluşturulmuştur. Hem uzaya hem zamana bağlı kavramları daha iyi modellediği, uzun süredir uygulandığı ve nesneye yönelik tasarımlarla gerçekleştirildiğinden dolayı tez çalışmasında, MADS kavramsal modeli kullanılmıştır. Bu araç, CASE araç özellikleri desteklenerek açık kaynak olarak sunulmamaktadır. Bu nedenle oluşturulan sistemlerin, veri tabanı modelindeki karşılıkları ayrıca tanımlanmaktadır.

Bu bölümde ayrıca, gezingeler için geliştirilen kavramsal model incelenmiştir. Gezingelerin kavramsal modellemesi konusu, yeni bir araştırma konusudur.

4. HAREKET EDEN NESNELER İÇİN SORGU DİLİ

Bu bölümde hareket eden nesneler için tanımlanmış sorgu tipleri, zamana bağlı kavramlar, zamana bağlı kavramların veri tabanlarında kullanımı, uzaya bağlı kavramlar, hem uzaya hem zamana bağlı kavramlar araştırılmıştır. Ayrıca literatürde tanımlanmış soyut model üzerinde, sorgu dilleri tanımlama çalışmaları da incelenmiştir.

4.1 Hareket Eden Nesneler için Sorgu Tipleri

Hem uzaya hem zamana bağlı veri tabanlarında veri, tanımı gereği çok-boyutlu olduğu için çok çeşitli hem uzaya hem zamana bağlı sorgular tanımlanabilir. Tanımlanan bir çok hem uzaya hem zamana bağlı sorgu, endeksleme veya veri işleme ihtiyaçlarından dolayı oluşmuştur. Uzaya bağlı sorgular için; nokta sorgu, aralık (range) sorgu, boolean sorgu, select-by-location, select-by-attribute, spatial join, ve query-by-sketch tanımlanmıştır. Zamana bağlı sorgular için; snapshot, timeslice, atttribute-history, key-history, interval intersection, time-range ve bitemporal aralıklar tanımlanmıştır (Ladner, R.ve K.S. 2002).

Özellik (attribute) sorgular, nesne veya bölge içinde tanımlanan yer ile ilgili özellik bilgisine dayanmaktadır. Bir çok durumda bu tip sorgular, belirli kriterleri sağlayan veri kayıtlarını okuyarak cevaplanmaktadır. Çoğu, boolean işlemlerle (and, or veya not) gerçekleştirilebilir. Örneğin alan veya çevre uzaya veya zamana bağlı olarak düşünülmemektedir (Ladner, R.ve K.S. 2002).

Uzaya bağlı sorgu tipleri üç şekilde olabilir:

- Basit uzaya bağlı sorgu,
- Uzaya bağlı aralık sorgusu,
- Uzaya bağlı ilişki (relationship) sorgusu.

Basit uzaya bağlı sorgu, seçimin belirli kriterlere göre verilerin çekilerek oluşturulmasından dolayı geometrik hesaplama gerektirmemektedir. Uzaya bağlı aralık sorgularında, belirlenmiş bir alanın özellikleri hakkında veri araştırılmaktadır. Örneğin bir parkın alanının, sınırları ile belirlenmesi. Uzaya bağlı ilişki sorguları, nesnelerin veya olayların uzayda nasıl ilişkilendirildiği ile ilgilenmektedir. Uzaya bağlı ilişkiler, yaklaşıklaştırma (proximity) ve topoloji sınıflarına ayrılabilir. Topoloji sorguları genellikleştirilerek sekiz adet topoloji: disjoint, contains, inside, equal, meet, covers, covered by ve overlap belirlenmiştir. Bu topolojiler, tüm ticari yazılımlarda kullanılmaktadır (Ladner, R.ve K.S. 2002).

Zamana bağlı sorgu tipleri de üç grupta incelenebilir;

- Basit zaman sorguları,
- Zaman aralık sorguları,
- Zaman ilişki sorguları.

Zamana bağlı sorgulara **geçerli zaman** ve **birim-işlem** zaman kavramları eklenmiştir. Geçmişteki olayları da veri tabanında saklayabilmek için bu kavramlar kullanılmaktadır. Basit zaman sorguları, herhangi bir özelliğin durumu hakkında bilgiyi verilen zaman için araştırmaktadır (snapshot). Özellik sorgudan farklı olarak, basit zaman sorgusu istenen zaman için veri kayıtlarını içerebilir veya içermeyebilir. Sorgu sonucunda hiç bir veri kaydının seçilmemesi durumunda zamana bağlı düşünme ve interpolasyon işlemlerinin yapılması gerekir. Örneğin veriler saat başlarında kaydedilebilir ama sorgu yarım saatte bir verilebilir. Zaman aralık sorguları verilen bir aralıkta bir özelliğin araştırılmasıdır. Rüzgarın bir saat içinde ne kadar hızlı hareket ettiği bu sorgu tipine örnek olarak verilebilir. Zaman ilişki sorgularında, zamanda birçok nesnenin nasıl ilişkilendiği sorusuna cevap araştırılır. Zamana bağlı yaklaşıklaştırma ve topoloji işlemleri gerçekleştirilir. Zamana bağlı topolojik ilişkiler; önce, sonra, sırasında, içinde, kesişme, o nesnesi ile kesişme, karşılaşma, m ile karşılaştı, başlama, s ile başlama, bitme, f ile bitme ve eşit. Bu sorguya örnek olarak, en son yağmur zamanlarında olan belirli rüzgarlarla kaç tane toprak parçasında kayma oldu sorusu verilebilir (Ladner, R.ve K.S. 2002).

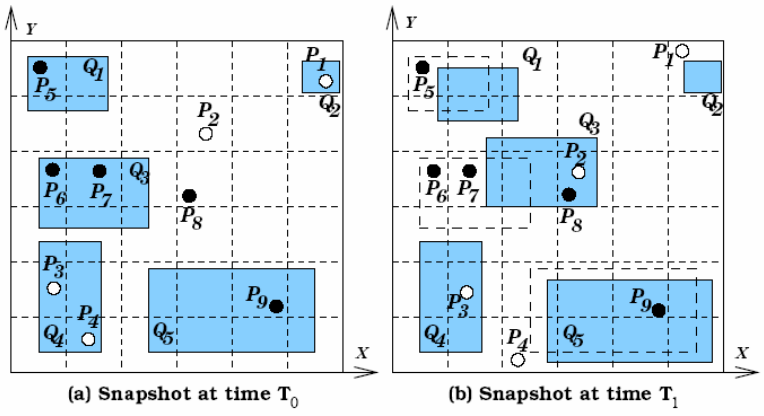
Hem uzaya hem zamana bağlı sorgular, nesnelerin veya olayların veya verilen zaman aralığında olanların yaşam-sürecidir (lifespan). Ayrıca hem uzaya hem zamana bağlı sorgular, olaylar ve işlemler arasındaki gelişmeleri, dinamikleri ve etkileşimleri araştırır. Bunları yaparken de ilişkileri tahmin etmek ve modellemek için özetlemeye çalışır. Hem uzaya hem zamana bağlı olaylar birbirinden ayrı davranışlar gösterir. Örneğin trafik kazaları hem uzaya hem zamana bağlı bir olaydır. Yollar üzerinde (davranışlar) bir veya birden fazla araç arasında (nesneler) ve yaralanmalara ve trafik desteklerine neden olabilir (phenomena). Hem uzaya hem zamana bağlı sorgular için dört tip sorgu belirlenmiştir:

- Basit hem uzaya hem zamana bağlı sorgu,
- Hem uzaya hem zamana bağlı aralık sorgusu,
- Hem uzaya hem zamana bağlı davranış sorgusu,
- Hem uzaya hem zamana bağlı ilişki sorgusu.

Basit hem uzaya hem zamana bağlı sorgu, ilgilenilen nesnelerin verilen zamanda nerede olduğunu, ilgilenilen nesnelerin verilen yerde ne zaman olacağını veya verilen yerde ve

zamanda neyin olduğu sorularını sorar. Bu sorgu, seçim ve boolean işlemler ile gerçekleştirilebilir. Fakat, hiçbir kayıt sorguya cevap olarak dönmezse, uzaya veya zamana bağlı interpolasyonun gerçekleştirilmesi gerekir. Sürekli olarak bir yol üzerinde hareket için, yer bilgisi; hız ve yön bilgisine bağlı olarak elde edilebilir.

Hem uzaya hem zamana bağlı aralık sorguları, bir bölgeye (region), belirli bir zaman aralığında ne olduğu sorusunu sorar (Mokbel, M. F. 2004; Mokbel, M. F.vd. 2005). Bu sorgularda geometride ve topolojide değişiklikler olur (Şekil 4.1).



Şekil 4.1 Aralık (range) sorgusu

Hem uzaya hem zamana bağlı davranış sorgusu ile, nesnelerin veya olayların veya işlemlerin zamanda ve uzayda nasıl değiştiği (yer, büyüklük, uzay parçaları, olayların sıklığı, örüntülerin hareketi ve yoğunluk dağılımı) ile ilgilenilir. Burada amaç, hem uzaya hem zamana bağlı bir varlığın özelliklerini ve özelliklerin yaşam-süreci boyunca nasıl değiştiğini araştırmak ve değişime neden olan içsel mekanizmaları ortaya koymaktır. Örüntülerin dağılımında ve yapının uzayda ve zamanda değişimi söz konusudur.

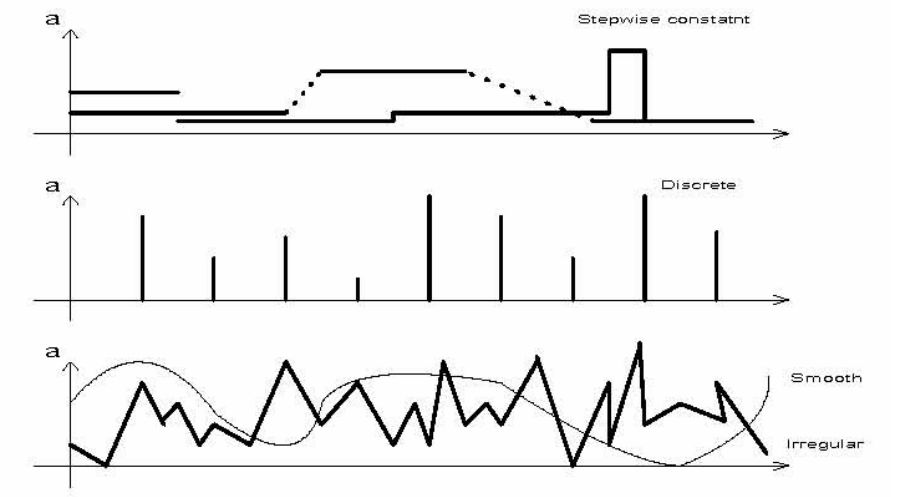
Hem uzaya hem zamana bağlı davranış sorgularına cevaplar, uzaya ve zamana bağlı karakteristikleri ve örüntüleri özetler. Hem uzaya hem zamana bağlı ilişki sorguları, birçok türdeki varlığın uzayda ve zamanda birbiri ile nasıl ilişkilendiğini araştırır. Böylece, hem uzaya hem zamana bağlı davranış sorgusuna bir katman daha eklenir. Çünkü, hem uzaya hem zamana bağlı davranış sorguları tek bir tiple ilgilenmekteydi. Teorik olarak, hem uzaya hem zamana bağlı ilişkiler iki bakış açısı ile elde edilebilir; uzaya bağlı ilişkilerin zamana bağlı olarak değişimi ve uzay üzerinde zamanın değişimi. Nerede ve ne zaman bir bölge ciddi bir açlık tehlikesini El Nino yılında çekecek sorusu bu sorguya örnek olarak verilebilir. Bu sorgu en karmaşık ve dinamik olanıdır (Ladner, R.ve K.S. 2002).

4.2 Zamana Bağlı Kavramlar

Zaman öğelerinin büyüklüğü (granularity), bölümleme uzunluğu ile ifade edilir. Her uygulamada zaman öğelerinin büyüklüğü farklıdır.

Zamana bağlı işlemler (temporal operations) ile zamanla ilişkili işlemlerin neler olduğu belirtilir. Buna örnek olarak, bir zaman noktasında A zaman aralığının içinde olma (inside) işlemi verilebilir.

Zaman yoğunluğu (time density) ile zamanın kesikli (discrete-tamsayılarla eşyapılı-isomorphic) veya sürekli (continuous-reel sayılarla eşyapılı) olarak tanımlanıp tanımlanmayacağı ifade edilir (Şekil 4.2). Kararlı özelliklerin ani olaylara maruz kalmasından dolayı stepwise sabit değerler oluşur. Sürekli olarak değişen özellikler, örüntülerinin değişmesine bağlı olarak iki alt gruba bölünebilir: tek tip (uniform) ve düzensiz (irregular). Varlıklar sürekli olarak zaman içinde durumlarını değiştirirken, bazı varlıklar da sabit olarak kalabilirler, diğer bir ifade ile durumlarında bir değişiklik olmayabilir (Pelekis, N.vd. 2004).



Şekil 4.2 Zaman yoğunluğu tipleri

Zamanın gösterimi (representation of time) her modelde farklıdır. Bir zaman modeli timestamp'ler ile gösterilir. Bu kriter ile nesnenin durumunun süresini elde etme veya durumundaki değişikliğin kaydedilmesi her modelde karşılaştırılabilir.

Birim işlem zamanı/Geçerli zaman (transaction/valid time) ile uzaya bağlı değişiklikler modele yansıtılır. Birim işlem zamanı ile durum değişikliğinin veri tabanına kaydedildiği zaman belirtilir. Geçerli zaman ile tanımlanan, olayın gerçek dünyada ne zaman olduğudur. Bir hem uzaya hem zamana bağlı model, bu iki zamanı (bitemporal) da saklar.

Zaman sırası (time order) zaman perspektifini, ok (arrow), ilerleyen gösterim veya daire ile temsil eder (Pelekis, N.vd. 2004).

Yaşam süresi (lifespan) ile bir olayın süresi modelde gösterilir. Ayrıca modelde gerçek dünya nesnelerinin, kesikli oluşlar (phenomenon) veya sürekli zaman farklılıkları tarihçesi tutulur.

4.2.1 Zamana bağlı kavramların veri tabanlarında kullanımı

Standart DBMS ile yönetilen veri tabanlarında veriler o anki durumu yansıtmaktadır. Dış dünyada meydana gelen değişiklikler belirli bir zaman sonra güncelleştirilir böylelikle, bir önceki durum kaybedilir (Guting, R. H.ve Schneider, M. 2005). Bazı uygulamalar için o anki durum bilgilerini saklamak yeterli değildir; bir şekilde geçmiş bilgilerin de tutulması gerekmektedir (Artale, A.vd. 2007). Standart DBMS'lerde bu mümkündür; uygulama zamanı veri tabanı tarafından yönetildiğinde bazı veriler için ek zaman bilgileri tanımlanarak bu işlem gerçekleştirilebilir (Carvalho vd., 2006). Örneğin bir şirketin çalışan bilgilerinin aşağıdaki şekilde bir tabloda olduğunu varsayalım:

```
Çalışan (ad: string, bölüm:string, maaş:int)
```

Şirket, önceki bölümleri ve çalışanların önceki maaşlarını da takip etmek isterse, yukarıdaki tablo aşağıdaki gibi genişletilebilir:

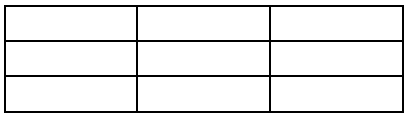
```
Çalışan (ad: string, bölüm:string, maaş:int, başlangıç:date, bitiş:date)
```

Fakat yukarıdaki şekilde zaman ile ilgili olan işlemleri gerçekleştirmek, uygulamayı karmaşıklaştırır, hata eğilimli olmasını sağlar.

Zaman, tek-boyutlu olarak geçmişten geleceğe modellenir. Burada bazı opsiyonlar bulunmaktadır: zaman uzayı **sınırlı** veya **sınırsız** olabilir. Sınırlı modele göre zaman boyutunun bir başlangıcı ve bitişi olmalıdır.

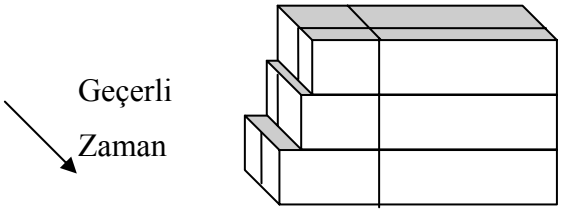
Zaman anlamsal olarak iki tiptir: **geçerli zaman** ve **birim-işlem** (transaction) zamanı. Geçerli-zaman ile kastedilen gerçekte olayın olmasını veya gerçeğin hala geçerli olduğudur. Birim-işlem zamanı ile veri tabanına değişimin kaydedildiği zaman veya veri tabanının herhangi bir durumunun belirli zaman aralığında varolduğu belirtilir. Bu bağlamda, standart veri tabanları **snapshot veri tabanları**; geçerli zaman ile ilgilenenler, **geçerli-zaman** veya **rollback veri tabanları**; iki kavramla da ilgili olanlar **bitemporal veri tabanları** olarak adlandırılır. **Zamana bağlı veri tabanları** kavramı ile herhangi bir zaman desteği olan sistemler veya modeller kast edilmektedir.

Şekil 4.3'te standart veri tabanlarının veya bir snapshot'ın ilişkisi gösterilmiştir (Guting, R. H.ve Schneider, M. 2005).



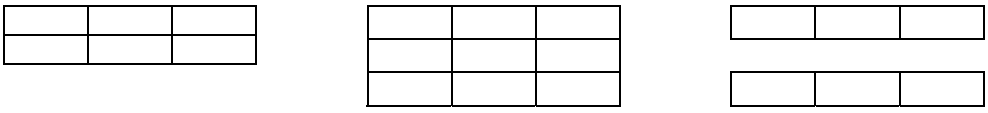
Şekil 4.3 Bir snapshot ilişkisi

Şekil 4.4'te ise bir geçerli-zaman ilişkisi verilmiştir. Üç veri de mevcut zamanda geçerliliğini korumaktadır.



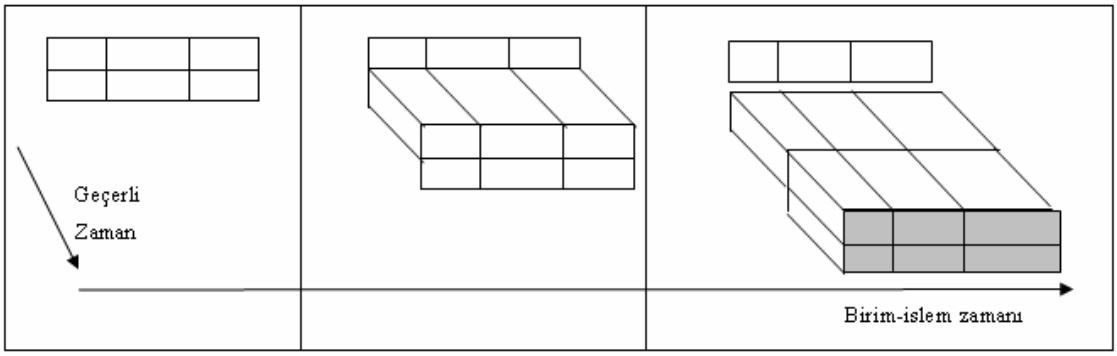
Şekil 4.4 Bir geçerli-zaman ilişkisi

Birim-işlem zaman ilişkisi, Şekil 4.5'te bir verilmiştir. Burada, birinci birim-işlem 2 ilişkiyi eklemiştir. İkinci birim-işlem de 3ncü satırı eklemiştir. Üçüncü birim-işlem 2nci satırı silmiştir.



Şekil 4.5 Birim-işlem zamanı

Şekil 4.6'da verilen birim-işlemlerde, ilk birim-işlem, iki satır veri ekler. İkinci birim-işlem, 2nci satırı değiştirir ve 3ncü satırı ekler. Üçüncü birim-işlem, 2nci satırı siler ve 3ncü satırı siler (gri renk, satırların geçerli olmadığını belirtmektedir.) Ayrıca, 2nci satırın başlangıç zamanını değiştirir. Bu durumda, 1nci satırdaki veri hala geçerlidir.



Şekil 4.6 Bir bitemporal ilişki

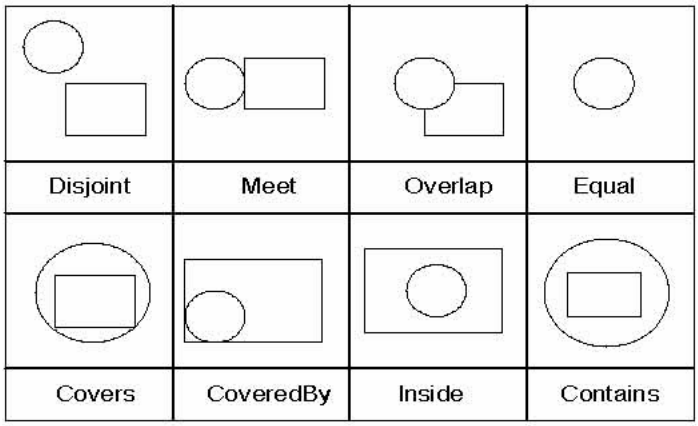
4.3 Uzaya Bağlı Kavramlar

Uzayın yapısı (structure of space), coğrafi verinin raster veya vektör veri modelleri ile saklanmasını ifade eder. Raster veri, hücre dizisi olarak yapısallaştırılmıştır. İki veya üç boyutlu veriler bu veri yapısı ile tutulur. Uzay grid'lere bölünmüştür. Vektör veri ise, uzaya ait veriyi başlangıç ve bitiş noktaları olarak saklar. Bu veri bilgisayarın veri saklama kapasitesini en iyi şekilde kullanır.

Yönelim/Yön (orientation/direction) nesnelerin yönelim ve yön bilgilerini modelin destekleyip desteklemediği belirtilir.

Ölçüm (measurement) ile bir uzaya bağlı nesnenin yer bilgisi değerinin veya modelin karşılaştırmalı işlemleri (daha büyük, daha küçük) desteklemesi ifade edilir.

Topoloji (topology) ile gerçek dünyadaki nesneler arasındaki ilişkiler gösterilir. Görsel olarak, topolojik ilişkiler Şekil 4.7'de verilmiştir (Pelekis, N.vd. 2004).



Şekil 4.7 Topolojik ilişkiler

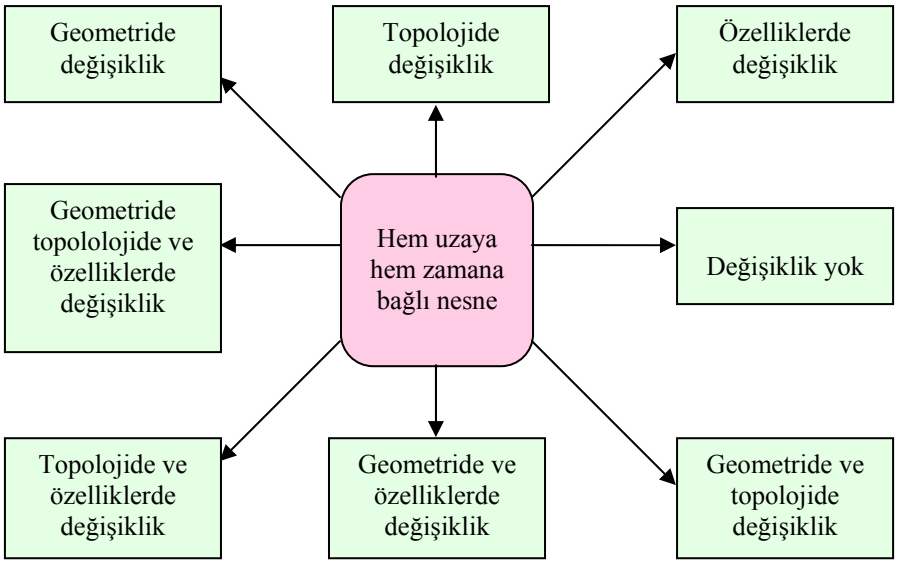
4.4 Hem Uzaya Hem Zamana Bağlı Kavramlar

Veri tipleri (data types) ile her modelde bulunan uzaya, zamana, hem uzaya hem zamana bağlı veri tipleri ifade edilir.Uzaya ait veri tipleri; nokta, çizgi ve bölgedir. Zamana ait veri tipleri ise; zamana ait nokta ve zaman aralığıdır. Hareket eden nokta ve hareket eden bölge de hem uzaya hem zamana bağlı veri tipleridir (Şekil 4.8).

Şekil 4.8 Veri tipleri

İlkel gösterimler (primitive notions) kriteri ile her modelde gerçek dünyanın nasıl soyutlaştırıldığı (abstraction) belirtilir. Her model, bilgi sistemlerindeki farklı yönleri ele alır.

Değişimin tipi (type of change) ile modellerdeki nesnelerin şekillerinde ve büyüklüklerinde oluşan değişiklikler tanımlanır. Olası değişiklikler Şekil 4.9'da verilmiştir.



Şekil 4.9 Hem uzaya hem zamana bağlı nesnede olası sekiz değişiklik

Zamanda ve uzayda gelişim (evolution in time & space) ile nesnelerdeki değişikliği modelleyebilen tanımlı fonksiyonlar ifade edilir. Bu terim ayrıca hareket eden nesnelerin hız ve ivmelerini hesaplamak için kullanılan fonksiyonları da tanımlar. Ayrıca yakın gelecek zaman için interpolasyon işlemleri de bu kavram başlığı altında incelenir.

Uzay-zaman topolojisi (space-time topology) ile hem uzaya hem zamana bağlı topolojik ilişkiler ifade edilir (Pelekis, N.vd. 2004).

Nesne kimlikleri (object identities) ile araştırılan hareket eden nesnenin durumunda değişiklik olduktan sonra, nesnenin bir önceki nesneden farklı olduğunun nasıl ifade edileceğidir. Bazı durumlarda nesneyi yok edip, yeniden yaratmak daha uygun olmaktadır. Fakat bu da çok geniş alanda değişikliğin yapılmasını gerekli kılmaktadır.

Boyutlandırma (dimensionality) ile coğrafi bilgi sisteminin (GIS) iki boyutlu modelleri desteklemesi tanımlanmaktadır (Pelekis, N.vd. 2004).

4.5 Sorgu Dilleri ve Soyut Model

Hareket eden nesneleri modellemek için ilk araştırmalarda soyut modeller ortaya konmuştur (An, K.vd. 2005). Bu çalışmalarda amaç, bilgisayar terminolojisinden bağımsız olarak böyle bir sistemde olması gerekenlerin tanımlanmasıdır. Model kesikli yapıdan, ziyade süreklidir. Uzay sonsuzdur. Bu nedenlerle de soyut modeller gerçekçi değildir, sadece kesikli modeller bilgisayarda kullanılabilir (Wang, X.vd. 2000).

Hareket eden nesne veri tabanları için veri tabanı tanımları, on adet non-predictive ve statik (Myllymaki, J.ve Kaufman, J. 2003) soyut model üzerinde SQL-tabanlı sorgularla Theodoridis ve arkadaşlarının çalışmasında verilmiştir (Theodoridis, Y. 2003). Bu sorgular için, deneyler eklenmemiştir.

Düntgen ve arkadaşları yakın zamanda BerlinMOD isminde bir benchmark teklif etmiştir. Bu çalışmada, hareket eden nesne verilerini yaratmak için SECONDO veri tabanı yönetim sistemi kullanılmıştır (Almeida, V. T. d.vd. 2006). Bir senaryo, Berlin yol ağında hareket eden nesneler için örneklenen uzaylardan alınan veriler hareketi modellemek için kullanılmaktadır. Toplam olarak seçilen 17 adet SQL-tabanlı sorgu üzerine çalışma yoğunlaşmıştır (Düntgen, C.vd. 2008). Bu benchmark geçmiş, hareket eden nesnelerin tarihçesini tutmakta ve hem uzaya hem zamana bağlı DBMS'leri tam olarak değerlendirmektedir (Chen, S.vd. 2008).

Hareket eden nesne veri tabanlarının önem kazanmasıyla, kesikli aralıklarda değişen geometriler üzerinde çalışılmıştır. Bu çalışmalarda, sürekli olarak hareket eden geometrileri desteklemek için veri modelleri ve sorgu dilleri üzerine yoğunlaşılmıştır. Soyut olarak zamana bağlı yerleri tanımlamak için *hareket eden noktalar (moving point)* ve zamana bağlı olarak şekilleri ve büyüklükleri değişen *hareket eden alan (moving region)* belirlenmiştir (Forlizzi, L.vd. 2000). Hareket eden noktalara arabalar, uçaklar, gemiler, mobil telefon kullanıcıları; hareket eden alanlara volkanlar, denize dökülen yağ kaçakları, orman yangınları, ordular

örnek olarak verilebilir (Lema, J. A. C.vd. 2001).

Tanımlanan bu tip veriyi modelleme ve sorgulama için hareket eden nokta ve alan olarak iki soyutlama (abstraction) ile DBMS veri modeline eklenecek uygun işlemlerle birlikte kullanılır. Bu işlemler, verilen bir zaman anında hareket eden alanı değerlendirme (sonuç bir alan olarak), hareket eden noktanın 2-boyutlu uzaya izdüşümü (sonuç bir 2-boyutlu eğri) veya hareket eden bir noktanın hareket eden bir alanın içinde olup olmadığını belirleme (sonuç zamana-bağlı bir Boolean değer) olabilir.

4.5.1 Veri tipleri

Bir tip bir sistem, imzalar ile tanımlanır. Bir imzanın genel olarak tipleri ve işlemleri vardır. Ayrıca, bir terim kümesini tanımlayabilir. İmza ile tanımlanan bazı veri tipleri *int*, *region*, *range(instant)* veya *moving(point)*'tir (Çizelge 4.1).

Çizelge 4.1 Soyut tipte bir sistemi tanımlama için imzalar

	→ BASE	*int*, *real*, *string*, *bool*
	→ SPATIAL	*point*, *points*, *line*, *region*
	→ TIME	*instant*
BASE ∪ TIME	→ RANGE	*range*
BASE ∪ SPATIAL	→ TEMPORAL	*intime*, *moving*

Sabit veri tipleri olan *int*, *real*, *string* ve *bool* genel olarak kullanıldığı gibi tanımlanır ve buna ek olarak 'undefined' değerini de içerecek şekilde genişletilmiştir. Bir nokta (point) veri tipi, iki boyutlu uzaydaki bir noktayı temsil eder. Bir çizgi (line) veri tipi, bir uzayda sürekli eğrilerin sonlu kümesidir. Bir alan (region) sonlu sayıda ayrık face (içi boş olmayan birbirine bağlı uzay alt kümesi)'lerdir. Her face'in delikleri olabilir ve diğer face'lerin delikleri içinde olabilir.

Bir *instant* tipi, bir zaman için seçilen öz niteliklerde reel sayılara göre eşyapılıdır. Bir range tipi, ilgilenilen zaman öz nitelikleri üzerinde birbiri ile ayrık sonlu sayıda ikili değerler üretir. Bir *intime* tipi, ilgilenilen zaman öznitelikleri üzerinde bir zaman anı ile ilişkilendirir.

En önemli veri tipi hareket eden (moving)'dir. Verilen BASE veya SPATIAL'de α argümanı moving(α)'yı oluşturur. Burada fonksiyon değerleri zaman boyutundan α boyutunadır. Fonksiyonlar kısmi olabilir ve sonlu sayıda sürekli bileşenden oluşmalıdır. Örneğin moving(region) zamandan region değerlerine bir fonksiyondur.

4.5.2 İşlemler

Tanımlanan veri tipleri üzerinde, soyut model çok sayıda işlemi teklif etmektedir. İlk adım olarak, zamana-bağlı olmayan tipler (*moving* veya *intime* tipleri hariç tüm tipler) üzerinde işlemler tanımlanmaktadır (Güting, R. H.vd. 2000). Bu işlemlerin predicate isimleri (örneğin inside veya ≤), küme işlemleri (örneğin, birleşim), grup (aggregate) işlemleri, sayısal sonucu olan işlemler (örneğin, bir alanın büyüklüğü) ve uzaklık ve yön işlemleridir (Çizelge 4.2).

Çizelge 4.2 Zamana-bağlı olmayan işlemler

Class	Operations
Predicates	**isempty** **=, ≠, intersects, inside** **<, ≤, ≥, >, before** **touches, attached,overlaps** **on_border,in_interior**
Set operations	**intersection, union, minus** **crossings** **touch_points, common_border**
Aggregation	**min, max, avg, center, single**
Numeric	**no_components, size, perimeter** **duration, length, area**
Distance and direction	**distance, direction**
Base type specific	**and, or, not**

İkinci adım olarak zamana bağlı yükseltme (temporal lifting) mekanizması ile zamana bağlı olmayan tipler üzerinde tanımlanan işlemler, tek tip ve tutarlı olarak zamana bağlı (hareket eden) tipler üzerine uygulanabilir. Örneğin inside işlemi, bir *point* ve *region* argümanıyla uygulanabilir ve sonuç olarak *bool* tipli veri gönderir. Üçüncü adım olarak zamana bağlı tipler üzerinde değerleri fonksiyon olan *moving(α)* ile özel işlemler teklif edilmektedir. Tüm bunların zaman ve aralık uzayına izdüşümü alınabilir (Çizelge 4.3).

Çizelge 4.3 Zamana bağlı işlemler

Class	Operations
Projection to domain/range	**deftime, rangevalues** **locations, trajectory** **routes, traversed, inst, val**
Interaction with domain/range	**atinstant, atperiods** **initial, final, present** **at, atmin, atmax, passes**
Rate of change	**derivative, speed** **turn, velocity**

4.5.3 DBMS'e eklenti ve sorgular

Bir örnekte bu veri tiplerinin DBMS veri modelindeki özellik (attribute) tiplerine nasıl ekleneceği ve sorgularda bu işlemlerin nasıl uygulanacağı gösterilecektir. Bunun için soyut veri tipleri, veri tabanına öncelikle eklenmelidir. Örneğin uçaklar tablosuna aşağıdaki ilişkiler eklenebilir.

Ucaklar (havayolu: *string*, uNo: *string*, uçus: *mpoint*)

Yukarıda *mpoint* ile tanımlanan *moving(point)*'tir. Ucus terimi, zamana bağlı olarak uçağın konumunu kaydeden hem uzaya hem zamana bağlı veriyi göstermektedir. Zor sorgular için bazı işlemlerin imzaları Çizelge 4.4'te verilmiştir.

Çizelge 4.4 Bazı örnek işlemler

Operation	Signature	
trajectory	*moving*(*point*)	→ *line*
length	*line*	→ *real*
distance	*moving*(*point*) ×*moving*(*point*)	→ *moving*(*real*)
atmin	*moving*(*real*)	→ *moving*(*real*)
initial	*moving*(*real*)	→ *intime*(*real*)
val	*intime*(*real*)	→ *real*

Hareket eden noktanın uzaya izdüşümü (projection) noktalar ve çizgiler olabilir. Trajectory işlemi, böyle bir izdüşümün çizgi parçalarını hesaplar. Length işlemi çizginin uzunluk değerini belirler. İki hareket eden nokta arasındaki uzaklık, distance işlemi ile hesaplanır. Atmin işlemi, aynı en küçük *real* değere hareket eden real değerleri tüm zamanlarda sınırlar. İlk (*instant*, *real*) hareket eden reel sayılar, initial işlemi ile başlatılır. Val işlemi, (*instant*, *real*) değerlerine uygulanır ve ikinci elemana izdüşümü alınır.

Yukarıda tanımlanan işlemlerle "5000 km'den daha uzun Türk Havayolları uçuşlarını bulma" sorgusu projection işlemi ile aşağıdaki gibi oluşturulabilir:

```
SELECT havayolu, uNo
FROM Ucaklar
WHERE havayolu='Türk Havayolları'
        AND length(trajectory(ucus)) > 5000
```

İzdüşümler (projection) yapılarak oluşturulan bir diğer sorgu da "Uçuşları sırasında 500m birbirlerine yakın gelen iki uçağı bulma"dır:

```
SELECT u.havayolu, u.uNo, t.havayolu, t.uNo
FROM Ucaklar u, Ucaklar t
WHERE val(initial(atmin(distance(u.ucus, t.ucus))))<0.5
```

Yukarıdaki sorgu hem uzaya hem de zamana bağlı join işlemi yapılarak elde edilmektedir.

Bir uzaya bağlı veri tabanı yaratmak için öncelikle bu veri tabanında saklanacak ilkel veri tiplerinin, bu veri tipleri ile yapılacak işlemlerin ve predicate'ların tanımlanması gerekmektedir. Daha sonra da veri tabanındaki bilgiler SQL ile sorgulanabilir.

Uzaya bağlı veri ilkel tipleri olarak; Line, point ve region tanımlanmaktadır. Bu ilkellerle intersection, minus, contour, sum ve length işlemleri gerçekleştirildikten sonra elde edilecek ilkel tipler aşağıdaki şekilde oluşturulmaktadır (Guting, R. H.ve Schneider, M. 2005):

```
Intersection:      line x line → points
Minus:             region x region → region
Contour:           region → line
Sum:               set(line) → line
Length:            line → real
```

Yukarıdaki veri ilkel tipleri (line, point ve region) yaratıldıktan sonra, DBMS'in veri modeli içine eklenebilir. Aşağıdaki predicate'ları da eklersek;

```
Inside:            points x region → bool
Adjacent:          region x region → bool
```

Aşağıdaki bilgilerden oluşan cities, rivers, highways ve states tabloları DBMS'e eklendiğini düşünelim:

```
Cities(name:string, population:int, location:points)
Rivers(name:string, route:line)
Highways(name:string, route:line)
States(name:string, area:region)
```

Daha sonra Fransa'nın nüfusu bilgisine ulaşmak için aşağıdaki SQL kullanılabilir:

```
Select sum(c.pop)
From cities as c, states as s
Where c.location inside s.area and s.name='France'
```

Almanya içinde Ren nehrinin parçasını döndürmek için aşağıdaki SQL kullanılabilir:

```
Select intersection(r.route, s.area)
From rivers as r, states as s
Where r.name= 'Rhine' and s.name='Germany'
```

Her ülkenin komşu ülkelerinin listesini elde etmek için aşağıdaki SQL kullanılabilir:

```
Select s.name, count(*)
From states as s, states as t
Where s.area adjacent t.area
Group by s.name
```

Böyle bir modeli gerçekleştirebilmek için veri tipleri için veri yapılarına, işlemler için de bu işlemleri gerçekleştiren algoritmalara ihtiyaç duyulmaktadır. Ayrıca uzaya bağlı kritere göre seçme (selection) ve join işlemlerinin de sağlanması gerekir. Seçim için özel endeks yapıları kurulmalıdır. Hiyerarşi şeklinde bir grup dikdörtgeni organize eden R-tree endeks yapısı, en çok kullanılan endeksleme yöntemidir. Gerçek uzaya bağlı veri tipleri (SDT-Spatial Data Types) değerleri böyle bir endekste minimum bounding rectangle (MBR) ile temsil edilir. Uzaya bağlı join için özel algoritmalar bulunmaktadır, bunların bazıları uzaya bağlı endeksleri kullanmaktadır.

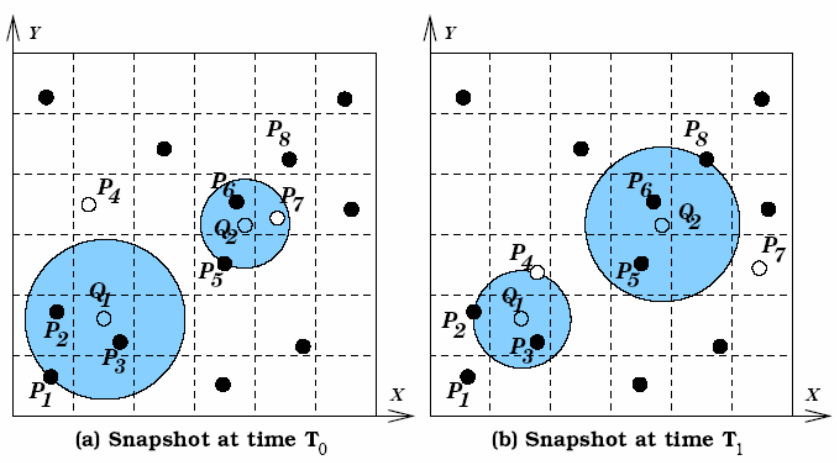
Bu yapıyı DBMS'e bütünleştirmek için geniş DBMS mimarisi gerekir (Guting, R. H.ve Schneider, M. 2005). DBMS, aşağıdaki parçaların eklenmesi için arayüzlere ihtiyaç duyar, örneğin;

- SDT'ler için veri yapıları ve algoritmaları,
- Uygun erişim yöntemleri ile uzaya bağlı endeks yapıları,
- Uzaya bağlı join metotları,
- Query optimizer için tüm metotların maliyet fonksiyonları,
- Seçim tahmini (selectivity estimation) için nesnelerin dağıtımı ile ilgili istatistikler,
- Optimizer için uzantılar,
- Sorgu dilinde tiplerin ve işlemlerin kaydı,
- Uzaya bağlı verinin kullanıcı arayüzleri ile işlenmesi.

Son olarak en yakın komşunun (Athitsos, V.vd. 2007) araştırıldığı çeşitli sorgular da araştırmalarda gerçekleştirilmiştir (Frentzos vd., 2005; Huang vd., 2005; Huang vd., 2006; Iwerks vd., 2006; Wang vd., 2007; Ding vd., 2008b). kNN sorgusunda k=3 değeri için, en yakın komşunun hesaplanması Şekil 4.10'da gösterilmiştir (Mokbel, M. F. 2004)[*]. Sorguların en iyilenmeleri için birleştirmenin, en yakın komşunun ve R-tree oluşturma maliyetleri ayrıca

[*] Daha fazla bilgi için; (Frentzos, E.vd. 2005; Iwerks, G. S.vd. 2006; Frentzos, E.vd. 2007b)

hesaplanmalıdır (Frentzos, E.vd. 2008c; Frentzos, E.vd. 2008b). Ayrıca performansı artırmak için sorgular, paralel çalıştırılabilir (Wang, X.vd. 2000).



Şekil 4.10 kNN sorgusu

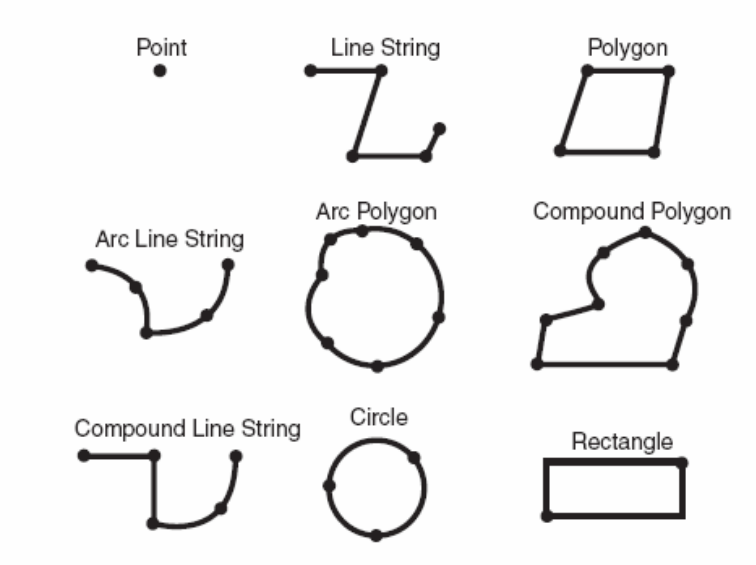
4.6 Değerlendirme

Bu bölümde soyut modelde sorgu tipleri ve dilleri tanımlanmıştır. Zaman boyutunun eklenmesi ile veri tabanlarında çeşitli sorgular gerçekleştirilebilmektedir. Bir veri tabanına geçerlilik ve birim işlem zamanları eklenerek, zamana bağlı yetenekler kazandırılabilir. Ayrıca, hem uzaya hem zamana bağlı veri tabanlarının gerçekleştirmesi gereken sorgular belirlenmiştir. Bu sorgularla tezde geliştirilecek sorgu sistemi, veri tabanı yönetim sistemi genişletilerek kurulacaktır.

5. UZAYA BAĞLI KAVRAMLARIN VERİ TABANINDA KULLANIMI

Günümüzde bir çok ticari uzaya bağlı veri tabanı yönetim sistemleri bulunmaktadır. Bu bölümde, tez kapsamında kullanılacak Oracle Spatial 10g veri modeli ve veri kartuşu yetenekleri incelenmiştir.

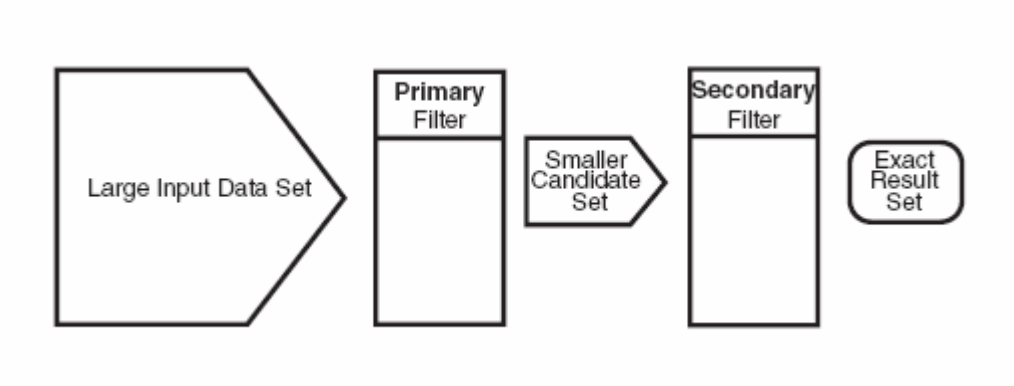
Haritalarda yer (location) ve özellikler (attributes) bilgileri ile gösterilmektedir. Yer ile bulunulan yerin koordinatları (enlem ve boylam olarak) ifade edilmektedir. Özellikler ile de karakteristik özellikler ifade edilmektedir. Örneğin yerin sahibi, kimlik numarası, verinin toplandığı tarih, vb. Oracle Spatial ile yer-tabanlı bilgiler, coğrafi bilgi sistemi bilgileri, bilgisayar destekli tasarım (CAD) ve bilgisayar destekli üretim (CAM) için bilgiler saklanabilmektedir. Bu uzaya ait bilgileri saklamak için Oracle Spatial çeşitli geometrik ilkel veri tipleri tanımlamıştır (Oracle 2006): point, line string, arc line string, compound line string, self-crossing line strings, polygon, polygon with hole, compound polygon, optimized polygons ve self-crossing polygons (Şekil 5.1).



Şekil 5.1 Geometrik veri tipleri (Oracle)

Bir geometrinin temel taşı eleman'dır (element). Oracle'ın desteklediği uzaya ait elemanlar: point, line string ve polygon'dur. Örneğin elemanlar bir yıldızı (point kümeleri), yolları (line strings) ve şehir sınırlarını (polygons) modelleyebilmektedir. Elemanlardaki her koordinat <x,y> çiftleri olarak veri tabanında saklanmaktadır. Point verileri sadece bir koordinatı içermektedir. Line verileri elemanın çizgi parçasını temsil etmek için iki koordinat tutmaktadır. Polygon veriler ise poligonun her çizgi parçası için en yüksek nokta koordinat çifti verilerini tutmaktadır (Oracle 2006).

Oracle Spatial, uzaya bağlı sorguları ve spatial join'leri çözmek için 2-katmanlı sorgu modelini kullanmaktadır. Bu iki katman ile birinci (primary) ve ikinci (secondary) filtreleme işlemleri gerçekleştirilmektedir (Şekil 5.2).



Şekil 5.2 Oracle Spatial sorgu modeli

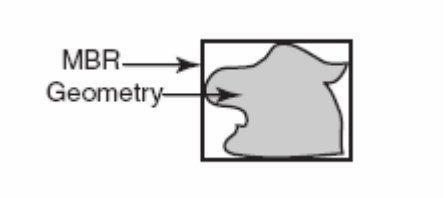
Uzaya ait verilerin kesiştiklerini ve birbirlerini kapsadıklarını anlamak için uzaya bağlı endeksler oluşturulmaktadır:

- Endekslenmiş veri uzayında verilen bir nokta veya alan ile kesişen nesneleri bulmak (window query),
- İki endekslenmiş veri uzayında kesişen nesne çiftlerini bulmak (spatial join).

Uzaya bağlı endeksler, mantıksal endeks olarak düşünülebilir. Uzaya bağlı endeksteki bilgiler, koordinat uzayındaki geometrinin yer bilgisine bağlıdır, fakat endeks değerleri farklı alandadır (domain). Endeks değerleri doğrusal olarak sıralanabilir ve geometri; tamsayı, kayan-noktalı çiftlerinden oluşabilir. Oracle Spatial; R-tree endeksleme (varsayılan), quad-tree endeksleme veya her ikisine de izin vermektedir (Oracle 2004a). Fakat performans açısından, genellikle, R-tree endeksleme yapılması tavsiye edilmektedir (Ravi Kanth V Kothuri, S. R., Ning An 2004).

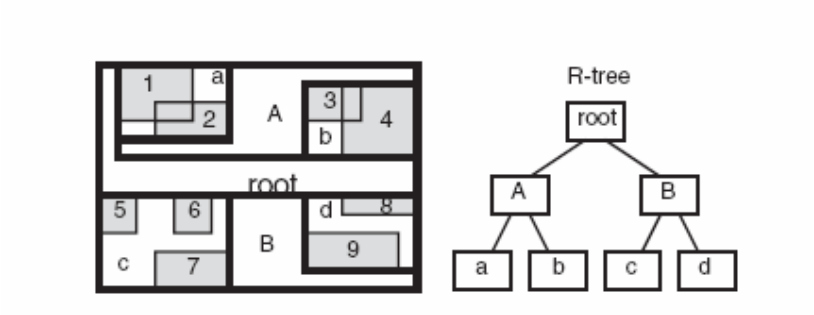
5.1 R-Tree Endeksi

R-tree endeksleme ile 4-boyuta kadar uzaya bağlı veriler endekslenebilmektedir (Oracle 2006). Bir R-tree endeksi, her geometriyi onu çevreleyen tek bir dikdörtgene yaklaşıklaştırır (minimum bounding tree-MBR) (Şekil 5.3).



Şekil 5.3 Bir geometriyi çevreleyen MBR

Bir katman (layer) geometrisi için R-tree endeks, katmandaki geometrilerin MBR'leri üzerinde hiyerarşi şeklinde endeksten oluşmaktadır (Şekil 5.4).



Şekil 5.4 MBR'lar üzerinde hiyerarşi şeklinde endeks

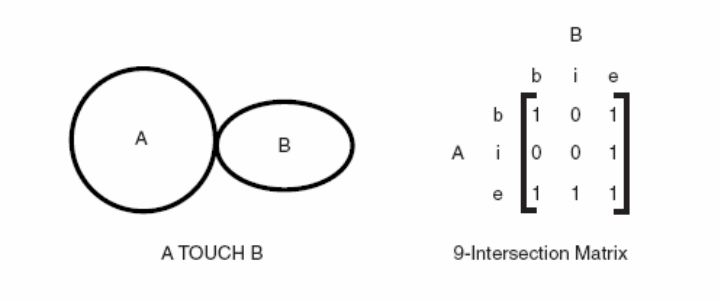
R-tree endekslerini yaratmak kolaydır. Ayrıca, daha az disk alanına ihtiyaç duyulur. Özellikle en yakın komşu sorgularını daha hızlı gerçekleştirmektedir. Fakat, veriyi güncellerken/eklerken/silerken daha yavaştır. Sık güncellemelerde, R-tree'nin tekrar oluşturulması nedeniyle daha yetersiz olmaktadır (Kothuri, R. K. V.vd. 2007).

5.2 Uzaya Bağlı İlişkiler ve Filtreleme

Veri tabanındaki varlıklar hakkındaki ilişkileri belirleyebilmek için Oracle Spatial ikinci filtrelemeyi kullanmaktadır. Uzaya bağlı ilişkiler, geometrinin yer bilgilerine dayanmaktadır. Ortak olarak en çok kullanılan uzaya bağlı ilişkiler topoloji ve mesafeye dayanmaktadır. Bazı ikinci filtreleme metotları aşağıda verilmiştir (Oracle 2006):

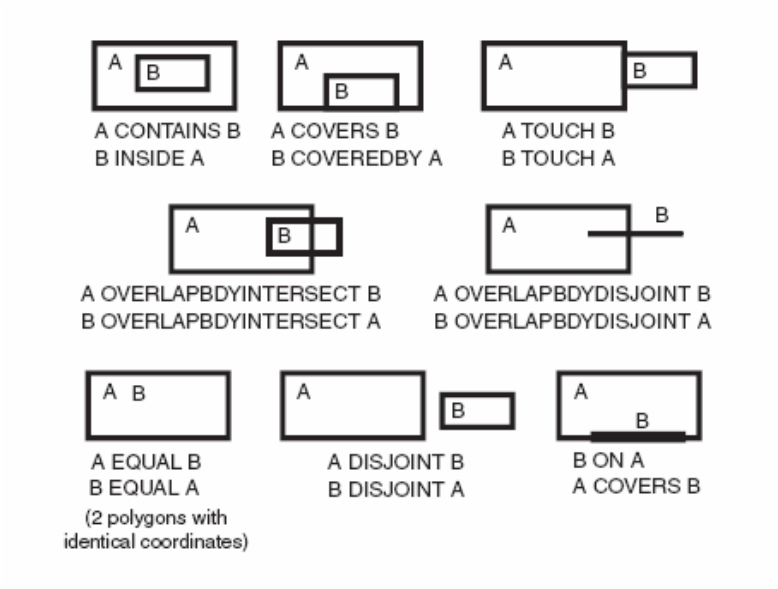
- SDO_RELATE, topolojik kritere göre işlem yapmaktadır.
- SDO_WITHIN_DISTANCE, verilen iki uzaya bağlı nesnenin birbirlerine verilen uzaklık kadar mesafede olup olmadığını araştırır.
- SDO_NN, bir spatial nesneye en yakın komşuları belirler.

SDO-RELATE ile bir topolojik ilişki, 9 farklı ilişki ile temsil edilebilmektedir. Bu da 3x3 matris ile gösterilmektedir (Şekil 5.5, b-boundary, i-interior, e-exterior).



Şekil 5.5 9 Etkileşim modeli

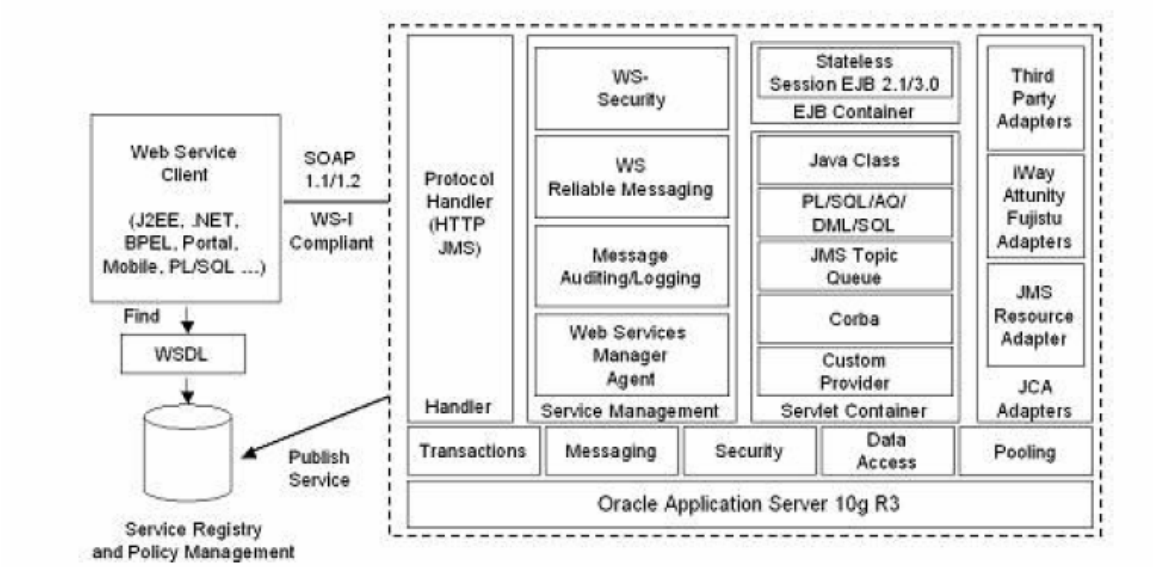
Olası tüm topolojik etkileşimler Şekil 5.6'da verilmiştir.



Şekil 5.6 Topolojik ilişkiler

5.3 Oracle 10g Yetenekleri

Oracle, Internet ve kablosuz için teknolojik çözüm olarak uygulama sunucusunu ortaya koymuştur (application server). Uygulama sunucuları; portal yazılımı, kablosuz ve ses, web sayfası ön belleğe alınarak (caching), güçlü kurum zekası (intelligence) özellikleri, tam bütünleşme gibi çözümleri sunmaktadır. Örneğin Oracle tüm J2EE, web servisleri ve XML endüstri standartları ve açık ve bütünleşmeye hazır mimarı sunarak bilişim teknolojileri ile web uygulamalarını bütünleştirebilmektedir (Karimi, H. A.ve A.H. 2004). Oracle Application Server'ın yetenekleri Şekil 5.7'de verilmiştir (Oracle 2006).



Şekil 5.7 Oracle application server yetenekleri

Oracle Spatial, Oracle 10g Veri tabanının Enterprise Edition'ın bir parçasıdır. Kurum bazında uzaya bağlı bilgi sistemlerini ve karmaşık uzaya bağlı veri yönetimi gerektiren web-tabanlı ve kablosuz yer-tabanlı geliştirmek için bir temeldir. Zengin, açık, standartları destekleyen veri tipleri ve modelleri bulunmaktadır. Ayrıca, veriyi işleyen SQL cümlelerini kullanıcılara sunmaktadır (Oracle 2008b). Oracle 10g, aşağıdaki veri tiplerine ve modellerini desteklemektedir:

- Vektör verisi: tüm basit OGC geometri tiplerini ve bunlar üzerinde temel işlemleri (operators) içerir. Ayrıca ilişki (relate) ve grup (aggregation) işlemlerini de içerir (Oracle 2006).
- Topoloji ve Ağ veri modelleri: Topolojinin geometri katmanında düğüm/kenar/face bilgilerini ve ayrıca topoloji verisini işleyen fonksiyon kümesini bulundurmaktadır (Oracle 2005b).
- GeoRaster: Raster resmi verisi, ilgili uzaya bağlı vektör geometrisi ve meta veriyi içeren geoRaster verisini saklamayı, endekslemeyi, sorgulamayı, analiz etmeyi ve dağıtmayı sağlar. GeoRaster, Oracle Spatial veri tiplerini sağlar ve çok-boyutlu grid katmanlarını ve dünyanın yüzeyine veya yerel koordinat sistemine göre referans alınmış sayısal resimleri nesne-ilişkisel şemalarını saklamak için kullanılır (Oracle 2005a).

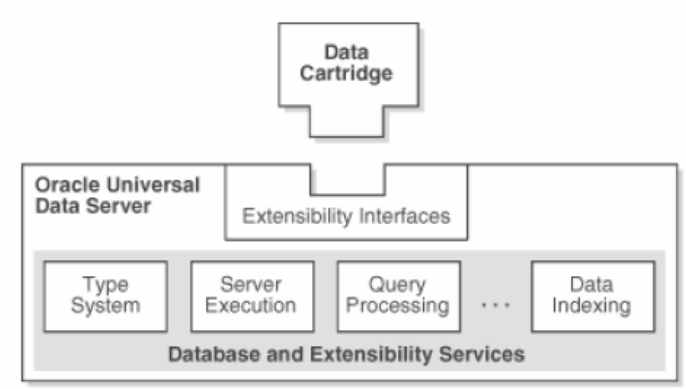
Spatial 10g, uzaya bağlı analiz ve madenciliği sağlayan SAM paketini de içerir (Oracle 2004b). SAM fonksiyonları ile kullanıcı genel amaçlı Oracle Data Mining motorunu kullanabilir. SAM günümüzde, uzaya bağlı binning, proximity, clustering ve co-location analizi fonksiyonlarını desteklemektedir.

Ayrıca Java uzaya bağlı API'lerini de Spatial 10g içermektedir. Böylece daha karmaşık uygulamalar daha kolay şekilde yazılabilmektedir. Ayrıca Java ile saklanmış programlara (stored procedures) da erişim gerçekleştirilmektedir (Oracle 1999).

5.4 Oracle Veri Kartuşu

Oracle Veri Kartuşu (Data Cartridge), sunucu tabanlı olup, yeni veri tiplerinin ve davranışlarının tanımlanmasını sağlayarak, sunucuyu genişletirler (extensible indexing). Şekil 5.8'de gösterilen veri kartuşları, seçilen özniteliklere özel olarak endeks (domain-specific indexing) tanımlanmasını ve giriş/çıkış işlemlerinde eniyileme yapılmasına olanak sağlar (Ravada, S.ve Sharma, J. 1999). Ayrıca yeni endekse uygun olarak veri yüklenme/değiştirme işlemlerini ve sorgu işleminde endekse göre arama yeteneklerini kullanıcılara sunar (Le, Y. 2004). Oracle veri tabanına bir birim olarak ayrıca yüklenirler (Oracle 2008a).

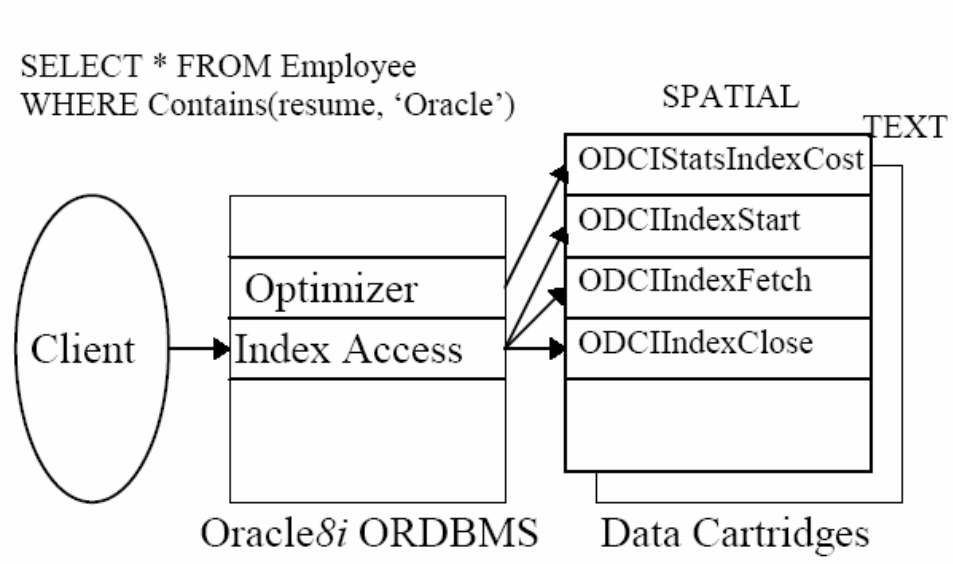
Genişleyebilir endeks yapısının en önemli avantajı, veri ile her zaman eş zamanlı olmasıdır. Diğer bir ifade ile endeks ilk defa yaratıldıktan sonra, ana tabloda veri değişiklikleri olunca veri endeks verisi de otomatik olarak değiştirilir. Böylece kullanıcılar veri tabanın bütünlüğü ve doğruluğu için ek işlemler yapmak zorunda kalmazlar (Ravada, S.ve Sharma, J. 1999).



Şekil 5.8 Oracle servisleri (Oracle)

Yeni veri tipleri olarak; kullanıcı-tanımlı nesne tipleri, özelliklerine göre sınıflanmış veriler (VARRAY-sıralı ve iç içe tablolar-sırasız), ilişkiler (REFs-bir veri satırını gösteren veri tabanı işaretçisi), büyük nesne tipleri (LOBs) tanımlanmasını veri kartuşları desteklemektedir (Scheugenpflug, S.ve Scilcher, M. 2004).

Oracle veri kartuşu arayüzü de (Oracle Data Cartridge Interface – ODCI) kullanıcı tanımlı ikincil endeks tanımlamak için geliştirilmiştir (Kriegel, H.-P.vd. 2003). Burada her tuple, row id (RID) ile materyalleştirilmiştir (Acker, R.vd. 2005). Oracle 8i'den itibaren veri kartuşları ile endeks kullanıldığı zaman, ODCI ile çalıştırılan metotlar Şekil 5.9'da gösterilmiştir (Srinivasan, J.vd. 2000).

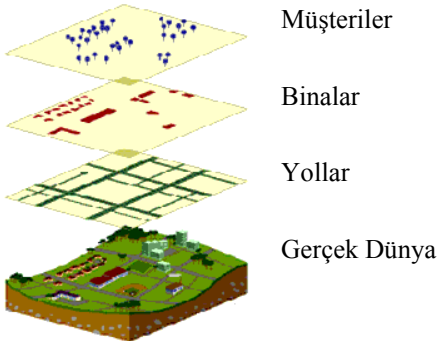


Şekil 5.9 Oracle genişleyebilir mimarisi

Oracle Extensible Optimizer, kullanıcı tanımlı fonksiyonlar ve endeksler için seçicilik ve maliyet fonksiyonları hakkında bilgi toplar ve SQL komutu için yürütme (execution) planı yaratarak veri tabanı kullanıcılarına veri kartuşu hakkında istatistikler sunar (Oracle 2004c).

5.5 Geliştirilen Örnek Hareket Eden Nesneler Uygulaması

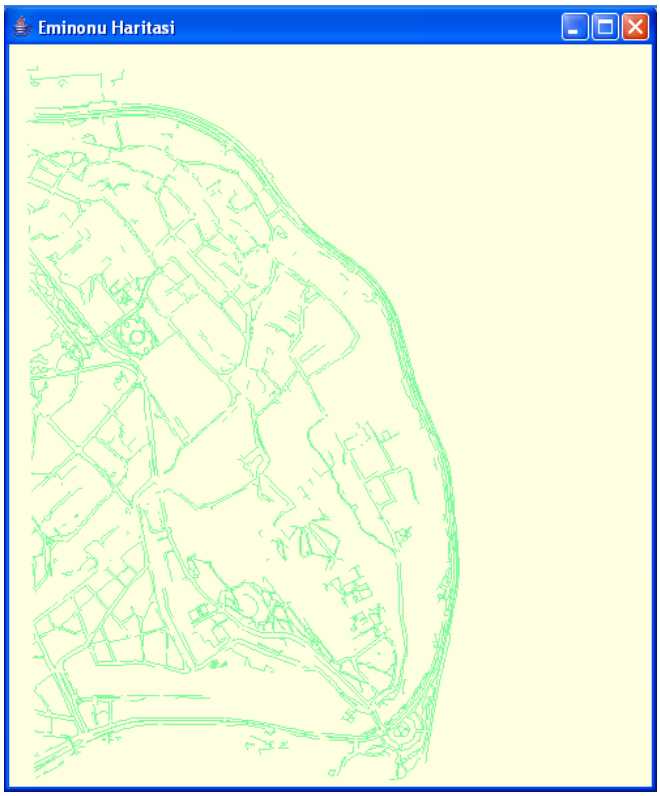
Haritaların sayısallaştırılması için, birbiri ile ilişkili bilgiler gruplanarak bir katman olarak tanımlanır. Örneğin bir organizasyon, müşterilerini sayısal harita üzerinde görmek istiyorsa, Şekil 5.10'da verilen gerçek dünyadaki resme göre yolları, binaları ve müşterileri sırasıyla sayısal haritaya eklemektedir. Yolları çizebilmek için çizgi, binaları çizebilmek için poligon ve müşterileri çizebilmek için de nokta geometrik yapılarından faydalanılabilir.



Şekil 5.10 Harita Katmanları

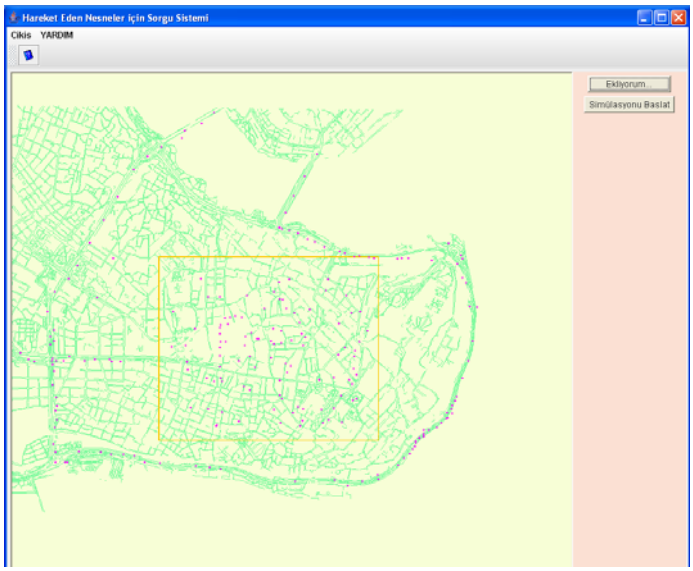
Uzay bilgisine dayalı veri tabanları, nesnelerin uzaya ait özelliklerini saklayabilmek için çeşitli geometri özelliklerden yararlanır. Oracle veri tabanı, haritada bulunan her katmanı ayrı bir tabloda saklamaktadır. Dolayısıyla, her katmanın geometri tipine göre verileri tablolardan okunabilir.

Sayısal harita bilgileri Oracle veri tabanında, Oracle Spatial Java Library (SDOAPI) ile işlenmektedir (Oracle 2000; Oracle 2001f). Bu kütüphane ile uzay verileri okunabilmekte ve değiştirilebilmektedir. Verilerin sayısal olarak alınması, bir harita arayüzü gerçekleştirmek için yeterli olmamaktadır. Bu verilerin bir grafik arayüz ile görselleştirilmesi gerekmektedir. Gerçekleştirilen uygulamada Eminönü sayısal haritası, Oracle 10g'nin izin verdiği geometriler ve Java 3D kütüphanesi ile Şekil 5.11'deki gibi bir arayüzle görüntülenebilir.



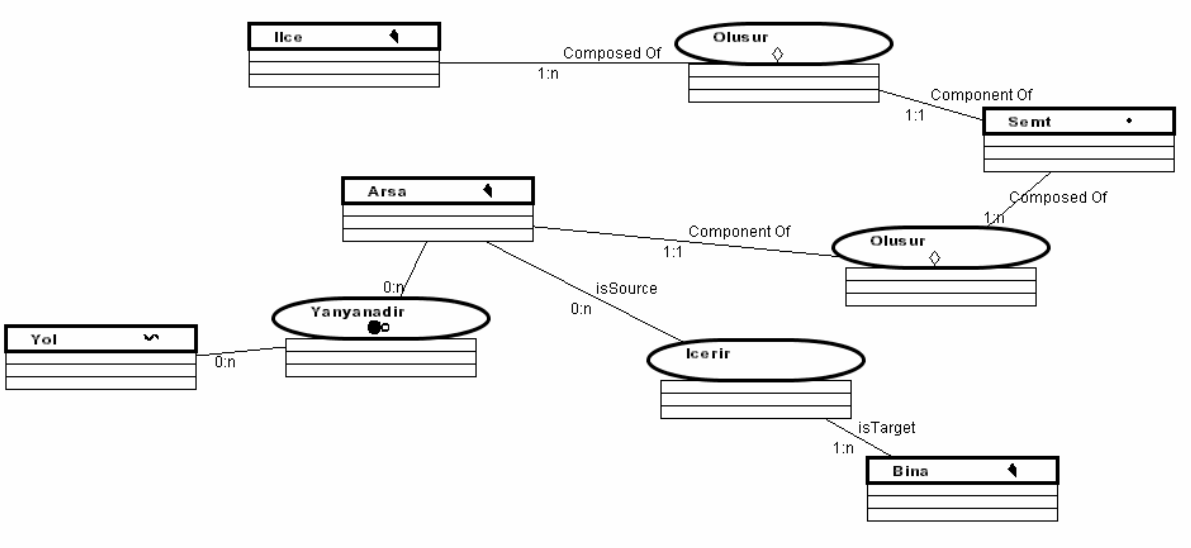
Şekil 5.11 Eminönü sayısal haritası

Şekil 5.11'deki sayısal haritanın üzerine hareket eden nesneler, Java programlama dilinde (Wood, J. 2002) gerçekleştirilen program ile rastgele oluşturulmuştur. Hareket eden nesnelerin ve sayısal haritanın görüntülenmesi belirli zaman aralıklarında tekrar edilmektedir. Hareketli nesnelerin bilgileri de belirli zaman aralıkları ile veri tabanına kaydedilmektedir (Şekil 5.12).



Şekil 5.12 Hareketli nesneler

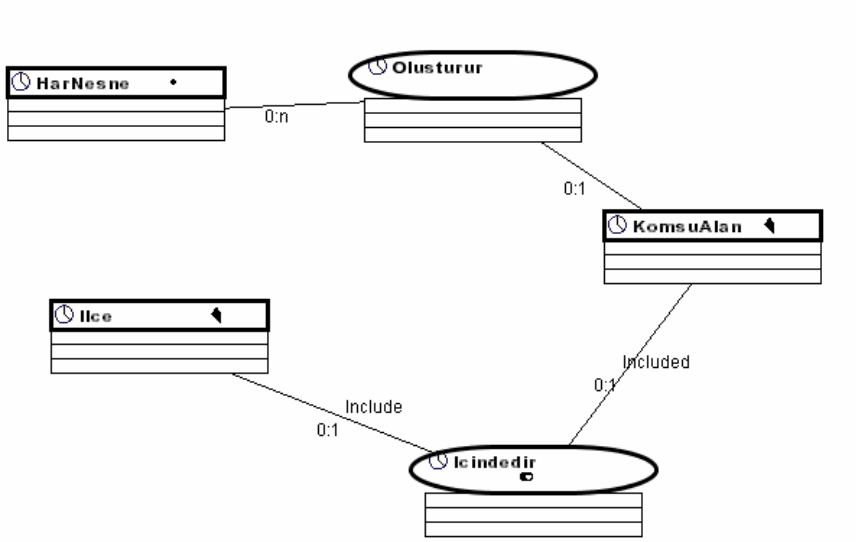
Gerçekleştirilen sistemde hareketli nesneler, kullanıcının girdiği yer bilgilerine göre sorgulanabilmektedir. Ayrıca veri tabanında zamana bağlı ilişkiler olmadığı için, bu ilişkiler gerçekleştirilmiştir. Bu sistemin MADS modeli Şekil 5.13'te verilmiştir.



Şekil 5.13 Gerçekleştirilen sistemin MADS modeli

Harita üzerinde poligon geometri tipinden oluşan ilçeler bulunmaktadır. Bir ilçede, birden fazla semt bulunabilir. Semtler nokta geometrisinden oluşmakta ve harita üzerinde nokta olarak semtler gösterilmektedir. Semtler, poligon geometrisinden oluşan arsalardan oluşmaktadır. Arsalar ile yollar harita üzerinde, yan yana bulunmaktadır. Yol, çizgi geometrisinden oluşmaktadır. Arsa üzerinde bir bina olabilir veya olmayabilir. Binalar poligon geometrisinde oluşmaktadır.

Şekil 5.14'te hareket eden nesneler için MADS modeli verilmiştir. Buna göre birden fazla hareketli nesne, en yakın komşuluk alanını oluşturmaktadır. Bu komşuluk alanı da ilçe haritasının içinde bulunmaktadır.



Şekil 5.14 Hareket eden nesneler modülünün MADS modeli

5.6 Oracle Spatial Yetenekleri

Oracle Spatial uzay işlemleri(spatial operators), geometri alt programları ve uzay aggregate fonksiyonları ile veri tabanı kullanıcılarına çeşitli yetenekler sunmaktadır. Uzaya ait işlemler ve açıklamaları aşağıda verilmiştir.

- SDO_FILTER, verilen bir geometri ile etkileşimde olan geometrileri belirler.
- SDO_JOIN, topolojik ilişkilere göre birleştirme işlemini gerçekleştirir.
- SDO_NN, bir geometriye en yakın komşuyu belirler.
- SDO_NN_DISTANCE, SDO_NN işlemi ile dönen geometrinin uzaklığını hesaplar.
- SDO_RELATE ile iki geometrinin belirtilen etkileşimde olup olmadığı belirlenir.
- SDO_WITHIN_DISTANCE, iki geometrinin verilen değer içinde birbirlerine uzakta olup olmadıkları belirlenir.

Geometri alt programları aşağıda verilmiştir.

- SDO_GEOM.RELATE, iki geometri nesnesinin nasıl bir etkileşimde olduğunu belirler.
- SDO_GEOM.SDO_ARC_DENSIFY Her dairesel arkı, yaklaşık düz çizgilere ve her daireyi de yaklaşık çizgilerden oluşan daireye çevirir.
- SDO_GEOM.SDO_AREA, iki boyutlu poligonun alanını hesaplar.
- SDO_GEOM.SDO_BUFFER, bir geometri nesnesi çevresinde veya içinde tampon yaratır.
- SDO_GEOM.SDO_CENTROID, poligonun ağırlık merkezini hesaplar.
- SDO_GEOM.SDO_CONVEXHULL, geometri nesnesinin dışbükey alt kümesini poligon

tipli nesne olarak gösterir.

- SDO_GEOM.SDO_DIFFERENCE, iki geometri nesnesi arasında topolojik farkı (MINUS işlemi), geometrik nesne olarak geri gönderir.
- SDO_GEOM.SDO_DISTANCE, iki geometri nesnesi arasında uzaklığı hesaplar.
- SDO_GEOM.SDO_INTERSECTION, iki geometri nesnesi arasında topolojik kesişimi (AND işlemi), geometrik nesne olarak geri gönderir.
- SDO_GEOM.SDO_LENGTH, bir geometri nesnesinin uzunluğunu hesaplar.
- SDO_GEOM.SDO_MAX_MBR_ORDINATE, bir nesnenin MBR'daki en büyük değerini geri gönderir.
- SDO_GEOM.SDO_MBR, geometrinin MBR'ını geri gönderir.
- SDO_GEOM.SDO_MIN_MBR_ORDINATE, bir nesnenin MBR'daki en küçük değerini geri gönderir.
- SDO_GEOM.SDO_POINTONSURFACE, poligon üzerinde olması garanti olan bir noktayı geri gönderir.
- SDO_GEOM.SDO_UNION, iki geometri nesnesinin birleşimini bir geometri nesnesi olarak geri gönderir.
- SDO_GEOM.SDO_XOR, iki geometri nesnesinin simetrik farklarını geometri nesnesi olarak geri gönderir.
- SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT, bir geometrinin geçerli olup olmadığını belirler.
- SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT, bir sütunda saklanan tüm geometrilerin geçerli olup olmadığını belirler.
- SDO_GEOM.WITHIN_DISTANCE, iki geometrinin birbirlerine verilen uzaklıkta olup olmadığını belirler.

Uzaya ait gruplama (aggregate) fonksiyonları aşağıda verilmiştir.

- SDO_AGGR_CENTROID, verilen geometri nesnelerinin ağırlık merkezini (centroid- "center of gravity") bir geometri nesnesi olarak geri gönderir.
- SDO_AGGR_CONCAT_LINES, verilen çizgileri veya çokluçizgileri birleştirerek, bir geometri olarak geri gönderir.
- SDO_AGGR_CONVEXHULL, verilen geometri nesnelerinin dışbükey alt kümelerini bir geometri nesnesi olarak geri gönderir.
- SDO_AGGR_LRS_CONCAT, verilen LRS geometri nesnelerini birleştirerek bir LRS geometrisi olarak geri gönderir.
- SDO_AGGR_MBR, verilen geometri nesnelerinin MBR'ını geri gönderir.
- SDO_AGGR_UNION, verilen geometri nesnelerinin, topolojik birleşimini bir geometri nesnesi olarak geri gönderir.

Oracle Spatial ile sunulan yetenekleri incelemek için öncelikle Oracle'da tanımlanmış sayısal haritaya ihtiyaç duyulmaktadır (Oracle 2001d; Oracle 2001e). Eminönü sayısal haritasındaki karmaşıklıkları indirgemek için basit bir harita üzerinde bu fonksiyonların uygulamaları Ek 1'de verilmiştir.

5.7 Değerlendirme

Bu bölümde uzaya bağlı kavramların veri tabanında kullanımı incelenmiştir. Tez çalışması için seçilen Oracle Spatial 10g veri tabanı ve veri kartuşu yetenekleri incelenmiştir. Oracle; hareketli nesneler uygulamalarını gerçekleştirmek için, uzaya bağlı pek çok fonksiyonu, alt yordamları ve grup fonksiyonlarını programcılara sunmaktadır. Sayısal bir haritanın grafik arayüz programı geliştirilerek, uzaya bağlı kavramlar araştırılmıştır. Bu bölümde İstanbul Büyükşehir Belediyesinden temin edilen Eminönü sayısal haritası verileri kullanılmıştır. Grafik arayüzde hareket eden nesneler rastgele hareket etmektedir.

Java programlama ile haritalar görselleştirildiği gibi (Oracle 2007), Visual Basic programlama ile de aynı işlem yapılabilir (Ralston, B. A. 2002). Tez çalışması kapsamında Oracle web sitesinden temin edilen USA sayısal haritası üzerinde deneme çalışmaları Ek 2'de verilmiştir.

Hareket eden nesneler söz konusu olduğunda, verilerin değiştirme aralıkları da performansı doğrudan etkilemektedir. Veri kartuşları ile gerçekleştiren çalışmalarda, verinin değiştirme komutu verildiğinde (update time) veya birim işlemi tamamlayıp kaydettikten sonra (commit time) R-tree endeks yapısına aktarılması söz konusudur. Verinin kaydedildikten sonra, R-tree endeks yapısına aktarılması ile bir çalışmada performans artışları elde edilmiştir (Kothuri, R. K. V.vd. 2004).

Bunlara ek olarak, Oracle veri tabanında hareket eden nesneler için tasarlanan endeks yapılarının paralel işlem yapması ile bir çalışmada ciddi sorgu en iyilemeleri gerçekleştirilmiştir (Kothuri, R. K. V.vd. 2003; Oracle 2008c).

Hareket eden nesneleri modelleyebilmek için, Oracle Spatial'e zaman özelliklerin ve işlemlerinin eklenmesi gerekmektedir (Fengli, Z.vd. 2003). Bu konuda gerçekleştirilmiş veri kartuşu çalışması (Jin, P.ve Sun, P. 2008) az sayıda gerçekleştirilmiştir (Le, 2004).

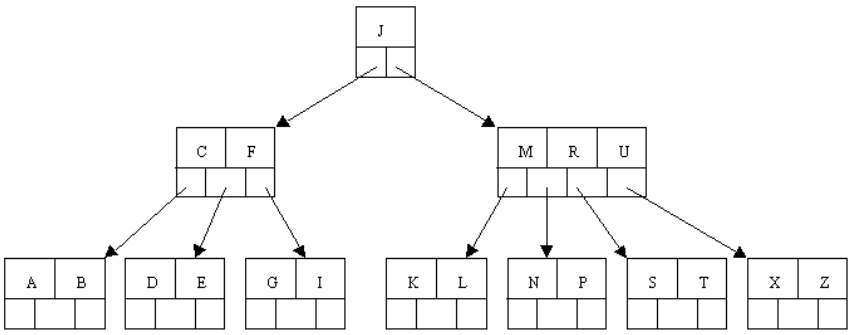
Son olarak, hareket eden nesnelerin sürekli olarak izledikleri örüntüleri bulmak için aynı veri kümelerini hesaplama konusu çeşitli çalışmalarla araştırılmaktadır (Behr, T.vd. 2006). Bu konuda RDBMS desteği sağlayarak da bir çalışma gerçekleştirilmiştir (Li, W.ve Mozes, A. 2004; Behr, T.vd. 2006).

6. MOORA MODELİ: HAREKET EDEN NESNELER İÇİN SORGU SİSTEMİNİN GERÇEKLEŞTİRİMİ

Bu bölümde öncelikle, ilişkisel veri tabanı yönetim sisteminde kullanılan endeks yapıları incelenmiştir. Daha sonra tez kapsamında ele alınan problemin kavramsal modeli, incelenen ve deneyler yapılan endeks yapıları, önerilen endeks yapısı, sorgu dili tasarım ve gerçekleştirim adımları verilmiştir.

6.1 İlişkisel Veri tabanı Yönetim Sistemlerinde Kullanılan Endeks Tipleri

RDBMS, kıyım-tabanlı (hash-based) ve ağaç-tabanlı endekslemeye olanak sağlamaktadır. Kıyım-tabanlı endeksleme, kayıtları (records) disk üzerindeki sayfalara haritalar. Özel bir değeri etkin olarak sorgulayabilmek için, o değere göre kayıtları diskten getiren kıyım-fonksiyonu uygulanır. Aynı şekilde RDBMS, ağaç-tabanlı endeksleme için B+tree oluşturulur. Sıralanmış tablo veride belirli bir sütun için, B + tree yaprak düğümler seviyesine kadar işaretçilerle gerçekleştirilir. Her düğüm, disk üzerindeki bir sayfayı gösterir. Şekil 6.1'deki B + tree'de üst düğümler, içerdikleri alt düğümlerdeki en küçük ve en büyük aralık değerlerine göre oluşturulurlar. Kök düğüm hariç, her düğüm en küçük **m** en büyük de **M** giriş alarak, ağacın dengeli bir yapıda olmasını sağlar (Mallett, D. J. 2004).

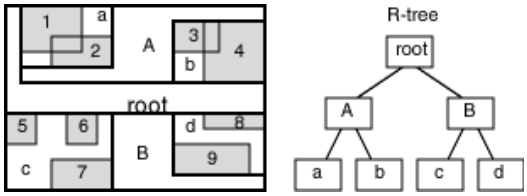


Şekil 6.1 B+tree

B+tree nokta sorgularında, tek bir değere göre ağaç araştırılmaktadır. Aralık sorgularında ise, ağaçta tek bir yol izlenilerek aşağı düğümlere inilir, aralıktaki en küçük değer bulunduktan sonra yaprak düğümlere kadar arama yapılarak düğümler arasındaki işaretçilerle cevap oluşturulur (Mallett, D.vd. 2005).

Uzaya bağlı verileri endeksleme için RDBMS, Şekil 6.2'de gösterilen R-tree yapısını kullanmaktadır. Veri tabanında bulunan her uzaya bağlı nesne için, en küçük kuşatan dikdörtgen (Minimum Bounding Rectangle - MBR) yaklaşıklaşma R-tree'nin yaprak

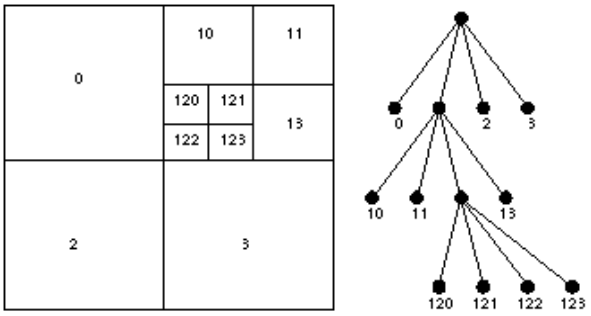
düğümlerine eklenmektedir (Oracle 2006).



Şekil 6.2 R-tree

R-tree, B + tree'deki gibi her düğümün en küçük ve en büyük girdi sayısı ile ağacın dengelenmesini sağlar. Sorgu zamanında ağaç, kökten itibaren sorgu penceresi ile MBR'nin kesiştiği her düğüm araştırılır. Yaprak seviyesinde, sorgu penceresi ile kesişen MBR'lar diskten okunur. B + tree'de bir veriye ulaşmak için tek bir yol araştırılmasına rağmen, R-tree'de bulunan birkaç yol sorgu penceresinin birkaç MBR ile kesişme olasılığından dolayı araştırılmaktadır.

RDBMS, uzaya bağlı endekslemeyi sağlamak için ayrıca Şekil 6.3'te verilen quad-tree yapısını kullanmaktadır. Quad-tree, nesnelerin sürekli olarak dört parçaya ayrıldığı partition-tabanlı endekslemeyi sağlar. Birçok uygulamada, R-tree yapısının, quad-tree'den daha iyi performans verdiği bilinmektedir (Kothuri, R. K. V.vd. 2002).

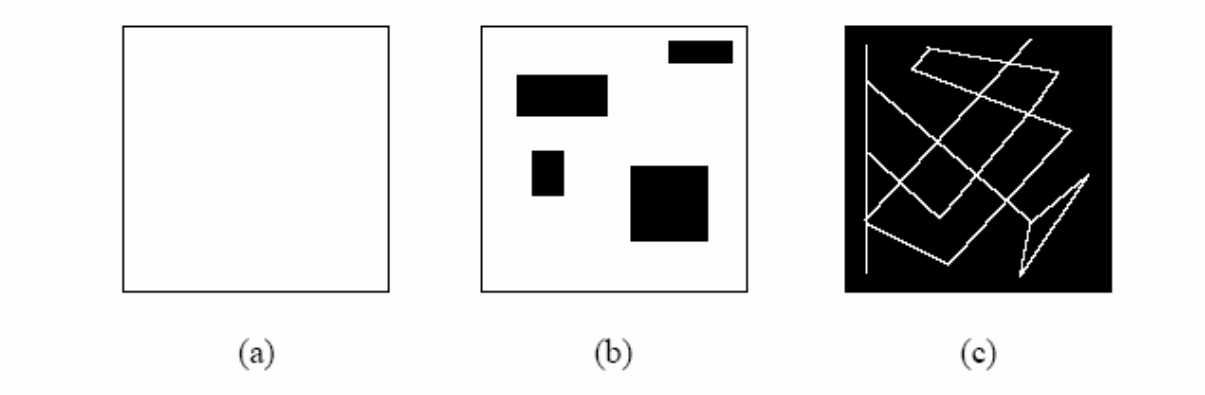


Şekil 6.3 Quad-tree

Uzaya bağlı veri gibi, RDBMS içinde zamana bağlı veri tipleri ve endeksleme yapılarına gereksinim duyulmaktadır. Standart veri tabanı zaman içinde tek bir anda verinin görünümü ile ilgilenir. Fakat bir çok alan uygulamasında, geçmişe ait bilgilerin veya geleceğe ait geçerlilik süresinin saklanması önemlidir. Buna bir yaklaşım, tek-boyutlu zaman aralık endekslemesini etkin şekilde yapan değiştirilmiş R-tree yapısıyla Relational-Interval Tree (RI-Tree)'dir (Kriegel, H.-P.vd. 2000; Kriegel, H.-P.vd. 2001). Bu yaklaşım, MAP21 tekniği ile gerçekleştirilmiştir (Nascimento ve Dunham, 1999). MAP21'de en büyük zaman aralığı başlangıç ve bitiş değerleri ile tek boyutlu değere eşleştirilerek B + tree'de saklanır (Kothuri, R. K. V.vd. 2008).

6.2 Hem Uzaya Hem Zamana Bağlı Veri

Bu tip veriler; insanların, hayvanların, araçların, uçakların, gemilerin, askeri nesnelerin, havanın hareketlerinden elde edilebilir. Hareket eden nesnelerin sınıflanmasında üç tip hareket senaryoları bulunmaktadır (Pfoser, D. 2002): kısıtlı hareket (constrained movement), kısıtsız hareket ve ağlarda hareket (movement in Networks) (Şekil 6.4).



Şekil 6.4 Hareket senaryoları: (a) kısıtsız (b) kısıtlı (c) bir ağ üzerinde

Kablosuz cihazlar hem uzaya hem de zamana bağlı verileri yaratmaktadır. Ağın yoğunluğuna göre, bu yaklaşımın doğruluğu oldukça yüksektir (yaklaşık 2 metre) (Mallett, D.vd. 2005). Ayrıca kablosuz ağ yapıları, sadece kısıtlı bölgelerde (binalar, üniversiteler, vb.) kurulabilmektedir.

Bu cihazların özel ve şahsi mallara ait olmasından dolayı, deneyler için bu gibi gerçek veri kümelerine ulaşmak mümkün olamamaktadır. Bazı hayvan izleme, volkanlar ve halka ait otobüs kümeleri araştırmacılara sunulmaktadır (Nascimento, M. A.vd. 2003). Fakat, yüzlerce kayıt, büyük ölçekli veri tabanlarında benchmark olmak için yeterli olmamaktadır. Bu nedenle de, sentetik hem uzaya hem de zamana bağlı veri üreteçleri geliştirilmiştir. Bunlardan en önemlileri, Generate SpatioTemporal data aracı (GSTD) (Theodoridis, Y.vd. 1999), Network-based Generator of Moving Objects (Brinkhoff, T. 2002) ve Generator for Time-Evolving Regional Data (G-TERD)'dir (Tzouramanis, T.vd. 2002).

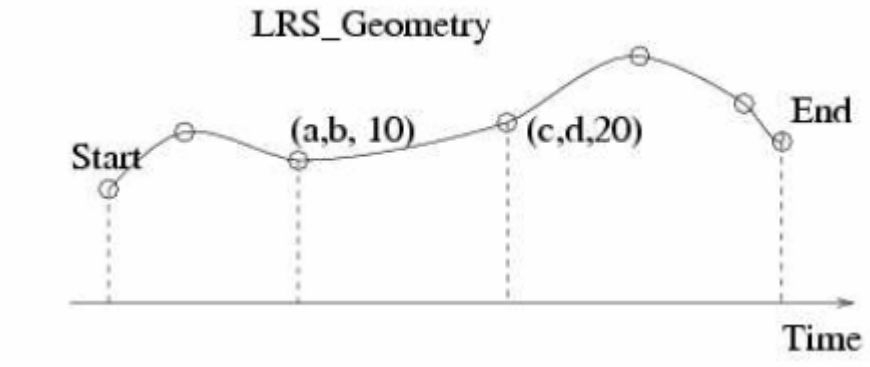
Network-based Generator, verilen bir ağda nesneler için veri kümeleri yaratır. Böyle bir senaryoda konumun ve hedefin önemli olduğu, olası en yüksek hız ve ağın kapasitesi önemlidir (Brinkhoff, T. 2002). Nesneler, başlangıç konumundan hedef konumlarına doğru hareket ederler. Bu üretecin, sentetik veriyi gerçek bir ağ üzerinde oluşturması önemli bir özelliktir.

6.3 RDBMS İçinde Hem Uzaya Hem Zamana Bağlı Endeksleme Yapıları

RDBMS'in içinde hem uzaya hem zamana bağlı veriyi endeksleyerek saklamak için çeşitli yöntemler bulunmaktadır. Bu bölümde; Linear Referencing System yöntemi, Uzay Doldurma Eğrileri yöntemi, B-tree ile Z-order yöntemi, Space Partititoning with Indexes on Time yöntemi ve Z-order sıralı SPIT incelenecektir.

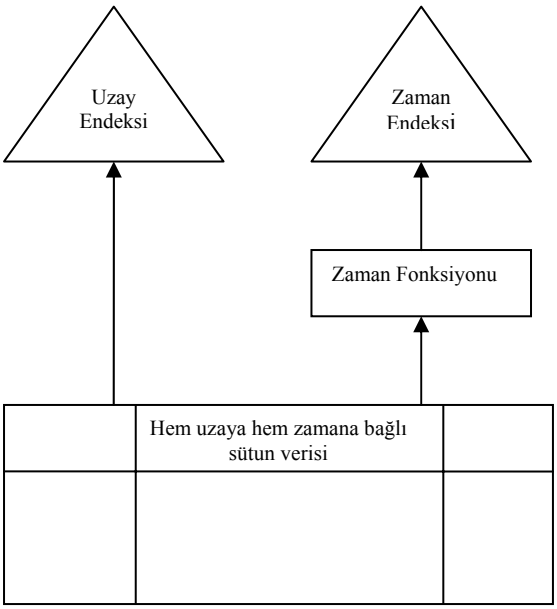
6.3.1 Linear Referencing System (LRS) yöntemi

Çoğu RDBMSs firmaları; DB2, MySQL ve Oracle uzaya bağlı veriyi, R-tree veya quad-tree desteği vererek sağlamaktadır (Oracle 2003). Uzaya bağlı veriyi RDBMS desteği alarak endeksleme için en kolay yöntem; zamana bağlı veriyi, uzaya bağlı veri olarak değerlendirip modellemektir. İki boyutlu uzaya bağlı konum bilgilerini ve tek boyutlu zamana bağlı verileri birleştirerek 3-boyutlu R-tree oluşturulabilir (Şekil 6.5).



Şekil 6.5 LRS_Geometry ile hem uzaya hem zamana bağlı endeksleme

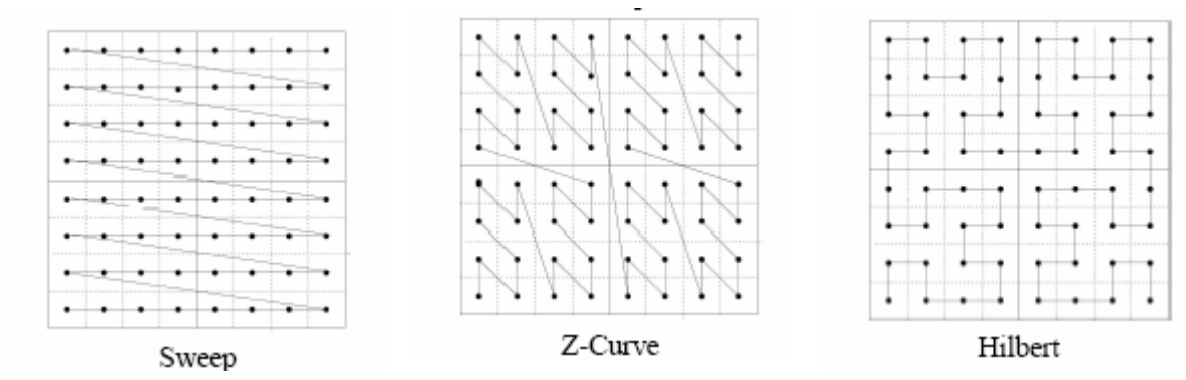
Fakat veri tabanında zaman bilgisi olmadığından dolayı, bu tür bir yaklaşımın performansı düşüktür. Oracle'ın Linear Referencing System'ini (LRS) kullanmanın avantajı zaman boyutunu ek bir sütun olarak tabloda yaratmamaktır (Kothuri, R. K. V.ve Ravada, S. 2002). Şekil 6.5'te LRS_Geometry nesnesi hareketli nesnenin izlediği gezingeyi (trajectory) göstermektedir. (a, b, 10) noktası, (a, b) konumunda 10 zaman dilimini göstermektedir. Gezingenin başlangıç ve bitiş noktaları LRS paketini kullanarak elde edilebilir. Şekil 6.6'da LRS-tabanlı ikili endeksleme yapısı gösterilmiştir. LRS yaklaşımı, tez çalışması kapsamında deneyler sırasında kullanılmamıştır.



Şekil 6.6 LRS yaklaşımı ile hem uzaya hem zamana bağlı endeksleme

6.3.2 Uzay doldurma eğrileri yöntemi

Hem uzaya hem zamana bağlı veriyi RDBMS içinde endeksleme için bir diğer yöntem de uzay doldurma eğrilerini (space-filling curves) kullanmaktır. Verilen bir veri uzayının dikdörtgen hücrelere ayrıldığı düşünüldüğünde, uzay doldurma eğrileri grid içinde her hücreyi tek defa dolaşmayı sağlar. Böylece çok-boyutlu uzayı, doğrusal sıralar (linear-order). B+tree, doğrusal olmasından dolayı uzay doldurma eğrileri, çok-boyutlu verinin tek-boyutlu veriye haritalanmasını B+tree ile gerçekleştirebilir. Nesneler (x, y) konumundan ziyade, hücre numaralarını gösteren tamsayılara yaklaşıklaştırılabilir. Sweep, Z ve Hilbert uzay doldurma eğrileri bulunmaktadır (Mokbel, M. F.vd. 2002). Şekil 6.7'de bu eğriler gösterilmiştir.

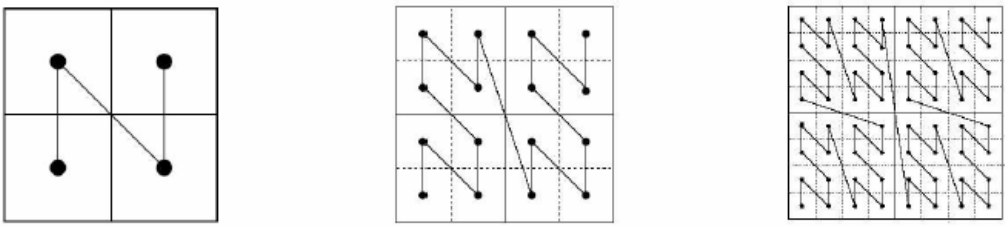


Şekil 6.7 Uzay doldurma eğrileri

Haritalama, yerelliği-koruma (locality preserving) ilkesine göre gerçekleştirilmelidir. Böylece iki-boyutlu uzayda noktalar yan yana gelerek, eğrideki tek-boyut yaratılır. Her tip eğri, farklı doğrusal sırayı gösterir. Bu eğrilerden Z-eğrisi, ikili bilgilerle kolayca kodlandığı için daha iyi performanslar üretmektedir. Hilbert-eğrisi en iyi sonucu vermesine rağmen hesaplama maliyeti fazladır. Bir diğer çalışma da, B+tree'de Z-eğrilerini kullanarak geo-spatial işlemlerin RDBMS içinde nasıl gerçekleştirildiği üzerinedir (Freytag, J.-C.vd. 2000). Bu çalışmada, konuma bağlı her kayıt hem uzaya hem zamana bağlı alana yaklaşıklaştırılır. Freytag. çalışmasında olduğu gibi, bu tez çalışmasında da Z-eğrisi kullanılacaktır.

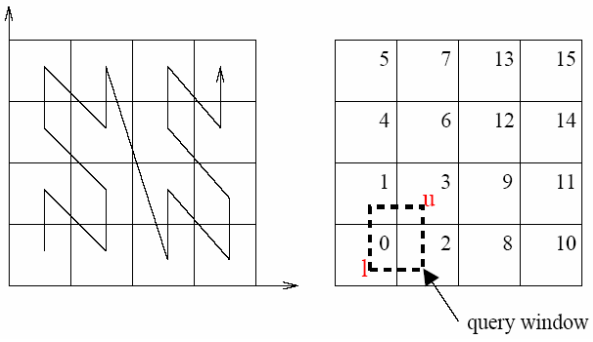
6.3.2.1 B-tree ile z-order yöntemi

Bu yöntem ile uzay, Z-order yöntemine göre numaralanmaktadır. Her hücre tek defa dolaşıldığı için çok-boyutlu uzay, doğrusal olarak sıralanır (z-order). Zaman boyutu için de ayrı bir endeks yaratılmaktadır. Şekil 6.8'de z-order'ın özyinelemeli olarak elde edilmesi gösterilmiştir.



Şekil 6.8 Özyinelemeli olarak z-order hesaplaması

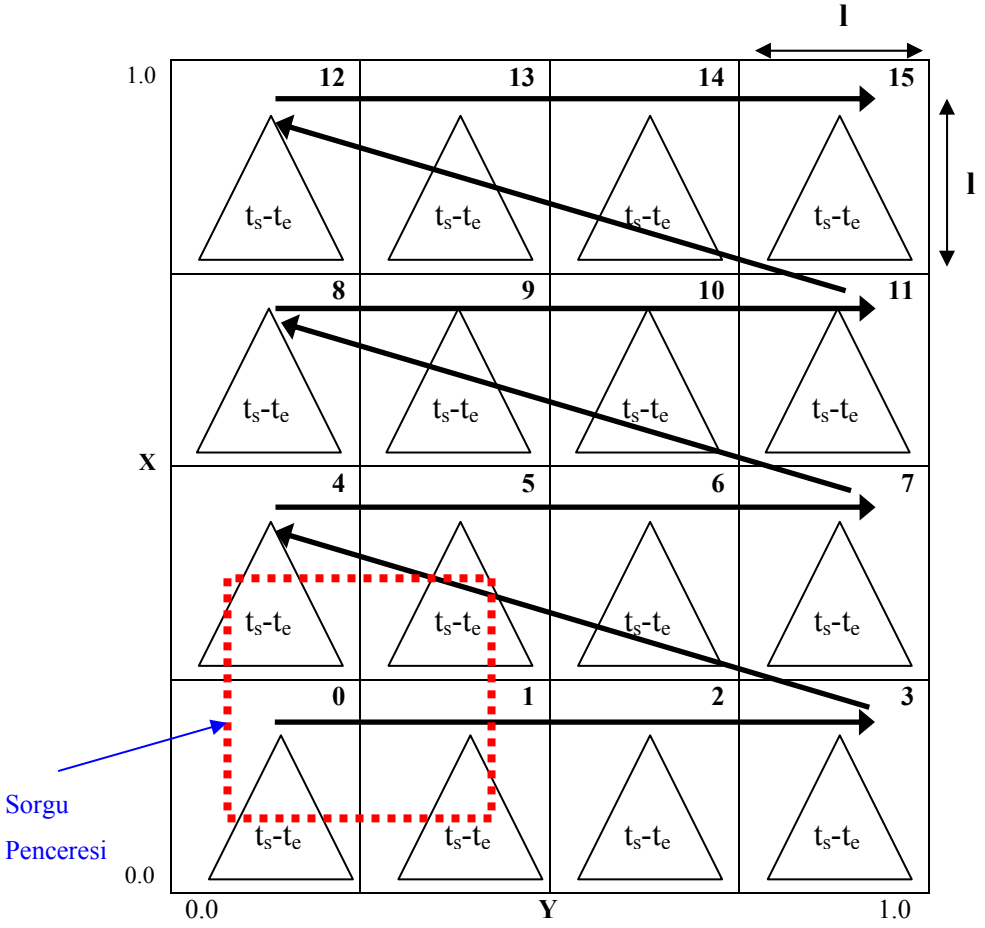
Aralık sorgusu sırasında birbirine yakın olan uzaylar, sabit disk üzerinde de yakın olduğu için sorgu zamanı azalmaktadır. Şekil 6.9'da **l**, en küçük z-order değerli hücreyi; **u** ise en büyük z-order değerli hücreyi göstermektedir.



Şekil 6.9 B-tree Z-order yöntemi ile aralık sorgusu

6.3.3 Space-Partitioning with Indexes on Time (SPIT) yöntemi

Space-Partitioning with Indexes on Time yöntemi ile veriler uzay bilgilerine göre bölünüp, her grid'de zamana bağlı (t_{start} ve t_{end}) yerel endeksler tanımlanır (Şekil 6.10). Veriler sabit sayıda grid hücreye ayrılarak, sabit hücre sayısı elde edilir (Mallett, D.vd. 2005).



Şekil 6.10 4x4 grid içinde SPIT

Bu yöntemde uzay, sweep algoritmasına göre sıralanmaktadır. İlk önce ilk satır, sonra ikinci satırda bulunan uzay, ..., en son n numaralı satırda bulunan uzay numaralandırılmaktadır. Böylece her hücreye tek (unique) bir pid numarası atanmaktadır. Burada **l**, her hücrenin uzunluğunu göstermektedir. Hem uzaya hem zamana bağlı aralık sorgusu, R uzaya bağlı bir alanda ve T zaman aralığında R içinde bulunan <x, y> noktalarını ve T ile kesişen <t_s, t_e> zamanları içeren kayıtların farklı nesne id'lerinin hesaplanması olarak tanımlanmıştı. Bu tip aralık sorgusunu cevaplamak için SPIT algoritması, her grid içinde yerel zaman endeksleri tanımlamaktadır. Uzaya göre belirli bir alanı bölmenin amacı, sorgu sırasında grid'leri elemektir. Şekil 6.10'da verilen sorgu penceresi içinde sadece <0, 1, 4, 5> numaralı grid hücreler incelenmektedir. Sorgu alanı dışındaki grid'ler bu bölümlemeden dolayı

incelenmemiştir. Ayrıca, SPIT yönteminde **l** uzuluğunun belirlenmesi çok önemlidir. Bu parametreye göre uzay grid'lere bölünmekte ve sorgu performansı elde edilmektedir.

Chakka ve arkadaşlarının çalışmasında belirtildiği gibi, SPIT yöntemi ile de sorgu işleme aynı 4 aşamada gerçekleşmektedir (Chakka, V. P.vd. 2003);

- Veritabanındaki kayıtları uzaya göre filtreleme,
- Grid'lerdeki zamana bağlı endeksler ile zamana göre filtreleme,
- Veritabanındaki kayıdın uzaydaki gerçek yerine göre uzayını iyileştirme,
- Aynı kayıtları ayıklama.

6.3.3.1 SPIT yöntemi algoritmaları

SPIT yöntemini RDBMS içinde tanımlamak için partition numaraları (pid) belirlenmeli, buna partitioned tablo oluşturulmalı, her partition'da zamana bağlı endeksler oluşturulmalı ve sorgu mekanizması kurulmalıdır. Verilen bir <x, y> noktası için pid hesaplama Algoritma 1'de, PL/SQL karşılığı Ek 4'te verilmiştir (Mallett, D. J. 2004). En büyük x ve y değerleri 1.0'dır.

Algoritma 1 *find_pid*() fonksiyonu

INPUT: <*x* ,*y*>
OUTPUT: pid

1: $x\_grid := \lfloor x \times N_p^* \rfloor$
2: $y\_grid := \lfloor y \times N_p^* \rfloor$
3: **if** *x=1.0* **then**
4: $x\_grid := x\_grid - 1$
5: **end if**
6: **if** *y=1.0* **then**
7: $y\_grid := y\_grid - 1$
8: **end if**
9: **return** $x\_grid + N_p^* \times y\_grid$

Burada uzayın $N_p^* \times N_p^*$ şeklinde bölündüğü varsayılmaktadır. Örneğin 25 grid'e bölümleme için 5x5 şeklinde bir grid oluşturulması gerekmektedir. Algoritma 1'e göre pid numaraları hesaplandıktan sonra, veritabanı özelliğinden partitioned olarak bir tablo yaratılarak veriler tabloya eklenmelidir. Böylece, aynı pid numaralı bilgiler aynı partition'da saklanmaktadır. SPIT yöntemi için son olarak, oluşturulan bu partitioned tabloya <t_s, t_e> zaman aralığı olarak zamana bağlı endeks oluşturulmalıdır.

Sorgu işleme SPIT yönteminde de Chakka ve arkadaşlarının tanımladığı gibi dört aşamadan oluşmaktadır. Bahsedilen bölümlenmiş uzay içinde, bir seçim sorgusu gerçekleştirmek için Algoritma 2 kullanılmaktadır (Botea, V.vd. 2008).

Algoritma 2 *st_sorgusu*() fonksiyonu

INPUT: <ℜ,ℑ>
OUTPUT: oid_list
1: *pid_list* := *p_intersect*(ℜ) // algoritma 3te
2: **for all** *pid* in *pid_list* **do**
3: *oid_list* := *oid_list* ∪
4: SELECT *oid*
5: FROM partition(*pid*)
6: WHERE t_s between ℑ.t_{min} - *MAX_TI* and ℑ.t_{max}
7: AND t_e between ℑ.t_{min} and ℑ.t_{max} + *MAX_TI*
8: AND x between ℜ.x_{min} and ℜ.x_{max}
9: AND y between ℜ.y_{min} and ℜ.y_{max}
10: **end for**
11: *oid_list*'i sırala ve çift girişleri ayıkla
12: **return** *oid_list*

Chakka ve arkadaşlarının tanımladığı dört aşama Algoritma 2 üzerinde aşağıdaki satırlar üzerinde gerçekleştirilmektedir:

- Veritabanındaki kayıtları uzaya göre filtreleme: 1 numaralı satır,
- Zamana göre filtreleme: 6 ve 7 numaralı satırlar,
- Uzaydaki gerçek yere göre uzayı iyileştirme: 8 ve 9 numaralı satırlar,
- Aynı kayıtları ayıklama: 11 numaralı satır.

Algoritma 2'de kullanılan intersect algoritması Algoritma 3'te verilmiştir.

Algoritma 3 *p_intersect*() fonksiyonu

INPUT: ℜ
OUTPUT: pid_list
1: pid_{min} := *find_pid*(ℜ.x_{min}, ℜ.y_{min})
2: pid_{max} := *find_pid*(ℜ.x_{max}, ℜ.y_{max})
3: *num_rows* := $\left\lfloor \frac{pid_{\max} - pid_{\min}}{N_p^*} \right\rfloor$
4: **for** i:= pid_{min} to $pid_{max} - num\_rows \times N_p^*$ **do**
5: **for** j := 0 to *num_rows* **do**
6: *pid_list* := *pid_list* ∪ ($i + j \times N_p^*$)
7: **end for**
8: **end for**
9: **return** *pid_list*

6.3.3.2 SPIT'in maliyet modeli

Sabit grid sayıda oluşan bir bölümleme için, olması gereken grid sayısını hesaplama formülü Çizelge 6.1'deki semboller kullanılarak bu bölümde açıklanacaktır.

Çizelge 6.1 Maliyet modeli sembolleri

Sembol	Anlamı
N	Veri tabanındaki kayıt sayısı
DA	Bir sorguyu cevaplamak için disk G/Ç sayısı
GA	Grid hücre ortalama erişim sayısı
DA_g	Diskteki veri sayısı her erişilen grid için
IA_g	Diskteki endeks sayısı her erişilen grid için
f	B-tree endeksinin fanout'ı
BS	Blok büyüklüğü (bir bloktaki kayıt-tuple sayısı)
q	Uzay boyutunda sorgunun büyüklüğü
q_t	Zaman boyutunda sorgunun büyüklüğü
l	Her boyutta grid'in uzunluğu
l^*	Her boyutta grid'in en iyi (optimum) uzunluğu
N_g	Grid'deki toplam hücre sayısı = $(1/l)^2$
N_{g^*}	Grid'deki toplam optimum hücre sayısı = $(1/l^*)^2$
N_p^*	Bir boyuttaki optimum partition sayısı

Her boyut için birim uzayda [0, 1] bir sorguyu cevaplayan toplam disk erişimi, her partition (grid hücre) için verilerin ve zaman endeks yapısının okunması için geçen süre kadardır. Bu da (6.1)'deki gibi ifade edilebilir.

$$DA = GA \times (DA_g + IA_g) \tag{6.1}$$

Ortalama okunacak hücre sayısı (GA), toplam hücre sayısının sorgu penceresinin kapladığı alanın **l** kadar genişletilmesi ile elde (6.2)'deki gibi elde edilebilir.

$$GA = N_g (l + q)^2 \tag{6.2}$$

Normal dağılımla verinin uzaya yayıldığı kabul edilirse, ortalama olarak her grid hücrede N/N_g kayıt(tuple) bulunur. Bu da disk üzerinde $\frac{N/N_g}{BS}$ blok saklandığını göstermektedir. Zamana göre endeks $<t_s, t_e>$ kayıt aralığını göstereceği için zamana bağlı sorguyu sağlayan kayıtlar okunacaktır (6.3)

$$DA_g = \frac{N/N_g}{BS} x q_t \tag{6.3}$$

B-tree'de,<t_s, t_e> aralığını sağlayan kayıtların olmadığı varsayıldığında, endeks erişimi fanout'a göre hesaplanmaktadır: $IA_g=\log_f N$. Burada N'nin milyon seviyesinde tuple olduğu ve f≈100 olduğu kabul edildiğinde $IA_g=3$ olur. (6.2) ve (6.3) kullanılarak aşağıdaki şekilde (6.4) eşitliği elde edilebilir.

$$DA = (1+q)^2\left(\frac{Nxq_t}{BS} + \frac{3}{l^2}\right) \tag{6.4}$$

(6.4)'ten hemen görüleceği gibi endeks performansını, sorgunun uzay parçasının büyüklüğü zaman parçasından daha çok etkilemektedir. Bu da sorgu alanını artırmanın, daha fazla partitionda endeksi kontrol etmek demektir. Disk erişimini azaltmak için (6.4)'ün türevi alındığında (6.5) elde edilir.

$$l^* = \sqrt[3]{\frac{6qxBS}{2Nxq_t}} \tag{6.5}$$

Optimum grid hücre sayısı (6.6)'da verilmiştir.

$$N_g^* = \frac{1}{(l^*)^2} = \left(\frac{Nxq_t}{3qxBS}\right)^{2/3} \tag{6.6}$$

SPIT'te optimum grid sayısını hesaplamada veritabanındaki kayıt (tuple) sayısı (6.6)'ya göre etkilenmektedir. Son olarak, her boyutta partition sayısı (6.7)'ye göre hesaplanabilir.

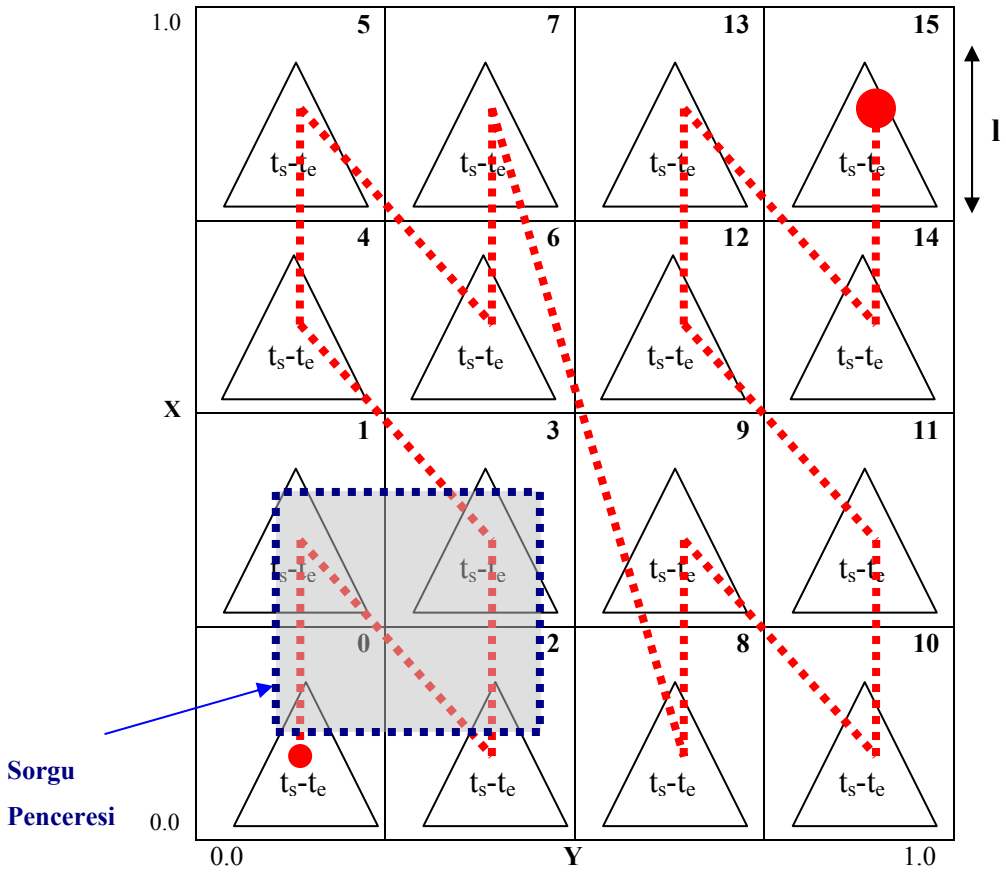
$$N_p^* = \sqrt{N_g^*} \tag{6.7}$$

6.3.4 Z-order sıralı SPIT yöntemi

Mallett'in çalışmasında uzay, sweep algoritmasına göre sıralanmıştır. Çalışmalarını bu şekilde yapmalarının nedeni, z-order fonksiyonunun endeks yaratma ve sorgu sırasında çok fazla hesaplama yükü getirmesinden dolayı, daha hızlı şekilde pid numaralarını atayan sweep algoritmasını kullanmaktır.

Mallett'in çalışmasından farklı olarak getireceği CPU yüküne rağmen bu tez çalışması kapsamında uzay; z-order algoritmasına göre doğrusal olarak sıralanmıştır. Daha önce de

belirtildiği gibi, belirlenen bir sorgu alanında disk üzerinde yakın olarak adreslenmiş sayfalardan bilgi alınabileceği ve böylece performans artışı elde edilebileceği düşünülmüştür. Şekil 6.11'de 4x4'lük bir grid için Z-order sıralaması verilmiştir.



Şekil 6.11 Z-order sıralı SPIT

Şekil 6.11'de tanımlanan bir sorgu penceresi içinde sadece birbirine yakın olan <0, 1, 2, 3> numaralı grid hücreler incelenmektedir.

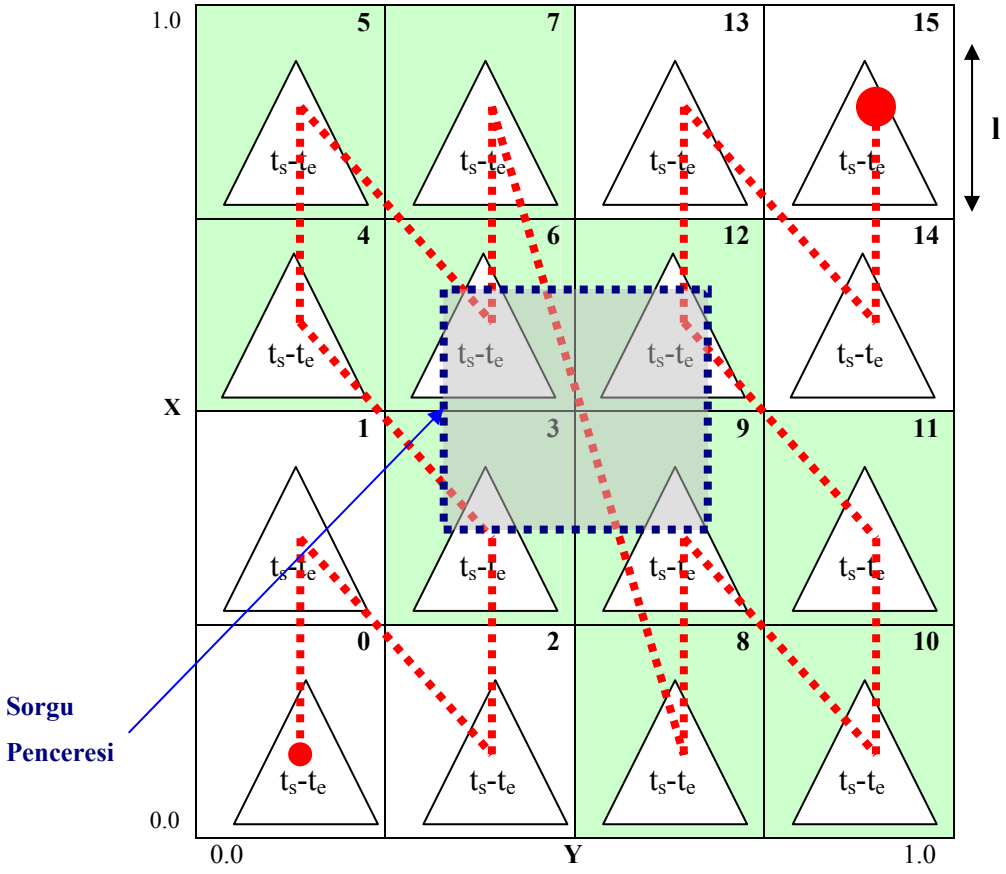
Doğrusal sıralama için kullanılan z-order uzay doldurma eğrisi algoritması Algoritma 4'te verilmiştir (Mallett, D. J. 2004). Bu algoritma, <x, y> değerlerini alarak z-değerini hesaplamaktadır. Bu konumda bulunan noktanın, 1-8 numaralı satırlar grid hücresini hesaplar. Algoritma 1'de verilen adımlar, bu algoritmada da aynı şekilde kullanılmıştır. Bit sayısı (num_bits), 9 numaralı satırda hesaplanmaktadır. Döngüyü gösteren 11-16 numaralı satırlarda z-değeri, x-grid ve y-grid değerlerine göre bulunmaktadır. İlgilenilen bit değeri, mask değişkeni ile seçilmektedir.

Algoritma 4 *z_order*() fonksiyonu

INPUT: <*x* ,*y*>
OUTPUT: <*x* ,*y*>yi içeren z_{value}

1: $x\_grid := \lfloor x \times N_p^* \rfloor$
2: $y\_grid := \lfloor y \times N_p^* \rfloor$
3: **if** *x=1.0* **then**
4: $x\_grid := x\_grid\text{-}1$
5: **end if**
6: **if** *y=1.0* **then**
7: $y\_grid := y\_grid\text{-}1$
8: **end if**
9: $num\_bits := \log_2 N_p^*$
10: $shift := num\_bits$
11: **for** *i*:=1 to *num_bits* **do**
12: $mask := 2^{num\_bits-i}$
13: $z_{value} := z_{value} + bitand(x\_grid, mask) * 2^{shift}$
14: $shift := shift - 1$
15: $z_{value} := z_{value} + bitand(y\_grid, mask) * 2^{shift}$
16: **end for**
17: **return** z_{value}

Şekil 6.11'den farklı olarak, Şekil 6.12'deki gibi bir sorgu penceresi seçildiğinde fazla CPU yükü getireceği için sorgulama problemleri oluşmaktadır.



Şekil 6.12 Z-order sıralı SPIT'te sorgulama

Şekil 6.12'de belirlenen sorgu penceresi içindeki en küçük z-order değeri 3, en büyük z-order değeri de 12'dir. Eğer sorgu 3 numaralı grid hücreden, 12 numaralı grid hücreye kadar sorgu penceresi ile çakışan alanı belirleme şeklinde tanımlanırsa, bu durumda 16 hücreden, 10 hücre boyunca sorgulama yapılacaktır. Bu da fazladan CPU yükü getirir. Bunun yerine sadece <3, 6, 9, 12> numaralı grid hücrelerde sorgulama yapmak sorgu performansını artıracaktır. Aşağıda belirtilen bu iyileştirilmiş sorgu algoritmaları açıklanmıştır.

Z-order'a göre grid'lere ayrılmış bir alan içinde, verilen bir sorgu penceresi ve zaman aralığı içinde <R ,T> aralık sorgusu Algoritma 5'te verilmiştir. Algoritma 2'den farklı olarak, 1 numaralı satır değişmiştir.

Algoritma 5 *st_zorder_sorgusu*() fonksiyonu

INPUT: <R ,T>
OUTPUT: oid_list
1: *pid_list* := *p_intersect_zorder*(R) // algoritma 7'de
2: **for all** *pid* in *pid_list* **do**
3: *oid_list* := *oid_list* ∪
4: SELECT *oid*
5: FROM partition(*pid*)
6: WHERE t_s between T.t_{min} - *MAX_TI* and T.t_{max}
7: AND t_e between T.t_{min} and T.t_{max} + *MAX_TI*
8: AND x between R.x_{min} and R.x_{max}
9: AND y between R.y_{min} and R.y_{max}
10: **end for**
11: *oid_list*'i sırala ve çift girişleri ayıkla
12: **return** *oid_list*

Herhangi bir sorgu penceresi içinde bulunan noktaların, verilen bir z-order aralığı ile kesiştiğini hesaplamak için, veritabanında ayrıca bir tabloda tüm gridde bulunan hücrelerin bulunduğu alan dikdörtgen geometrisi olarak saklanmaktadır. Bunun için döngü içinde her grid içinde bulunabilecek dikdörtgenin en büyük sağ köşesi <rectangle_u_x, rectangle_u_y> noktası ile belirlenmektedir. Daha sonra en küçük sol köşesi <rectangle_l_x, rectangle_l_y> noktası hesaplanarak, sdo_geometry nesnesi kullanılarak veritabanına eklenmektedir. Açıklanan pid_zorder alt programı Algoritma 6'da verilmiştir. Her grid hücrenin büyüklüğü (1 / N_p^*)x(1 / N_p^*) kadar olduğundan <rectangle_u_x, rectangle_u_y>'nin ilk değeri bu değere atanmış ve döngü içinde bu değer kadar artırılmış, sweep algoritmasındaki gibi uzay taranarak z-değerleri hesaplanmıştır.

Algoritma 6 *pid_zorder* altprogramı

```
1:  rectangle_u_x := 1 / N*p
2:  rectangle_u_y := 1 / N*p
3:  for i:= 0 to N*p do
4:     for j := 0 to N*p do
5:        zorder := z_order(rectangle_u_x, rectangle_u_y)
6:        rectangle_l_x := rectangle_u_x – ( 1 / N*p )
7:        rectangle_l_y := rectangle_u_y – ( 1 / N*p )
8:        rectangle :='mdsys.SDO_GEOMETRY(2003, null, null,sdo_elem_info_array('
                      ||'1,1003,3), sdo_ordinate_array('||rectangle_l_x||','
                      ||rectangle_l_y||','||rectangle_u_x||','||rectangle_u_y||'))';
9:        INSERT INTO ST_zorderNumber(zorder, rectangle)
             VALUES ('||zorder||','||rectangle||');')
10:    rectangle_u_x := rectangle_u_x + ( 1 / N*p )
11:    end for
12: rectangle_u_x := ( 1 / N*p )
13: rectangle_u_y := rectangle_u_y + ( 1 / N*p )
14: end for
```

Grid hücrelerle çakışan sorgu penceresini hesaplamak için Algoritma 7 kullanılmaktadır. ST_ZORDERNUMBER tablosundaki dikdörtgenlerle çakışan grid hücreler sdo_relate predicate'ı ile kontrol edilmekte, herhangi bir kesişim olduğunda pid_list'e eklenmektedir.

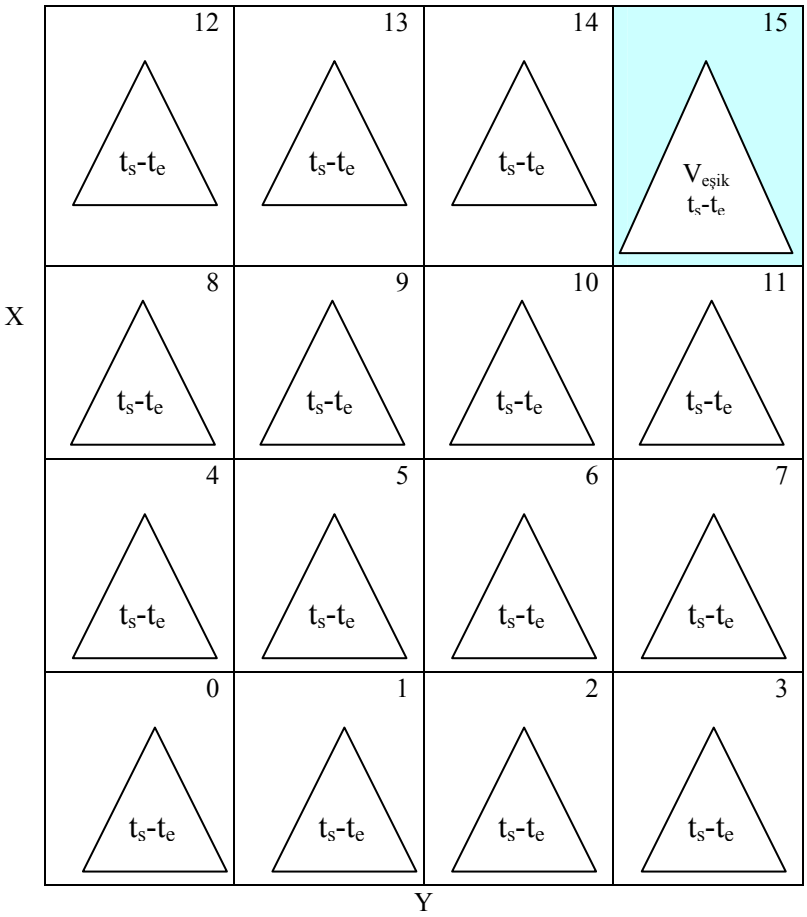
Algoritma 7 *p_intersect_zorder*() fonksiyonu

```
INPUT: R
OUTPUT: pid_list
   1: for record in
      ( select zorder from st_zorderNumber where sdo_relate( rectangle,
           mdsys.sdo_geometry(2003, null, null,
                              mdsys.sdo_elem_info_array(1,1003,3),
                              mdsys.sdo_ordinate_array(R.xmin, R.ymin, R.xmax, R.ymax)),
                              'mask=ANYINTERACT querytype=window')='TRUE') do
   2:       pid_list := pid_list ∪ record.zorder
   3: end for
   4: return pid_list
```

6.4 Hızlı Hareket Etmiş Nesneler için Endeksleme Yöntemi

Hareketli nesnelerin hız bilgisinin önemli olduğu uygulamalarda, bu tip nesnelere veritabanında ulaşabilmek için tüm uzayda, tüm grid hücrelerde arama yapılması gerekir. Bu tip uygulamalar için performansı artırıcı teknikler bu tez çalışmasında incelenmiştir. Önceki bölümlerde açıklanan endeks yapıları arasında uzayı bölümleme algoritmaları en etkin sorgu sonuçlarını hesapladığı için, hareket eden nesneler içinde gezingesinde yüksek hız yapmış olan nesneler, ayrı bir grid hücrede veri tabanında saklanması düşünülmektedir.

Mallett'in çalışmasında hareket eden nesnelerin hızları çalışmaya dahil edilmemiştir (Mallett, D.vd. 2005). Bu çalışmada, hızlı nesnelerin sayısının bir uygulamada az sayıda olduğu kabul edilerek, R-tree'nin performansını düşürmemek için ayrı bir grid hücrede saklanıp endeksleme işleminin gerçeleştirilmesi teklif edilmektedir (Şekil 6.13).



Şekil 6.13 Teklif edilen endeks yapısı

6.5 RDBMS İçinde Endekslerin Tanımlanması

Bu bölümde Oracle veri tabanı yönetim sisteminde SPIT yönteminin, z-order sıralı SPIT yönteminin, R-tree + B-Tree yönteminin ve Z-değerleri + B-Tree yönteminin tanımlaması verilecektir.

6.5.1 SPIT yönteminin gerçekleştirimi

Oracle ilişkisel veri tabanı yönetim sisteminde, bu algoritmalar PL/SQL kullanılarak gerçekleştirilebilir (Oracle, 2001b; Oracle, 2001c; Oracle, 2001a; Oracle, 2004b; Oracle, 2008b). Space Partitioning with Indexes on Time(SPIT) algoritmasının DDL ve SQL tanımlamaları aşağıda verilmiştir (Mallett, D. J. 2004).

Verilen "0.1-0.3 aralığında x ve 0.2-0.4 aralığında y ve 0.5'ten 0.6'ya kadar zaman aralığında bulunan hareketli nesneleri hesaplama" SQL sorgusu için Çizelge 6.2'deki tanımlamalar kullanılabilir. Burada sorgunun 4x4 grid içinde hesaplandığı varsayılmaktadır.

Çizelge 6.2 SPIT yöntemi için DDL ve SQL cümleleri

Tablo Yaratmak için DDL	CREATE TABLE ST_SPIT(oid INTEGER, x NUMBER, y NUMBER, t_s NUMBER, t_e NUMBER, pid INTEGER) PARTITION BY RANGE (pid) (partition p01 values less than (1), partition p02 values less than (2), partition p03 values less than (3), partition p04 values less than (MAXVALUE))
Endeks Yaratmak için DDL	CREATE INDEX idx_st_spit_t ON ST_SPIT(t_s, t_e) LOCAL
Örnek SQL Sorgusu	1: SELECT unique oid 2: FROM ST_SPIT 3: WHERE pid in (0,1,4,5) 4: AND t_s between 0.5 – MAX_TI and 0.6 5: AND t_e between 0.6 and 0.6 + MAX_TI 6: AND x between 0.1 and 0.3 7: AND y between 0.2 and 0.4

6.5.2 Z-order sıralı SPIT yönteminin gerçekleştirimi

Oracle ilişkisel veri tabanı yönetim sisteminde, bu algoritmalar PL/SQL kullanılarak gerçekleştirilebilir (Oracle, 2001b; Oracle, 2001c; Oracle, 2001a; Oracle, 2004b; Oracle, 2008b). Z-order sıralı SPIT algoritmasının DDL ve SQL tanımlamaları aşağıda verilmiştir (Mallett, D. J. 2004).

Verilen "0.1-0.3 aralığında x ve 0.2-0.4 aralığında y ve 0.5'ten 0.6'ya kadar zaman aralığında bulunan hareketli nesneleri hesaplama" SQL sorgusu için Çizelge 6.3'teki tanımlamalar kullanılabilir. Burada sorgunun 4x4 grid içinde hesaplandığı varsayılmaktadır.

Çizelge 6.3 Z-order sıralı SPIT yöntemi için DDL ve SQL cümleleri

Tablo Yaratmak için DDL	CREATE TABLE ST_SPIT_ZORDER(oid INTEGER, x NUMBER, y NUMBER, t_s NUMBER, t_e NUMBER, pid INTEGER) PARTITION BY RANGE (pid) (partition p01 values less than (1), partition p02 values less than (2), partition p03 values less than (3), partition p04 values less than (MAXVALUE))
Endeks Yaratmak için DDL	CREATE INDEX idx_st_spit_zorder_t ON ST_SPIT(t_s, t_e) LOCAL
Örnek SQL Sorgusu	1: SELECT unique oid 2: FROM ST_SPIT_ZORDER 3: WHERE pid in (0,1,2,3) 4: AND t_s between 0.5 – MAX_TI and 0.6 5: AND t_e between 0.6 and 0.6 + MAX_TI 6: AND x between 0.1 and 0.3 7: AND y between 0.2 and 0.4

6.5.3 R-tree + B-Tree yönteminin gerçekleştirimi

R-tree + zamana bağlı B-tree endeks yapısının DDL ve SQL tanımlamaları aşağıda verilmiştir (Mallett, D. J. 2004). Çizelge 6.4'te, SPIT yönteminde açıklanan sorgunun R-tree + B-tree yönteminde gerçekleştirimi gösterilmiştir.

Çizelge 6.4 R-tree + B-Tree yöntemi için DDL ve SQL cümleleri

Tablo Yaratmak için DDL	CREATE TABLE ST_RTREE(oid INTEGER, position MDSYS.SDO_GEOMETRY, t_s NUMBER, t_e NUMBER,)
Endeks Yaratmak için DDL	CREATE INDEX idx_st_rtree ON ST_RTREE(position) INDEXTYPE IS MDSYS.SPATIAL_INDEX CREATE INDEX idx_st_rtree_t ON ST_RTREE (t_s, t_e)
Örnek SQL Sorgusu	1: SELECT unique oid 2: FROM ST_RTREE 3: WHERE sdo_relate(4: position, 5: MDSYS.SDO_GEOMETRY(2003, NULL, NULL, MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 3) MDSYS.SDO_ORDINATE_ARRAY(0.1, 0.2, 0.3, 0.4)), 6: 'mask=ANYINTERACT querytype=window') = 'TRUE' 7: AND t_s between 0.5 – MAX_TI and 0.6 8: AND t_e between 0.6 and 0.6 + MAX_TI

Çizelge 6.4'te sdo_relate, 4 numaralı satırdaki hareketli nesnenin konumunun ve 5 numaralı satırdaki sorgu penceresinin kesiştiği yerleri hesaplayarak sorgu sonucunu hesaplar. B-tree endeksi tanımlanarak zamana bağlı aramalar gerçekleştirilir.

6.5.4 Z-değerleri + B-Tree yönteminin gerçekleştirimi

Z-değerleri + B-tree endeks yapısının DDL ve SQL tanımlamaları Çizelge 6.5'te verilmiştir (Mallett, D. J. 2004).

Çizelge 6.5 Z-değerleri + B-Tree yöntemi için DDL ve SQL cümleleri

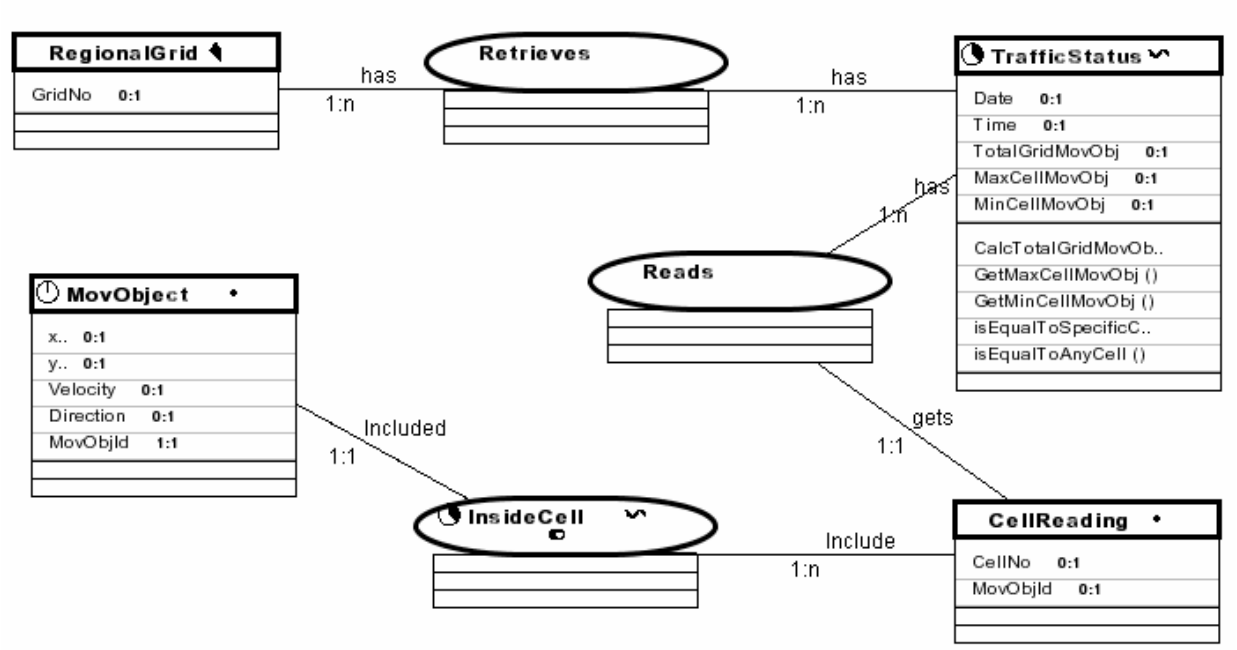
Tablo Yaratmak için DDL	CREATE TABLE ST_ZORDER(oid INTEGER, x NUMBER y NUMBER t_s NUMBER, t_e NUMBER,)
Endeks Yaratmak için DDL	CREATE INDEX idx_st_zorder ON ST_ZORDER(t_s, t_e, z_order(x, y)) CREATE INDEX idx_st_zorder ON ST_ZORDER(z_order(x, y), t_s, t_e)
Örnek SQL Sorgusu	1: SELECT unique oid 2: FROM ST_ZORDER 3: WHERE z_order(x, y) between 0 and 3 4: AND t_s between 0.5 – MAX_TI and 0.6 5: AND t_e between 0.6 and 0.6 + MAX_TI 6: AND x between 0.1 and 0.3 7: AND y between 0.2 and 0.4

Z-değeri endeksleme için kullanılacağı için, ayrıca bir sütunda saklanmamıştır. Bunun yerine verilerin tabloya eklenmesinden sonra, fonksiyonel endeks yaratılmıştır. Çizelge 6.5'te sorgunun 3 numaralı satırında, sorgu penceresinin en küçük ve en büyük z-order değeri verilmektedir. Yaratılan iki endeks t z index and z t index arasındaki fark temel endeks filtresinin zamana veya uzaya bağlı olduğudur.

6.6 Örnek Veri Kartuşu

Güting (Düntgen, C.vd. 2008) ve Wolfson'un (Wolfson, O.vd. 1998) yaklaşımlarını temel alarak Pelekis ve arkadaşları, hareket eden nesneler için HERMES* isimli bir yapı kurmuşlardır. Bu veri kartuşu, Oracle'ın statik veri tiplerini genişletir ve TAU-TLL zaman veri kartuşunu kullanmaktadır (Pelekis, N.ve Theodoulidis, B. 2002).

Veri kartuşunun nasıl oluşturulacağını gösteren Oracle referansındaki örnek uygulama (Oracle 2008a), hareketli nesneler üzerine uyarlanmıştır. Oluşturulan modelin MADS veri modeli gösterimi Şekil 6.14'te verilmiştir. Gridler üzerinde, trafik nesneleri hareket etmektedir. Kavramsal veri modeli verilen bu örnek SQL/Plus ortamında çalıştırılmıştır. Aşağıda gösterilen her nesne veri tabanı sisteminde tanımlanmış, domain endeks oluşturulmuş ve sorgular gerçekleştirilmiştir.



Şekil 6.14 Tasarlanan hareket eden nesneler sistemi için MADS modeli

* Daha fazla bilgi için; (Pelekis, N.ve Theodoridis, Y. 2006; Pelekis, N.vd. 2006; Pelekis, N.ve Theodoridis, Y. 2007; Pelekis, N.vd. 2008)

6.7 Değerlendirme

Bu bölümde geliştirilen MOORA sorgu sistemi için endeks yapıları oluşturulmuştur. Deneylerde kullanılan endeks yapıları; SPIT yöntemi, R-tree + zamana bağlı B-tree ve Z-değerleri + B-tree'dir. Bu yaklaşımlarda hareket eden nesnelerin hız bilgisi kullanılmamıştır.

Tez çalışması kapsamında araştırılan bu endeks yapıları incelendiğinde, z-değerleri + B-tree algoritmasının istenen sorgu penceresi alanı dışında daha geniş alanları taradığı görülmüştür. Bu nedenle, istenen sorgu penceresi alanı içinde hareketli nesnelerin araştırılması için, her grid hücrenin z-order değeri ayrı bir tabloda saklanmıştır. Daha sonra, sorgu penceresi ile çakışan hücrelerin z-order değerleri veri tabanından alınarak, yeni bir algoritma geliştirilmiş ve performans artışı elde edilmiştir.

Bu tez çalışması kapsamında gerçekleştirilen bilime bir katkı da, SPIT bölümleme algoritmasını z-order sıralamasına uygun olarak gerçekleştirmektir. Z-order sıralı SPIT algoritması da bu bölümde açıklanmıştır.

Hareket eden nesnelerin verileri, bir çok çalışmada da kullanılan Brinkhoff'un veri üreteci ile oluşturulmuştur (Pfoser, D.ve Jensen, C. S. 2003; Zhang, W.vd. 2006; Chen, S.vd. 2008).

Bir trafik simülasyonunda, gezingesinin bir bölümünde hızlı hareket etmiş araçların sayıları az olduğu için bu nesnelerin ayrı bir bölümde (partition) endekslenmesi bu çalışmada teklif edilmiştir.

Bölümleme konusunda gerçekleştirilen bir araştırmada ise, bölümler aktif ve aktif olmayan nesneler için oluşturulmuştur (Cui, B.vd. 2005).

Bu bölümde son olarak sorgu sistemini gerçekleştirmek için oluşturulan örnek veri kartuşu özellikleri ve işlemleri MADS kavramsal modeli ile birleştirilmiştir.

7. DENEYLER

Bir hem uzaya hem zamana bağlı aralık (range) sorgusu Q, Q = <R ,T> şeklinde gösterilir. Burada R, uzaya bağlı bir alanı ve T de zaman aralığını ifade eder. Q, R içinde bulunan <x, y> noktalarını ve T ile kesişen <t_s, t_e> zamanları içeren kayıtların farklı nesne id'lerini geri göndermektedir. Bu tez çalışmasında incelenen problem, hem uzaya hem zamana bağlı verilerin endekslenmesi ve en etkin şekilde aralık sorgu sonucunun elde edilmesidir.

Tüm deneyler Windows XP işletim sistemi kurulu, Intel Pentium Centrino işlemcili, 1400 MHz hızlı, 2.0 GB bellekli, 60 GB sabit diskli Sony Vaio dizüstü bilgisayar üzerinde gerçekleştirilmiştir. Veritabanı olarak "Oracle Database 10g Enterprise Edition Release 10.1.0.2.0" bilgisayara kurulmuştur.

Bu bölümde MOORA sorgu sistemi kapsamında incelenen endeksleme yapıları ve MOORA veri kartuşu deneyleri sonuçları verilmiş ve tartışılmıştır.

7.1 Tezde Kullanılan Hem Uzaya Hem Zamana Bağlı Veri

Araştırmalarda iki tip hem uzaya hem de zamana bağlı veri tabanları tanımlanmıştır. Bunlar, geçmişe veya mevcut/gelecek zamana ait iki grupta tanımlanmış sorguları gerçekleştirilmek için kullanılmıştır. Bu tez çalışması kapsamında, geçmişe ait toplanmış veriler ile hem uzaya hem de zamana bağlı veritabanı oluşturulmuştur. Bu verilerin belirli aralıklarla GPS ile veritabanına <$movObj_{id}$, x, y, t_s, t_e, v> şeklinde gönderildiği varsayılmaktadır. Burada,

- $movObj_{id}$, hareketli nesnenin belirleyici numarasını,
- <x, y>, konum koordinatlarını,
- <t_s, t_e>, hareketli nesnenin <x, y> konumunda kaldığı zaman aralığını,
- v, hareketli nesnenin hızını göstermektedir.

Verilerin bu şekilde alınmasını sağlayacak bir sistemin kurulup, elde edilmesi yerine tez çalışması kapsamında bu verilerin pek çok çalışmada kullanılmış olan bir üreteç ile elde edilmesi araştırılmıştır. Hareketli nesnelerin sayısal bir harita üzerinde hareket etmesi ve bu haritanın da İstanbul haritası olmasına karar verilmiştir. İnternet üzerinde bu şekilde detaylı bir sayısal harita bulunamamıştır. İstanbul Büyükşehir Belediyesinin sayısal harita kullanarak gerçekleştirdiği uygulamalar incelenmiştir. Belediyenin, sayısal harita üzerinde geliştirdiği şehir rehberi Ek 3'te tanıtılmıştır. Eminönü-İstanbul sayısal haritası da sonuç olarak, Belediyeden temin edilmiştir.

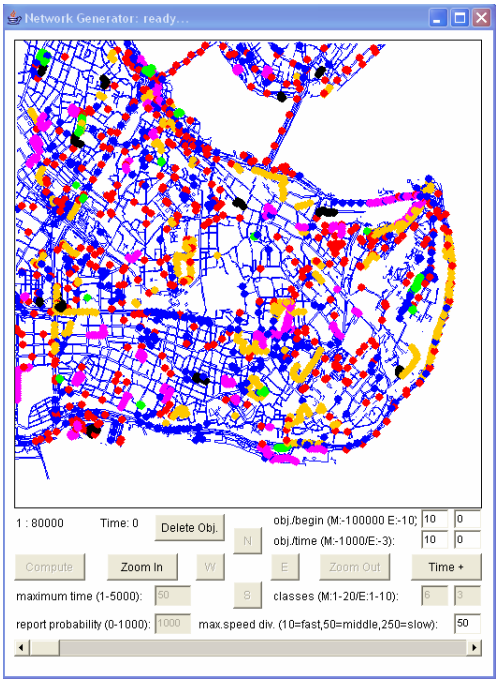
Tüm bu koşullara uygun hareket eden nesneler için veriler, gerçeğe uygun olarak tez çalışmasında hem uzaya hem zamana bağlı Brinkhoff'un geliştirdiği trafik üreteci (Brinkhoff, T. 2000) ile Eminönü-İstanbul haritası üzerinde üretilmiştir. Üretecin parametreleri değiştirilerek ağ üzerinde hareket eden farklı veriler oluşturularak deneyler gerçekleştirilmiştir.

Tezde kullanılmış veriler Brinkhoff'un üreteci ile üretilirken, baslangıçta 170000 nesnenin olması ve her zaman diliminde 30000 nesnenin hareket etmesi sağlanmıştır. Üretecin hareket eden nesneleri ürettikten sonra, verdiği bilgilerden önemli olanlar Çizelge 7.1'de verilmiştir.

Çizelge 7.1 Brinkhoff'un trafik üreteci ile üretilmiş veriler

Uzaya Zamana Bağlı Veri
data space width: 29854
data space height: 29999
number of nodes: 12013
number of edges: 8590
maximum time: 20
moving objects: 770000

Hareket eden nesneler böylece 29854x29999 veri uzayında oluşturulmuştur. Tüm deneyler sırasında veri uzayının yaklaşık 30000x30000 olduğu varsayılarak, grid hücreler oluşturulmuştur. Eminönü sayısal haritası kullanılarak oluşturulan hareketli nesneler veri kümesi Şekil 7.1'de gösterilmiştir.



Şekil 7.1 Network-based Generator'ın ürettiği veri kümesi

Brinkhoff'un trafik üreteci ile yukarıdaki parametrelere uygun veri üretildiğinde, 770000 hareketli nesne ve yaklaşık 5 milyon kayıt oluşturulmuştur. Bu veriler SQL*Loader ile veri tabanına aktarılmıştır. Deneyler sırasında farklı veri büyüklükleri (data cardinality) kullanılmıştır. Bunlar sırasıyla 1 milyon, 2.5 milyon ve 5 milyon kayıtlı verilerdir. Üreteç ile oluşturulan bu verilerden belirli hareket eden nesne numaraları seçilerek belirtilen bu veri büyüklüklerinde kümeler oluşturulmuştur (Çizelge 7.2).

Çizelge 7.2 Deneylerde kullanılan farklı veri kümeleri

Veri Büyüklüğü	**Hareket Eden Nesne Sayısı**	**Kayıt Sayısı**
1 Milyon	120.250	1.000.378
2.5 Milyon	306.400	2.500.464
5 Milyon	770.000	5.016.309

Mallett ise çalışmasında 1.5 milyon, 3 milyon ve 6 milyon büyüklüğünde veri kümeleri kullanmıştır. Her kümede sırasıyla 15000, 30000 ve 60000 hareketli nesne 100 zaman biriminde hareket ettirilmiştir (Mallett, D.vd. 2005). Mallett daha sonraki çalışmasında ise 1 milyon, 2.5 milyon ve 5 milyon büyüklüğünde veri kümeleri kullanmıştır. Her kümede sırasıyla 10000, 25000 ve 50000 hareketli nesne 100 zaman biriminde hareket ettirilmiştir (Botea, V.vd. 2008). Bu tez çalışmasında da, Mallett'in son çalışması esas alınarak veri kümeleri oluşturulmuştur.

Önceki bölümde açıklanan tablolara veri ekleme ve endeks yapılarını oluşturmak için geçen süre Çizelge 7.3'te verilmiştir.

Çizelge 7.3 Veri ekleme ve endeks yaratma süreleri (dk:sn)

Tablo Adı	**Endeks**	**Zaman**	**Veri Kümesi**		
			1M	**2.5M**	**5M**
ST_ZORDER	t_z index	insert	03:26	04:56	07:32
		index	01:28	04:05	09:23
		toplam	**04:54**	**09:01**	**16:55**
ST_ZORDER	z_t index	insert	03:26	04:56	07:32
		index	01:31	04:18	08:58
		toplam	**04:57**	**09:14**	**16:30**
ST_RTREE	R-tree	insert	02:36	06:08	08:44
		index	00:39	01:36	03:08
		toplam	**03:15**	**07:44**	**11:52**
ST_SPIT	B-tree	insert	02:32	06:48	14:49
		index	00:35	01:33	03:14
		toplam	**03:07**	**08:21**	**18:03**

Farklı sayıda partition yaratılırken veri ekleme ve endeks yaratma süreleri Çizelge 7.4'te gösterilmiştir.

Çizelge 7.4 Değişken partitionlarda veri ekleme ve endeks yaratma süreleri(dk:sn)

Partition Sayısı	İşlem	Veri Kümesi		
		1M	2.5M	5M
1	ekleme	01:38	04:25	10:52
	endeks	00:26	01:08	02:11
	toplam	**02:04**	**05:33**	**13:03**
25	ekleme	02:06	05:01	12:38
	endeks	00:34	01:32	02:47
	toplam	**02:40**	**06:33**	**15:25**
100	ekleme	02:21	06:05	12:34
	endeks	00:31	01:31	03:22
	toplam	**02:52**	**07:36**	**15:56**
169	ekleme	02:32	06:48	14:49
	endeks	00:35	01:33	03:14
	toplam	**03:07**	**08:21**	**18:03**
225	ekleme	02:53	10:20	17:49
	endeks	00:34	02:02	03:28
	toplam	**03:27**	**12:22**	**21:17**
400	ekleme	03:49	10:12	21:12
	endeks	00:53	01:51	03:21
	toplam	**04:42**	**12:03**	**24:35**
1600	ekleme	05:43	16:44	39:26
	endeks	02:16	02:55	05:17
	toplam	**07:59**	**19:39**	**44:43**

7.2 Deneyler için Oracle Veri Tabanı Yapılandırması

İlk deneyler gerçekleştirildikten sonra bilgisayarın performansını artırmak için bir dizi Oracle ayarı yapılmıştır. Deneyler sırasında kullanılan dizüstü bilgisayar açıldığında, Oracle'ın hemen 1.0 GB belleği tahsis etmesi Oracle Enterprise Manager'dan sağlanmıştır. İkinci en önemli ayar da deneylerde kullanılan tüm tablolar için ayrı bir tablespace ve endeksler için ayrı bir tablespace yaratılmasıdır. Böylelikle sorgu işleme sırasında birbirlerini beklemedikleri için Mallett'in çalışmasından daha iyi deney sonuçları elde edilmiştir.

Bu bölümde açıklanan deneyleri gerçekleştirmenin amaçları aşağıda sunulmuştur:

- Normal dağılım için oluşturulmuş maliyet modeline göre, SPIT algoritmasını gerçekleştirme ve farklı veri büyüklüklerinde ve partition sayılarında değerlendirme,
- SPIT endeksleme yöntemlerini diğer endeksleme yöntemleri ile karşılaştırma,
- İstanbul haritası üzerinde hareket eden nesneler için SPIT algoritması gerçeklendiğinde, en uygun partition sayısını deneysel olarak belirleme,
- Hızlı hareket etmiş nesneler için en etkin endeks yapısını oluşturma,
- MOORA veri kartuşunu oluşturarak hareket eden nesneler için veri tipleri yaratma ve sorgu dili oluşturma.

7.3 Deneylerde Kullanılan Yöntemler

Deneylerde iki boyutlu veri uzayının %0.25'i, %1'i ve %4'ü sorgulanmıştır. %4'lük sorgu her boyuttaki %20'lik seçimi göstermektedir. 30000x30000 veri uzayında %4'lük uzayda sorgu, 6000x6000'lik bir sorgu penceresi tanımlanarak gerçekleştirilmiştir. Zaman için de %5, %10 ve %20'lik zamana bağlı sorgulamalar gerçekleştirilmiştir. 20 zaman birimi içinde %20'lik alan 4 zaman birimi değişimi ile gerçekleştirilmiştir.

Çizelge 7.5'te uzaya ve zamana bağlı sorgulamalar gerçekleştirildiğinde, sorgu sonucu olarak elde edilen hareketli nesne sayısı verilmektedir. Diğer bir ifade ile veri kümelerinin büyüklüklerinden bağımsız olarak yaklaşık %0.7 - %0.9 arasında kayıtlar veritabanından okunmuş ve yürütme planları 100 sorgunun çalışması sırasında diske yazılmıştır. Zamana bağlı sorgularda sorgu yüzdesinin değişmesi, hareketli nesne sayısını değiştirmemiştir (Çizelge 7.5). Bunun nedeni ise veri kümesinin kısa zaman aralığında (20) oluşturulmasıdır.

Çizelge 7.5 Sorgulama sonucunda okunan hareketli nesne sayısı

		Kayıt Sayısı		
		1 Milyon	2.5 Milyon	5 Milyon
Uzay	s %0.25	105	385	912
	s %1	393	1379	3282
	s %4	1646	5453	12850
Zaman	t %5	393	1379	3282
	t %10	393	1379	3282
	t %20	393	1379	3282

Her endeks yapısı için 100 adet sorgu yürütülerek, bunların ortalamaları alınmıştır. Sorgunun yürütme zamanını ölçmek için Oracle'ın timing komutu kullanılmıştır. Disk okuma için Oracle'ın set autot komutu ile oluşturulan istatistiklerden "physical reads" ve bellek okuma için de "consistent gets" ortalamaları alınmıştır. Her 100 sorgu yürütüldükten sonra DBMS'in tamponları temizlenerek, sorgu performansını etkilemeleri önlenmiştir.

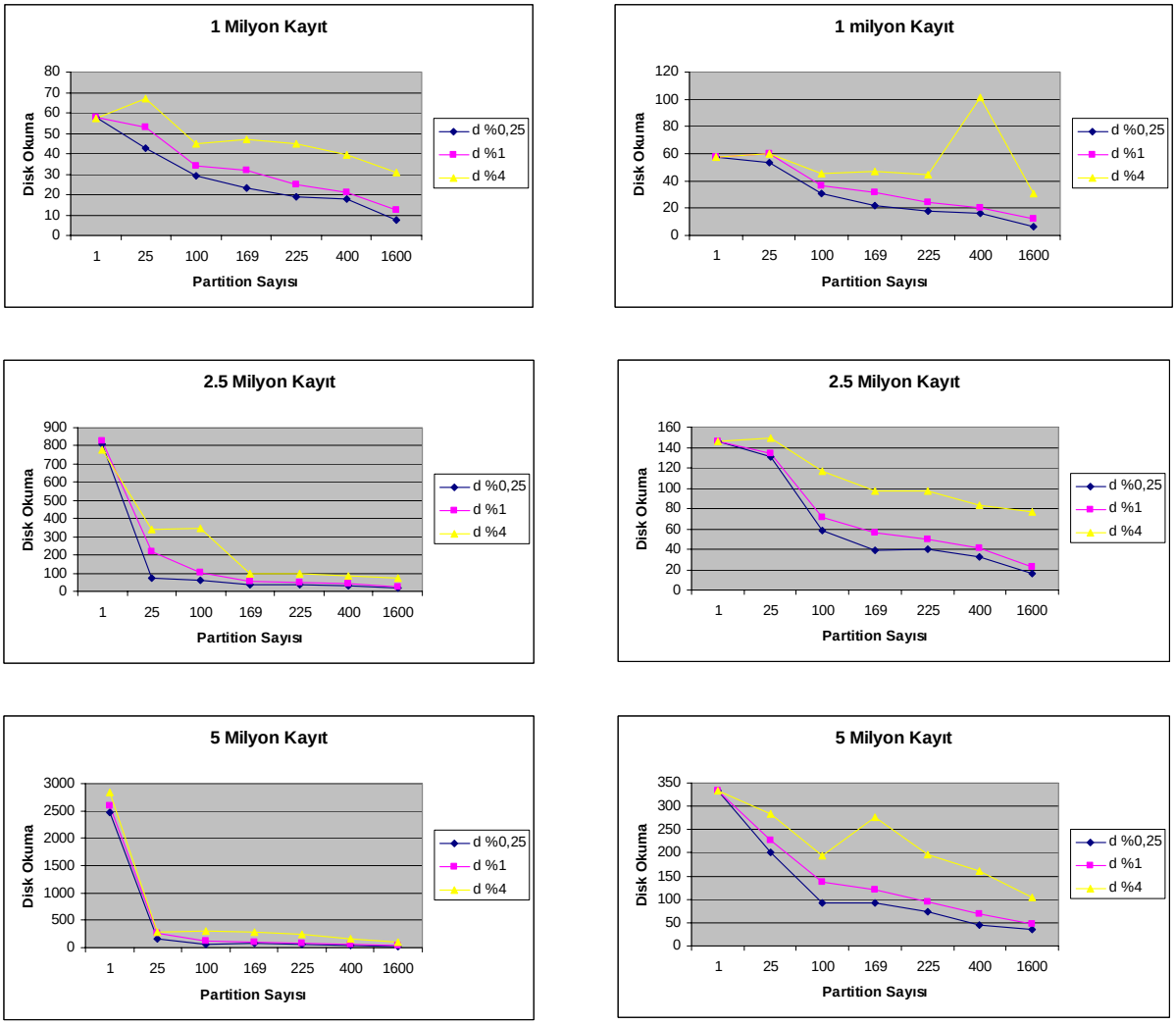
Aşağıdaki bölümlerde gerçekleştirilen deneyler; Oracle ayarları yapılmadan önce ve sonra gerçekleştirilen deneyler, rastgele sorgu pencereleri ile deneyler, aynı sorgu penceresi ile deneyler, zaman aralığı değiştirilerek oluşturulan deneyler, z-order sıralı SPIT deneyleri, z-order deneyleri ve veri kartuşu deneyleri açıklanmıştır.

7.4 Oracle Ayarlarının Deneylere Etkisi

Gerçekleştirilen ilk deneylerde en yüksek disk okuma sayısı, Mallett'in deneylerinden daha yüksek olmuştur (1200). Bunu iyileştirmek için açıklanan Oracle ayarları yapılmış ve aynı deneyler tekrar gerçekleştirilmiştir. Şekil 7.2'de farklı sayıda partition (grid hücre) seçilerek, elde edilen disk erişimleri üzerinde Oracle ayarlarının deneylere etkisi gösterilmiştir. 5 milyon kayıt için Oracle ayarları yapılmadan en fazla 2840 disk erişimi olmuşken, ayarlar yapıldıktan sonra en fazla 333 disk erişimi olmuştur. Böylece yaklaşık %89 iyileştirme gerçekleştirilmiştir.

a) Oracle Ayarları Yapılmadan Önce

b) Oracle Ayarları Yapıldıktan Sonra



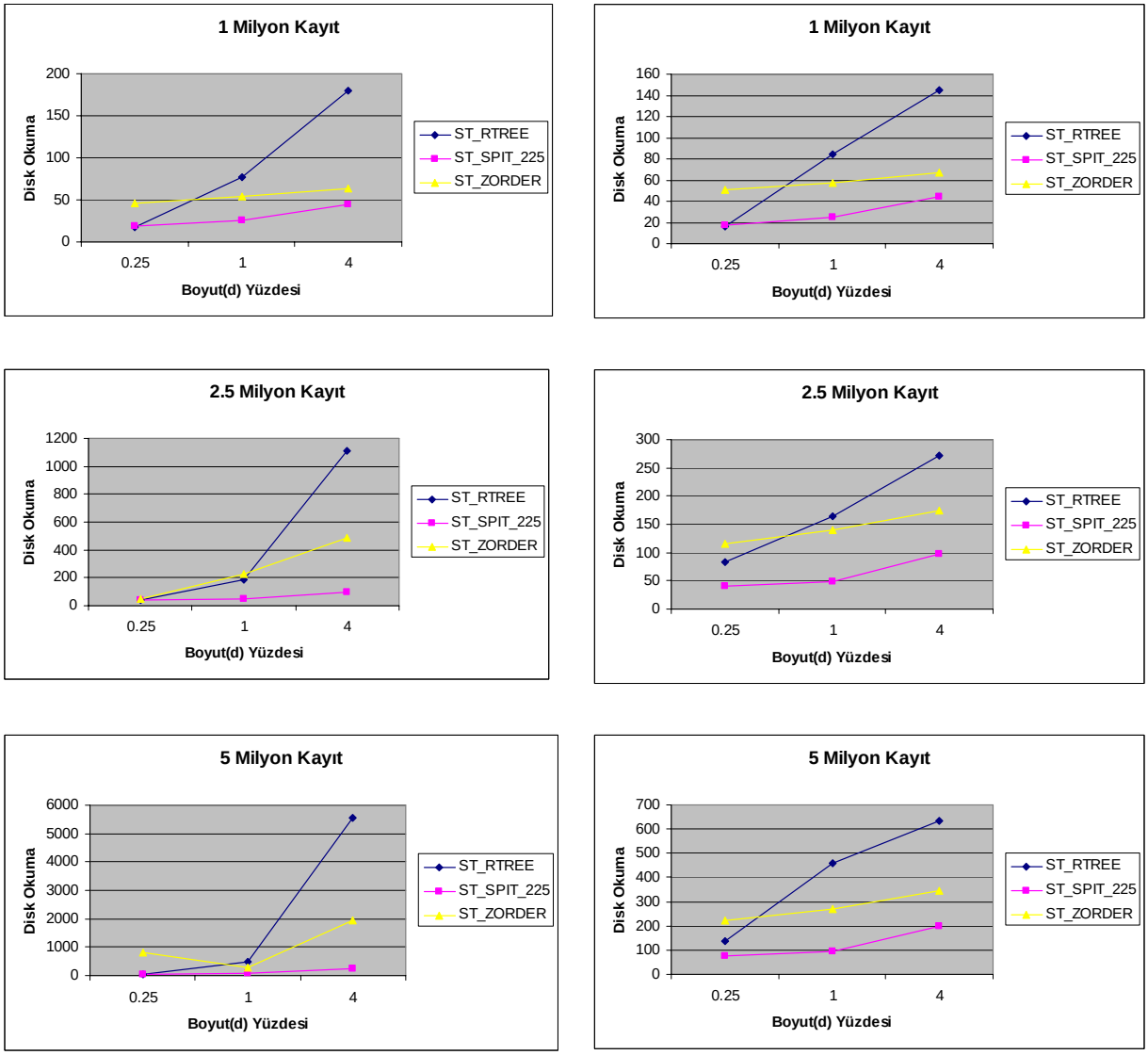
Şekil 7.2 Farklı sayıda grid hücre için disk erişimleri

İlk deneylerde iki boyutlu veri uzayı yerine tek boyutlu veri uzayının %0.25'i, %1'i ve %4'ü büyüklüğünde sorgu pencereleri harita üzerinde rastgele oluşturulmuştur. Sorgu penceresi

büyüklükleri 75x75, 300x300 ve 1200x1200 olarak hesaplanmıştır. Sorgu büyüklükleri, Şekil 7.2'de d %0.25 olarak gösterilmiştir. Boyut (dimension), d ile belirtilmiştir. Sorgu pencerelerinin rastgele yaratılmasından dolayı grafikte bazen büyük artışlar oluşmuştur. Şekil 7.2'de uzayın en iyi 169 grid hücreye bölünmesi gerektiği görülmektedir. Fakat tüm deneylerde SPIT algoritmasına göre uzayın 225 grid hücreye bölünmesi durumundaki ölçümler grafiklerde karşılaştırılmıştır. Şekil 7.3'te R-tree, SPIT ve ZORDER performansları farklı sorgu büyüklüklerinde gösterilmiştir. R-tree çok fazla disk erişimi ve bellek erişimi gerektirmektedir. SPIT'in maliyetinin en düşük olduğu grafiklerden görülmektedir.

a) Oracle Ayarları Yapılmadan Önce

b) Oracle Ayarları Yapıldıktan Sonra

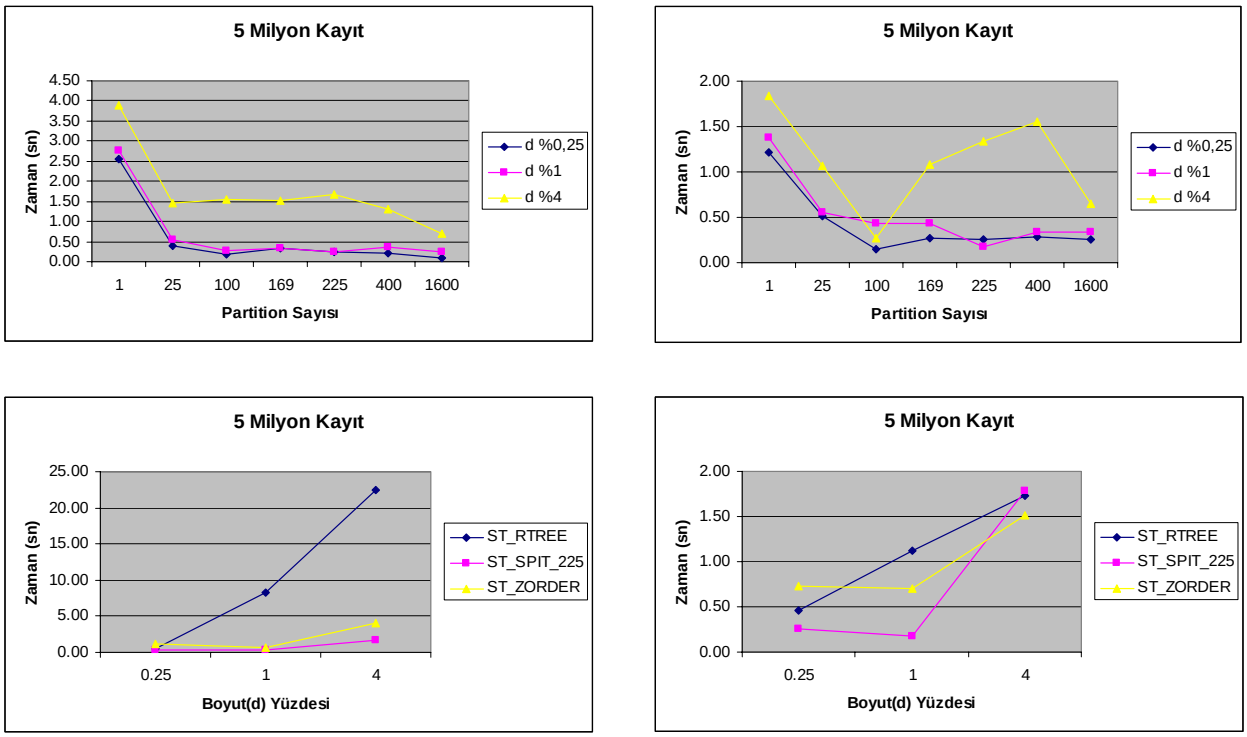


Şekil 7.3 Farklı sorgu büyüklüklerinde disk erişimleri

Mallett deneylerini 4-işlemcili 32 GB bellekli IBM p690 sistemi üzerine kurulu Oracle 9.2i Enterprise Edition'da gerçekleştirmiştir. Sunucu bilgisayarı üzerinde daha hızlı erişim yapan sabit diskler bulunmakta olduğu hatırlandığında, deneyleri gerçekleştirdiğimiz dizüstü bilgisayarının performansının iyi olduğu değerlendirilmektedir. Mallett farklı sayıda partitionlar ve normal dağılımı olmayan 6 milyon kayıtlı veri üzerinde en yüksek 1200 dolayında disk erişimi ve 2 saniye üzerinde sorgu erişim süresi ölçmüştür. Şekil 7.4'te farklı partitionlarda ve ve farklı sorgu yüzdelerinde sorgu işleme süreleri karşılaştırılmıştır.

a) Oracle Ayarları Yapılmadan Önce

b) Oracle Ayarları Yapıldıktan Sonra

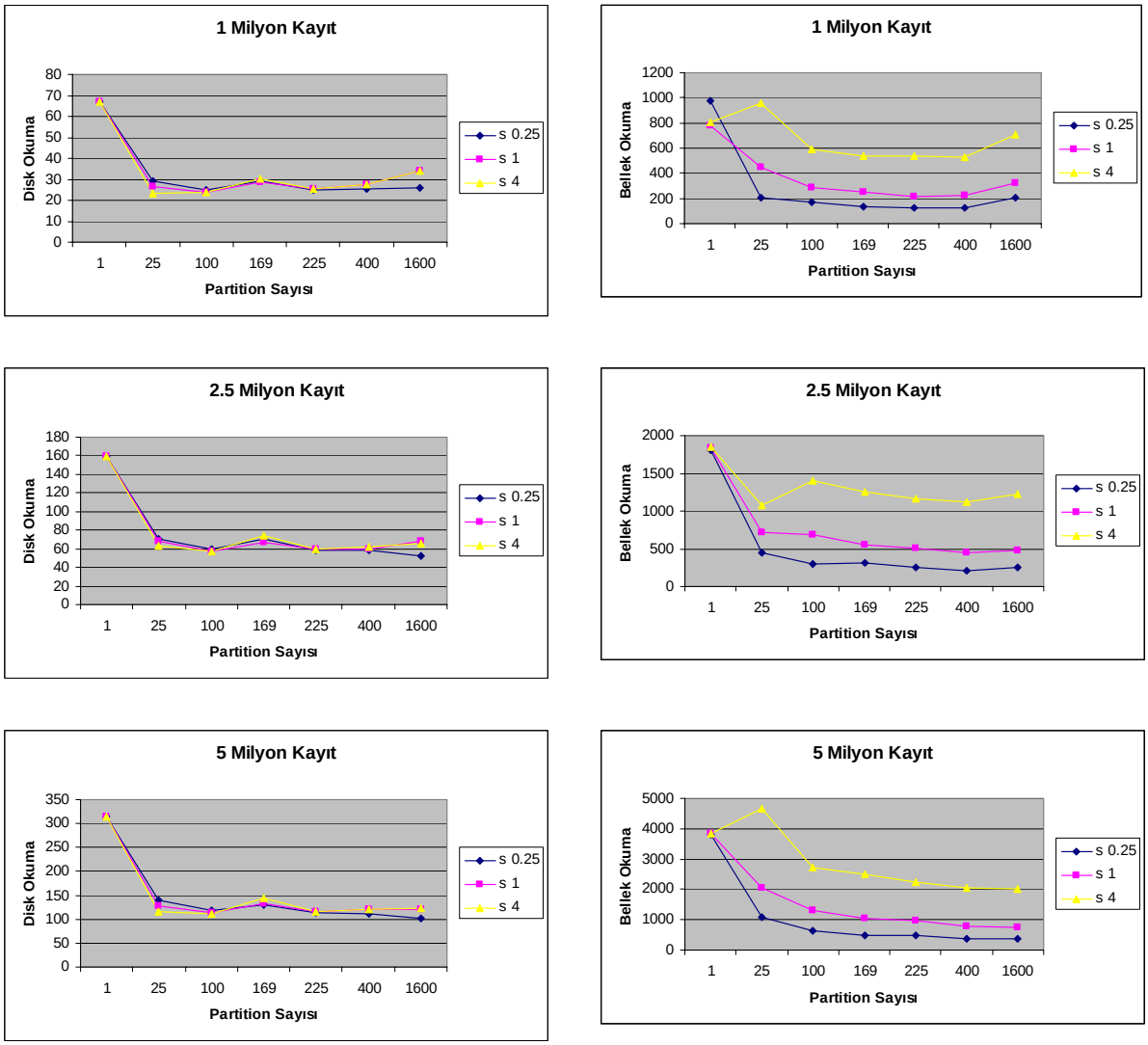


Şekil 7.4 Sorgu işleme zamanları

Bu sorgu sonuçlarında 1 milyon kayıt için %0.25 sorgu uzayında ortalama 0 kayıt; %1 sorgu uzayında ortalama 2 kayıt ve %4 sorgu uzayında ortalama 2 kayıt okunmuştur. 2.5 milyon büyüklüğünde veri kümesinde ise %0.25 sorgu uzayında ortalama 7 kayıt; %1 sorgu uzayında ortalama 22 kayıt ve %4 sorgu uzayında ortalama 45 kayıt okunmuştur. Son olarak, 5 milyon büyüklüğünde veri kümesinde ise %0.25 sorgu uzayında ortalama 87 kayıt; %1 sorgu uzayında ortalama 250 kayıt ve %4 sorgu uzayında ortalama 519 kayıt okunmuştur.

7.5 Rastgele Sorgu Penceresi Oluşturularak Gerçekleştirilen Deneyler

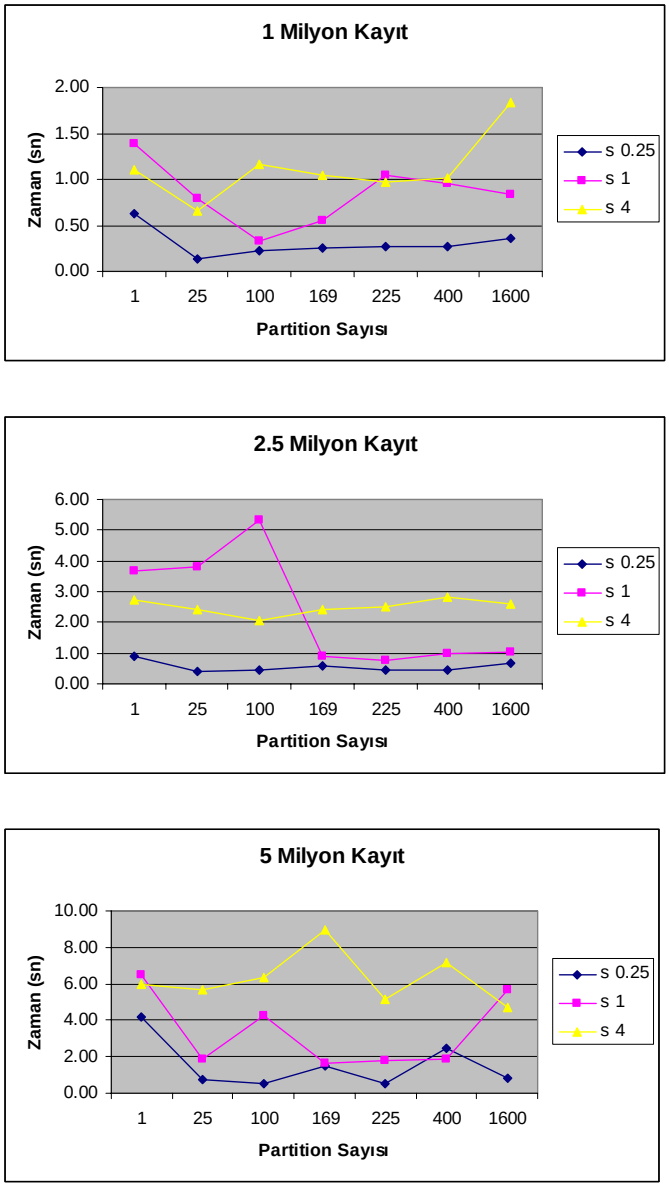
Rastgele sorgu pencereleri oluşturulurken, sorgu penceresinin sağ üst köşesi rastgele olarak oluşturulmuş ve sol alt köşesi sorgu alanının yüzdesine göre belirlenmiştir. Veri uzayının (data space-s) %0.25' i için 1500x1500, %1'i için 3000x3000 ve %4'ü için de 6000x6000 büyüklüğünde sorgu pencereleri oluşturulmuştur. Zaman aralığı 2 birim değişmiştir.



Şekil 7.5 Rastgele sorgu penceresi için farklı partitionlarda disk ve bellek erişimleri

Şekil 7.5'te farklı sayıda partition'lar için verilen disk okumalarının, sorgu uzayı değişmesine rağmen hemen hemen aynı olduğu görülmüştür. Ayrıca s %1 için en iyi partition sayısının veri tabanındaki kayıt sayısının değişmesine rağmen her zaman 100 olduğu deneylerle ispatlanmıştır. Partition sayısının en iyi bölümlemeden çok az veya çok yüksek olması durumunda disk okumalarının artma eğiliminde olduğu değerlendirilmektedir. Partition sayısı arttıkça, bellek okumalarının azalma eğiliminde olduğu görülmektedir.

Rastgele sorgu pencereleri ile çakışan hareketli nesneleri belirleme sorgularının erişim süreleri Şekil 7.6'da verilmiştir. Grafiklerdeki düzensizlikler, sorgu pencerelerinin rastgele oluşturulmasından kaynaklanmaktadır. Sorgu uzayının artması ile, hareket eden nesne sayısının artması sorgu erişim sürelerini arttırmaktadır.

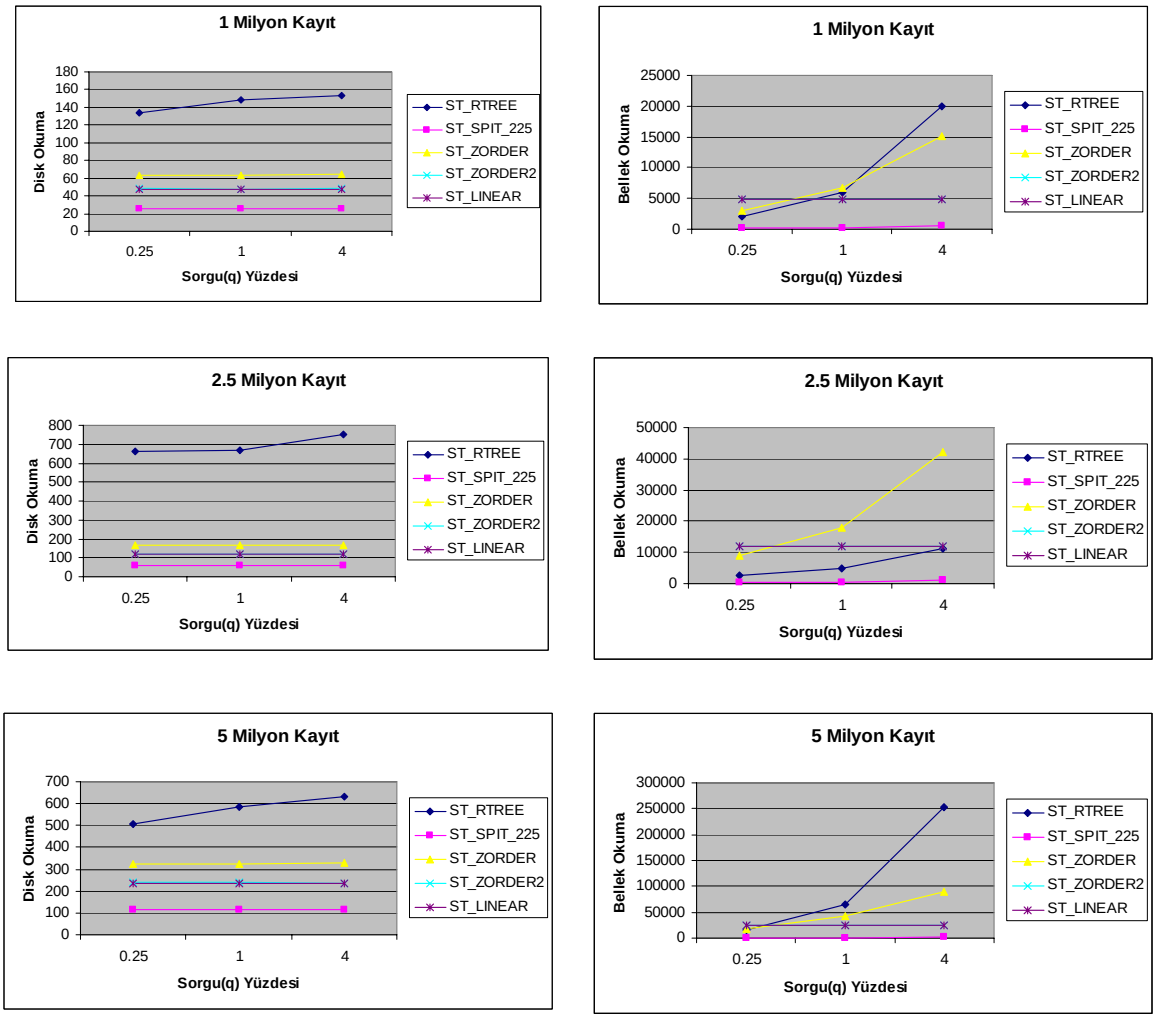


Şekil 7.6 Rastgele sorgu penceresi için farklı partitionlarda sorgu erişim süreleri

Mallet vd. yaptıkları son çalışmada AMD Athlon XP 3200+ işlemcili, 2.19Hz hızlı, 1.00 GB bellekli PC bilgisayar üzerinde deneylerini gerçekleştirmişlerdir (Botea, V.vd. 2008). Söz konusu çalışmada farklı partition sayıları için disk erişimleri normal dağılımda olmayan GSTD veri kümesi üzerinde en yüksek 1400 olmasına rağmen, bu tez çalışmasında sorgu pencerelerinin rastgele oluşturulmasına karşın 2.5 milyon kayıtlı veri kümesinde en yüksek 160 değerini almıştır. Bu da Mallett'in çalışmasında herhangi bir Oracle ayarlaması

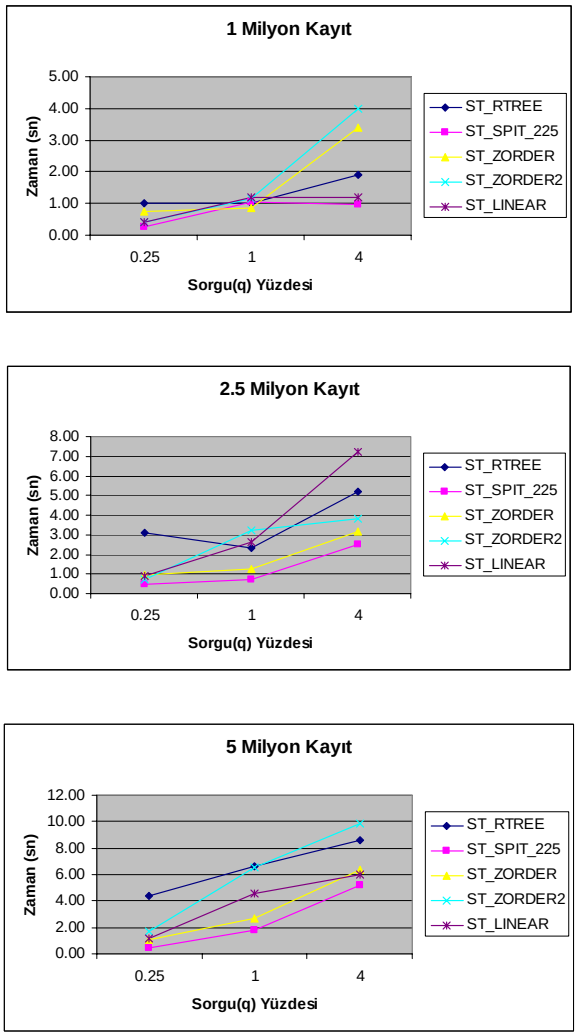
gerçekleştirmediğini göstermektedir.

Şekil 7.7'de farklı endeks yapılarının performansları verilmiştir. Bu kapsamda R-tree, 225 adet grid hücreli spit, z-order (uzay-zaman endekslenmiş) ve z-order2 (zaman-uzay) endekslenmiş ve hiç endeks yaratılmamış tablo sorgu erişimleri incelenmiştir. R-tree'nin çok fazla disk ve bellek ihtiyaçlarından dolayı bu yöntem en düşük performans ölçümlerini vermiştir. Z-order endeksleme, fonksiyonel endekslerle gerçekleştirildiği ve sorgu penceresinin alt sol ve üst sağ aralığındaki tüm hücrelerde arama yapmasını sağladığı için düşük performans alınmıştır. Z-order2 (zaman-uzay) endekslemenin hiç endeks yaratılmamış taramada arama ile aynı performansı verdiği görülmüştür. Tüm endeksleme yöntemleri içinde SPIT en iyi performans ölçümünü vermiştir. Sorgu oluşturan örnek PL/SQL kodu, Ek 4'te verilmiştir.



Şekil 7.7 Rastgele sorgu penceresi için disk ve bellek erişimleri

Rastgele sorgu penceresi için farklı endeks yapılarında sorgu işleme süreleri Şekil 7.8'de verilmiştir. 225 grid hücreye bölünmüş SPIT yapısı en iyi performansı göstermiştir.



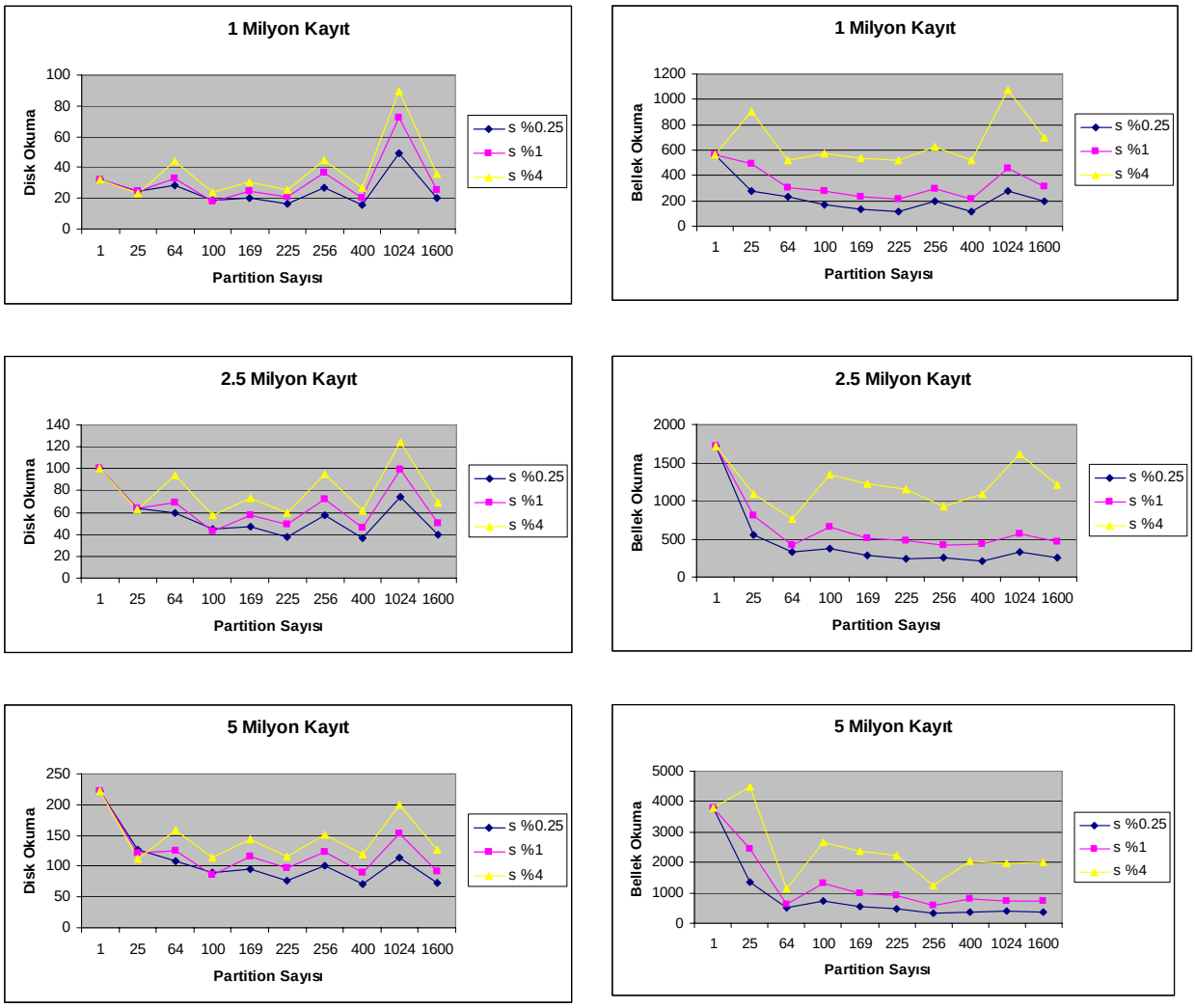
Şekil 7.8 Rastgele sorgu penceresi için sorgu işleme süreleri

Rastgele dikdörtgenlerin yaratılması ile farklı yoğunluktaki hareketli nesnelerin veri tabanından okunması, sorgu işleme zamanı grafiğinde düzensizliklere neden olmuştur. Gerçekleştirilen bu deneyin tez çalışmasına eklenmesinin nedeni, disk erişim süresinin yine de Mallett'in çalışmasından daha düşük olmasıdır. Ayrıca aynı sorgu penceresi içinde gerçekleştirilen ve bir sonraki bölümde verilen deneylerle en yüksek disk erişim, bellek erişim ve sorgu işleme süreleri değerlerine yakın sonuçların elde edilmesidir.

7.6 Aynı Sorgu Penceresi Oluşturularak Gerçekleştirilen Deneyler

Aynı sorgu pencereleri oluşturulurken, rastgele olarak seçilen sorgu penceresinin sağ üst köşesi tüm sorgularda aynı olarak kullanılmıştır. Sol alt köşesi sorgu alanının yüzdesine göre rastgele sorgu penceresindeki gibi oluşturulmuştur. Veri uzayının (space-s) %0.25' i için 1500x1500, %1'i için 3000x3000 ve %4'ü için de 6000x6000 büyüklüğünde sorgu pencereleri belirlenmiştir. Zaman aralığı tüm sorgularda, 2 birim değişmiştir.

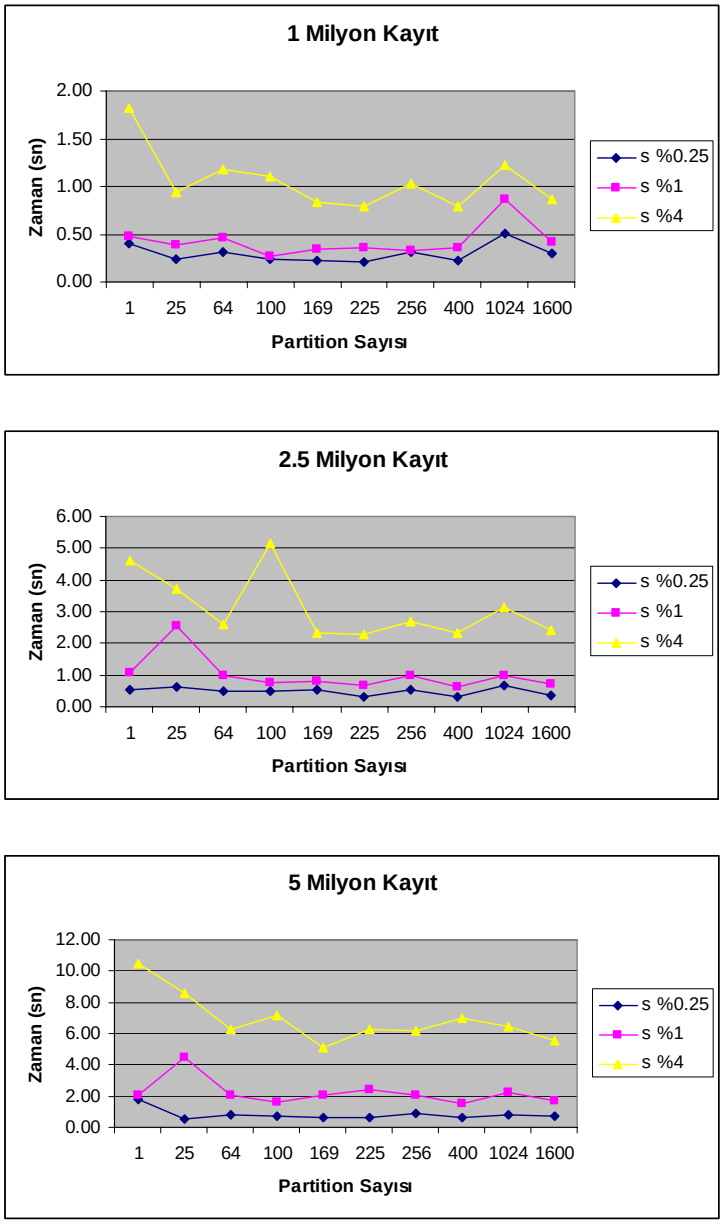
Şekil 7.9'da verilen grafiklerde farklı partition sayılarında disk ve bellek erişimleri gösterilmiştir. Sorgunun veri uzayı arttıkça disk ve bellek erişimleri de artmaktadır.



Şekil 7.9 Partitionlar üzerinde aynı sorgu penceresi için disk ve bellek erişimleri

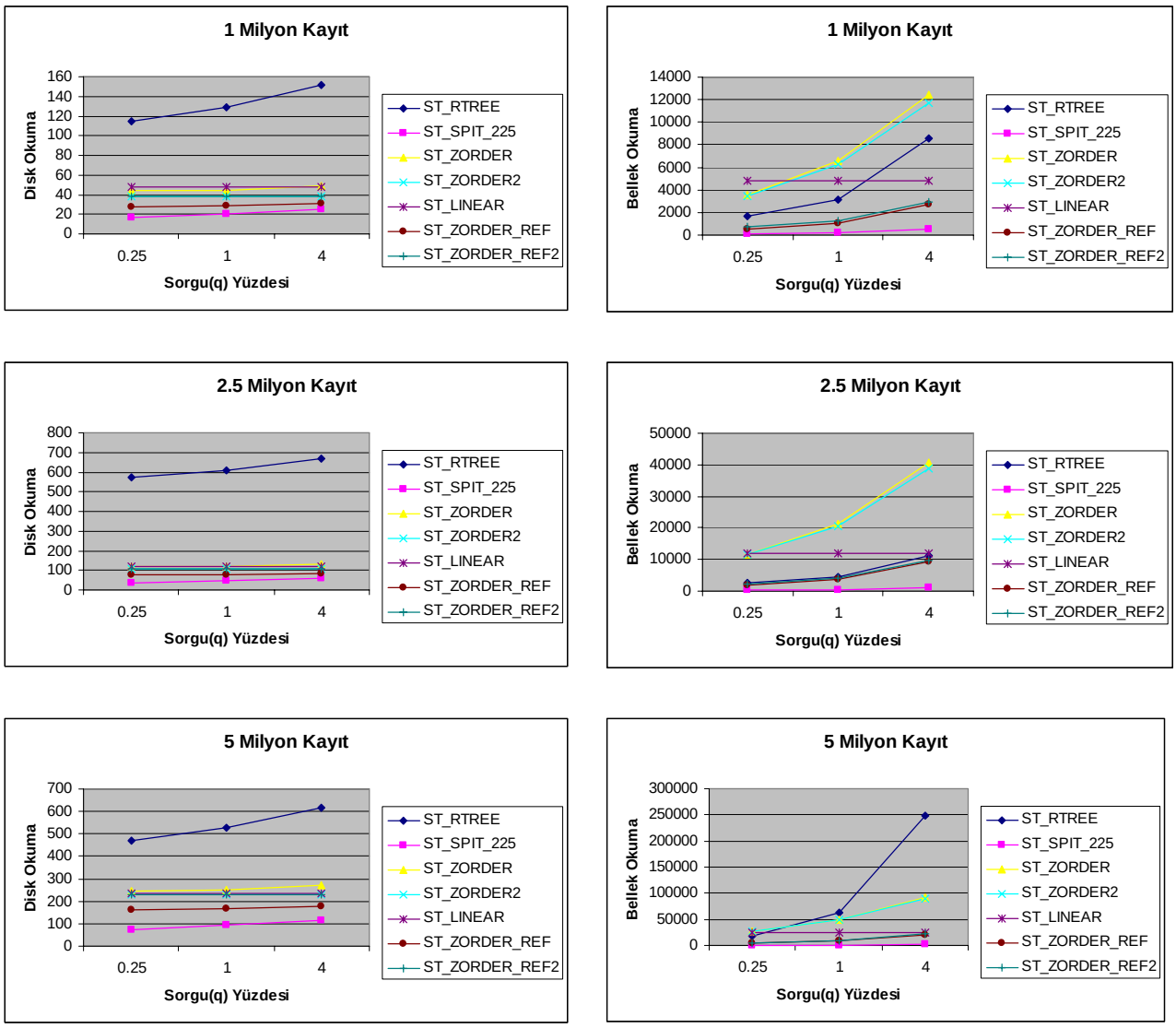
SPIT'in maliyet modeline göre en düşük disk erişiminin elde edildiği grid hücre sayısı, veri kümesinin büyüklüğüne, sorgu uzayının büyüklüğüne ve sorgu zaman uzayının büyüklüğüne göre değişmektedir. Fakat gerçekleştirilen deneylerde görülmüştür ki s %0.25 sorgularında en

iyi performans 400 grid hücreye bölündüğünde; s %1 sorgularında en iyi performans 100 grid hücreye bölündüğünde; s %0.25 sorgularında en iyi performans 25 grid hücreye bölündüğünde elde edilmiştir. Bu deneylerde veri kümesinin büyüklüğünün değişmesi bu sonuçlarda herhangi değişikliğe neden olmamaktadır. Bu değerler de aslında 1500, 3000 ve 6000 büyüklüğünde grid hücrelerden ve sorgu pencerelerinden elde edilmektedir. Örneğin 20x20 grid yaratmak için 30000x30000 büyüklüğündeki uzay 1500x1500 büyüklüğünde grid hücrelere bölünmüştür. Şekil 7.10'da farklı sayıdaki partition'lar için sorgu işleme süreleri verilmiştir.



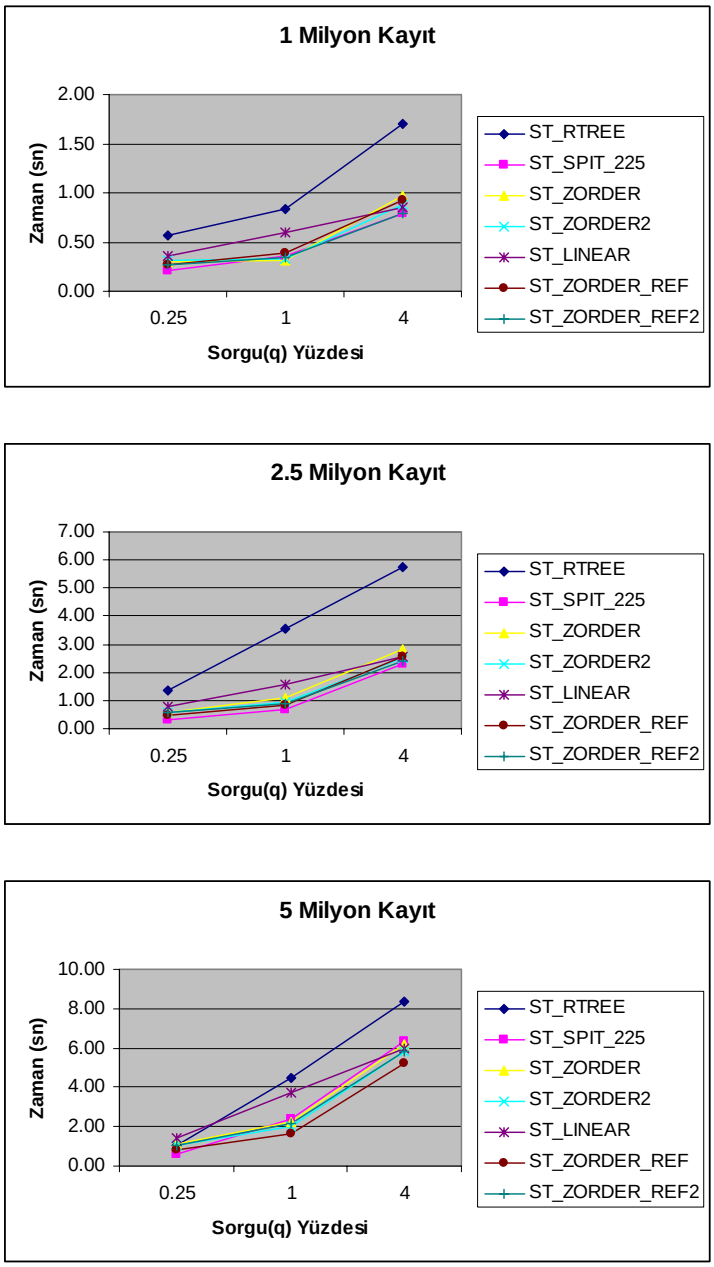
Şekil 7.10 Aynı sorgu penceresi için sorgu işleme süreleri

Şekil 7.11'de 7 farklı endeksleme yöntemi birbiri ile karşılaştırılmıştır. SPIT endeks tipinin en düşük disk ve bellek erişimi gerçekleştirdiği görülmektedir. Buna karşın R-tree çok fazla veriyi sorgu sırasında sabit diskten okumaktadır. Z-order ve Z-order2 endekslerinin fonksiyonel olmasından dolayı tüm deneylerde en yüksek belleği kullanmıştır. Veri kümesinde kayıt sayısı arttıkça R-tree de çok fazla bellek kullanmaktadır. Zorder-ref (refined) ile bu tez çalışmasında gerçekleştirilen *st_zorder_sorgusu*() algoritmasına göre gerçekleştirilen endekslerin de SPIT endeksinden sonra en etkin sorgulama yaptığı görülmüştür. Hiç endeksin olmadığı durumda ise doğrusal aramadan dolayı sabit sayıda disk ve bellek erişimleri gerçekleştirilmektedir.



Şekil 7.11 Aynı sorgu penceresi için farklı endeks performansları

Şekil 7.12'de farklı sorgu uzaylarında gerçekleştirilen sorguların sorgu erişim süreleri verilmiştir. R-tree'nin sorgu erişim süresi tüm veri kümelerinde en yüksektir. Buna karşın SPIT'in sorgu erişim süresi 1 milyon ve 2.5 milyon kayıtlı veri kümelerinde en düşük olanıdır. 5 milyon kayıtlı veri kümesinde ise zorder-ref'in erişim süresi en düşüktür. Bu da gerçekleştirilen deneylerde kullanılan endekslerin ölçeklenebilir ve güvenilir olduğunu göstermektedir.



Şekil 7.12 Aynı sorgu penceresi için farklı endekslerin sorgu işleme süreleri

7.7 Zamana Bağlı Deneyler

Sorgu penceresinin büyüklüğünü veri uzayının s%1' i olarak alıp, zaman noktalarının %5'i, %10'u ve %20'si alınarak deneyler gerçekleştirilmiştir. Deney ortalama sonuçları Çizelge 7.6'da verilmiştir.

Çizelge 7.6 Zaman aralığı deney sonuçları

	t %5								
	1 Milyon Kayıt			2.5 Milyon Kayıt			5 Milyon Kayıt		
	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma
ST_RTREE	1.27	2356	130	2.05	4598	609	4.58	8529	1547
ST_SPIT_225	0.48	219	21	0.96	472	49	2.04	912	96
ST_ZORDER	0.35	6569	44	1.06	21337	120	2.15	49665	250
ST_ZORDER2	0.38	6187	37	0.94	20379	104	1.93	47875	228
ST_LINEAR	0.50	4794	48	1.45	11970	120	2.29	23783	238
ST_ZORDER_REF	0.38	1063	29	0.87	3656	80	1.69	8225	168
ST_ZORDER_REF2	0.39	1235	38	0.93	4228	106	1.93	9536	231
	t %10								
	1 Milyon Kayıt			2.5 Milyon Kayıt			5 Milyon Kayıt		
	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma
ST_RTREE	0.83	3125	129	3.56	4598	609	4.51	61984	526
ST_SPIT_225	0.36	219	21	0.66	473	49	2.40	913	96
ST_ZORDER	0.31	6569	44	1.08	21337	120	2.21	49665	250
ST_ZORDER2	0.33	6267	38	0.93	20638	106	2.04	48442	231
ST_LINEAR	0.60	4794	48	1.59	11970	120	3.73	23783	237
ST_ZORDER_REF	0.39	1063	29	0.83	3656	80	1.67	8225	168
ST_ZORDER_REF2	0.35	1235	38	0.90	4228	106	2.15	9536	231
	t %20								
	1 Milyon Kayıt			2.5 Milyon Kayıt			5 Milyon Kayıt		
	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma
ST_RTREE	0.68	8359	126	2.06	9651	481	3.51	61984	526
ST_SPIT_225	0.38	219	21	0.73	472	49	1.53	912	96
ST_ZORDER	0.47	6571	44	1.06	21341	120	2.19	49671	250
ST_ZORDER2	0.43	6534	43	1.02	21463	119	2.48	49814	244
ST_LINEAR	0.47	4794	48	1.16	11970	120	2.45	23783	238
ST_ZORDER_REF	0.40	1071	31	0.84	3683	85	2.00	8269	175
ST_ZORDER_REF2	0.45	1502	43	1.08	5163	164	2.12	10908	244

1 milyon kayıtlı veri kümesi zaman aralığının çeşitli değerlerde alınmasına rağmen, birbirine çok yakın veya aynı değerler 100 deney ortalaması olarak elde edilmiştir. Bunun nedeni, daha önce de belirtildiği gibi t değerinin en fazla 20 zaman birimi olarak alınmasıdır. Hareketli nesne sayısı da bu deneylerde aynı sayıda elde edilmiştir.

7.8 Z-order Sıralı SPIT Yöntemi ile Gerçekleştirilen Deneyler

SPIT yöntemi z-order sıralı olarak gerçekleştirmek için tüm uzay 64, 256 ve 1024 grid hücreye bölünmüştür. Karşılaştırmayı sağlamak için, SPIT yöntemi ile de eş değer grid hücre sayısında bölümleme gerçekleştirilmiştir. Gerçekleştirilen deney ortalamaları Çizelge 7.7'de verilmştir.

Çizelge 7.7 Z-order sıralı SPIT deneyleri

	s %0.25								
	1 Milyon Kayıt			2.5 Milyon Kayıt			5 Milyon Kayıt		
	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma
ST_SPIT_64	0.32	234	28	0.50	328	60	0.85	517	109
ST_SPIT_ZORDER_64	**0.29**	**248**	**30**	**0.40**	**353**	**67**	0.90	**514**	**108**
ST_SPIT_256	0.32	194	27	0.55	253	57	0.93	343	100
ST_SPIT_ZORDER_256	**0.33**	**193**	**26**	**0.38**	**256**	**58**	1.37	**344**	**100**
ST_SPIT_1024	0.51	277	49	0.68	325	74	0.80	402	114
ST_SPIT_ZORDER_1024	**0.45**	**272**	**50**	**0.50**	**331**	**72**	1.08	**397**	**112**
	s %1								
	1 Milyon Kayıt			2.5 Milyon Kayıt			5 Milyon Kayıt		
	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma
ST_SPIT_64	0.46	308	33	1.00	412	69	2.09	613	124
ST_SPIT_ZORDER_64	**0.45**	**303**	**36**	**1.11**	**408**	**76**	2.02	**616**	**126**
ST_SPIT_256	0.33	299	36	1.01	412	72	2.08	576	123
ST_SPIT_ZORDER_256	**0.45**	**308**	**35**	**0.99**	**413**	**72**	2.05	**586**	**123**
ST_SPIT_1024	0.87	459	72	0.98	570	100	2.22	739	153
ST_SPIT_ZORDER_1024	**0.62**	**472**	**74**	**1.13**	**578**	**100**	2.18	**754**	**153**
	s %4								
	1 Milyon Kayıt			2.5 Milyon Kayıt			5 Milyon Kayıt		
	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma	Zaman	Bellek Okuma	Disk Okuma
ST_SPIT_64	1.18	523	44	2.61	758	94	6.24	1123	159
ST_SPIT_ZORDER_64	**0.97**	**574**	**49**	**2.66**	**725**	**100**	6.19	**1126**	**159**
ST_SPIT_256	1.03	623	45	2.66	929	95	6.18	1252	150
ST_SPIT_ZORDER_256	**0.96**	**671**	**44**	**2.57**	**931**	**91**	6.13	**1254**	**152**
ST_SPIT_1024	1.22	1075	89	3.14	1611	124	6.47	1981	199
ST_SPIT_ZORDER_1024	**1.13**	**1158**	**89**	**2.65**	**1452**	**127**	5.96	**1995**	**200**

Deney ortalama sonuçları değerlendirildiğinde, SPIT yönteminden daha düşük sonuçlar elde edileceği düşünülmüştür. Fakat bazı deneylerde daha kötü ve birbirine yakın deney sonuçları çıkmıştır. En önemli farklar çizelgede farklı renkte gösterilmiştir. Bu deney sonuçları, 1024 grid hücreli deneylerin z-order sıralı SPIT yönteminde fark yaratabileceğini düşünmemize neden olmuştur.

Ayrıca Oracle'a dizüstü bilgisayar açıldığında 1.0 GB belleği hemen tahsis etmesi nedeniyle, sabit diskten verileri almak yerine bellekten alması beklediğimiz sonuçları almamamıza neden olabileceğini düşündürmüştür. Bu nedenle bellek tekrar ilk deneyler sırasında kullandığımız miktara 256 MB'a düşürülüp, 1024 grid hücre için tekrar deneyler gerçekleştirilmiştir. Çizelge 7.8'de önceki çizelgedeki 1024 grid hücre deney sonuçları ve 256 MB'a düşürülmüş deneyler(parantez içinde 256 yazanlar ve koyu yazılanlar) verilmiştir.

Çizelge 7.8 256MB bellek ile Z-order sıralı SPIT yöntemi deneyleri

s %0.25

	1 Milyon Kayıt				2.5 Milyon Kayıt				5 Milyon Kayıt			
	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma
ST_SPIT_1024	105	0.51	277	49	385	0.68	325	74	912	0.80	402	114
ST_SPIT_1024(256)	**105**	**0.37**	**219**	**40**	**385**	**0.37**	**280**	**60**	**912**	**0.84**	**402**	**109**
ST_SPIT_ZORDER_1024	105	0.45	272	50	385	0.50	331	72	912	1.08	397	112
ST_SPIT_ZORDER_1024(256)	**105**	**0.51**	**272**	**50**	**385**	**0.72**	**331**	**72**	**912**	**0.59**	**404**	**96**

s %1

	1 Milyon Kayıt				2.5 Milyon Kayıt				5 Milyon Kayıt			
	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma
ST_SPIT_1024	393	0.87	459	72	1379	0.98	570	100	3282	2.22	739	153
ST_SPIT_1024(256)	**393**	**0.56**	**407**	**52**	**1379**	**0.85**	**558**	**78**	**3282**	**2.23**	**739**	**137**
ST_SPIT_ZORDER_1024	393	0.62	472	74	1379	1.13	578	100	3282	2.18	754	153
ST_SPIT_ZORDER_1024(256)	**393**	**0.61**	**472**	**74**	**1379**	**1.13**	**578**	**99**	**3282**	**1.51**	**881**	**135**

s %4

	1 Milyon Kayıt				2.5 Milyon Kayıt				5 Milyon Kayıt			
	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma
ST_SPIT_256	1646	1.03	623	45	5453	2.66	929	95	12850	6.18	1252	150
ST_SPIT_256(256)					sorgu penceresi haritada gezdiriliyor				**12439**		**1261**	**148**
ST_SPIT_ZORDER_256	1646	0.96	671	44	5453	2.57	931	91	12850	6.13	1254	152
ST_SPIT_ZORDER_256(256)					sorgu penceresi haritada gezdiriliyor				**12439**		**1279**	**146**
ST_SPIT_1024	1646	1.22	1075	89	5453	3.14	1611	124	12850	6.47	1981	199
ST_SPIT_1024(256)	**1646**	**0.94**	**1042**	**64**	**5453**	**2.43**	**1490**	**112**	**12850**	**6.65**	**1981**	**217**
ST_SPIT_ZORDER_1024	1637	1.13	1158	89	5453	2.65	1452	127	12850	5.96	1995	200
ST_SPIT_ZORDER_1024(256)	**1637**	**0.87**	**1158**	**89**	**5453**	**2.89**	**1456**	**120**	**12850**	**5.64**	**2515**	**263**

Sonuçlar değerlendirildiğinde, en önemli fark yaratanlar çizelgede farklı renkle gösterilmiştir. Sweep SPIT yönteminin düşük bellek ile daha iyi sonuçlar verdiği görülmüştür. Z-order sıralı SPIT yönteminde, sorgu yüzdesi arttıkça sonuçların daha kötü olduğu görülmüştür. Buna göre belleğin 256MB, grid hücre sayısının 1024, kayıt sayısının 5 milyon ve sorgu yüzdesinin %.0.25 olduğu durumda en iyi performans artışı elde edilmiştir.

Gerçekleştirilen bu deneylerden şu sonuç çıkarılmıştır: 1024 grid hücre sayısı 32x32 bir grid tanımlanarak elde edilmektedir. Grid hücrenin büyüklüğü 937.5x937.5'tir. Sorguların uzayın s%0.25'ini kapsadığı durumda 1500x1500 için performans artışı elde edilmiştir.

Bu deneylere ek olarak Çizelge 7.8'de s%4 için; 256 MB bellek ile 256 grid hücre için sorgu penceresi tüm uzay üzerinde hareket ettirilerek deney yapılmıştır. Bu deney sonucu fark yaratabilecek sonuçlar almamıza neden olabileceğini düşündürmüştür. Z-order sıralı SPIT yöntemi ile son olarak 256 MB bellek ile, 1024 grid hücre üzerinde 5 milyon kayıt için sorgu penceresi gezdirilmiştir. Çizelge 7.9'da bu deneylerden elde edilen ortalama değerler verilmiştir.

Çizelge 7.9 Z-order Sıralı SPIT yöntemi ile sorgu penceresinin uzay üzerinde gezdirilmesi

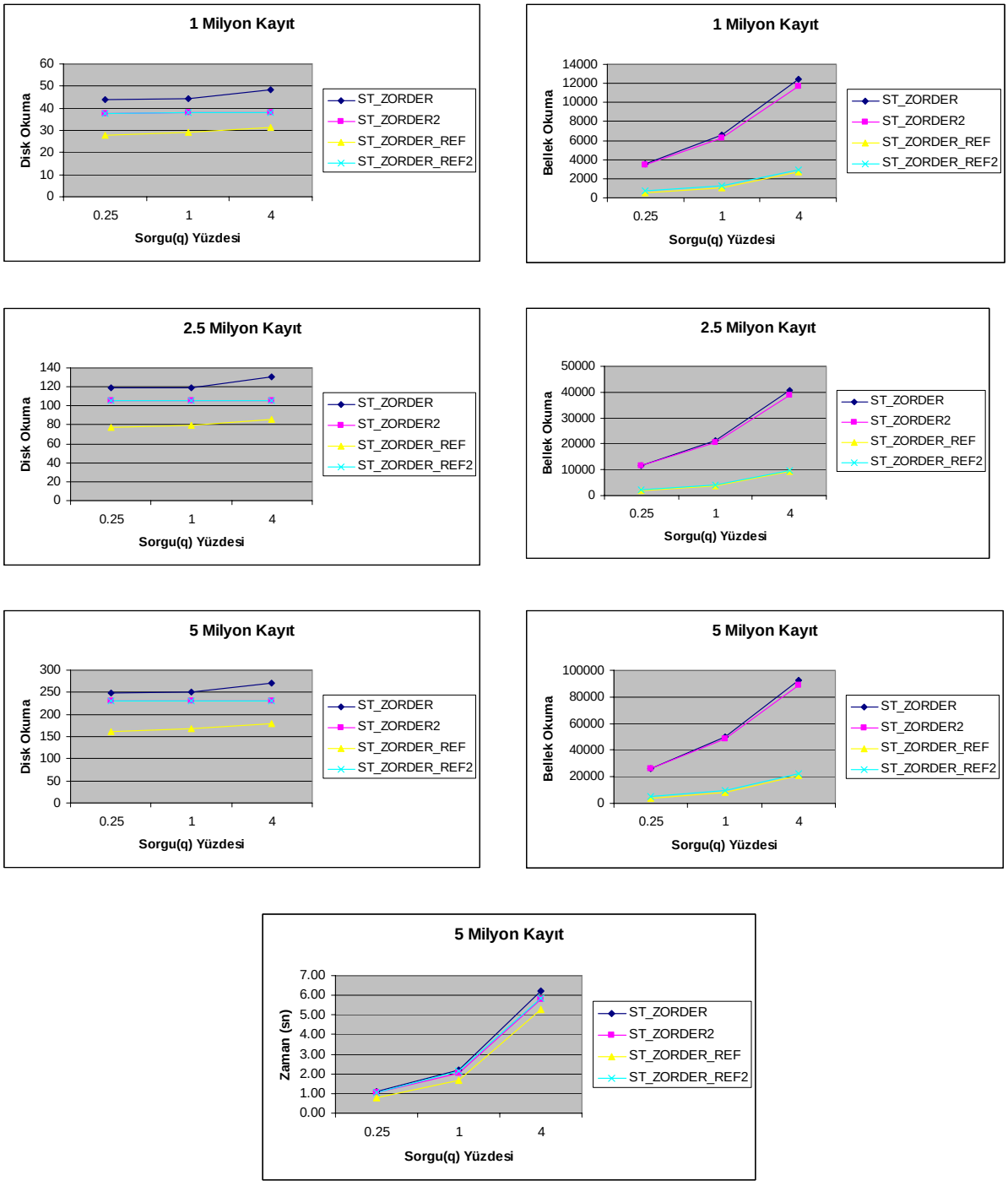
	ST_SPIT				**ST_SPIT_ZORDER**			
Sorgu Penceresi	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma	Kayıt Sayısı	Zaman	Bellek Okuma	Disk Okuma
x:=14700 ve y:=4600 noktası x:=x-100; y:=y+150;	744	0.5184	348	42	**744**	**0.616**	**346**	**31**
x:=14700 ve y:=3600 noktası x:=x-50; y:=y+250;	806	0.6908	364	52	**806**	**0.606**	**380**	**40**
x:=26700 ve y:=3600 noktası x:=x-50; y:=y+250;	515	0.8278	306	42	**515**	**0.4093**	**287**	**32**
x:=23000 ve y:=16500 noktası x:=x-50; y:=y+120;	1334	1.0548	566	46	**1334**	**0.802**	**562**	**42**
x:=18000 ve y:=16500 noktası x:=x+50; y:=y+120; Bir önceki satırın çaprazı	1164	0.7025	490	37	**1164**	**0.5783**	**486**	**33**

Z-order algoritması düşünüldüğünde tüm uzayın en sağ köşesinden uzayın sol tarafına doğru sorgu penceresinin gezdirilmesi birbirine en yakın sıralamayı verdiğinden disk okumalarının daha hızlı olacağı düşünülmüştür. Bu nedenle gerçekleştirilen ilk deneyde <x, y> değeri olarak <14700, 4600> noktası sorgu penceresinin en sağ üst köşesi olarak alınmış ve sorgu penceresinin sağ üst köşesi bir sonraki sorguda x'te 100 azaltılarak, y'de 1500 artırılarak oluşturulmuştur.

Z-order algoritmasının performansını düşürücü olarak tüm uzayın sol köşesinden tüm uzayın sağ köşesine doğru sorgu penceresi hareket ettirilmiştir. Bu deney sonucu Çizelge 7.9'da son satırda verilmiştir. Bu deneyde performansın iyi olduğu; Z-order sıralı SPIT yönteminde 33 disk okuma, sweep sıralı SPIT yönteminde 37 okuma olduğu görülmüştür.

7.9 Z-order Deneyleri

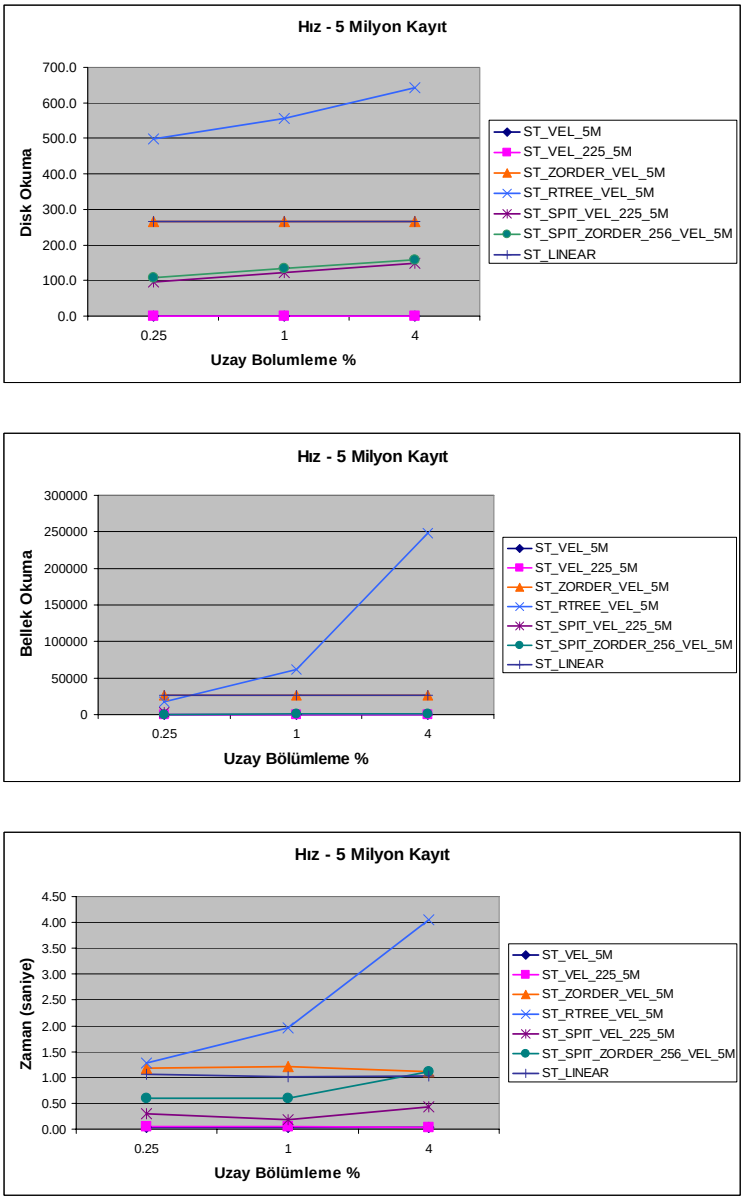
Zorder-ref (refined) ile bu tez çalışmasında gerçekleştirilen *st_zorder_sorgusu*() algoritmasına göre gerçekleştirilen endekslerin Mallett'in çalışmasında verilen Zorder sorgulamaları deneylerle karşılaştırılmıştır (Şekil 7.13). Veri kümesinin büyüklüğünden bağımsız olarak gerçekleştirdiğimiz sorgulamanın en iyi performansı verdiği görülmüştür.



Şekil 7.13 Z-order sorguları deney sonuçları

7.10 Hız Deneyleri

Tüm uzaya yayılmış serbest hareket eden nesneler içinde, yörüngesinin herhangi bir zamanında hız yapmış nesneleri sorgulama deneyleri bu bölümde incelenmiştir. Zorder(refined), R-tree, Spit, Z-order sıralı SPIT ve linear için yaratılmış tablolara hareket eden nesnelerin hızları için bir sütun eklenmiştir. Tüm yöntemler ölçeklenebilir olduğu için 5 milyon kayıt ile deneyler yapılmış ve farklı sorgu uzayları deneyleri Şekil 7.14'te verilmiştir.

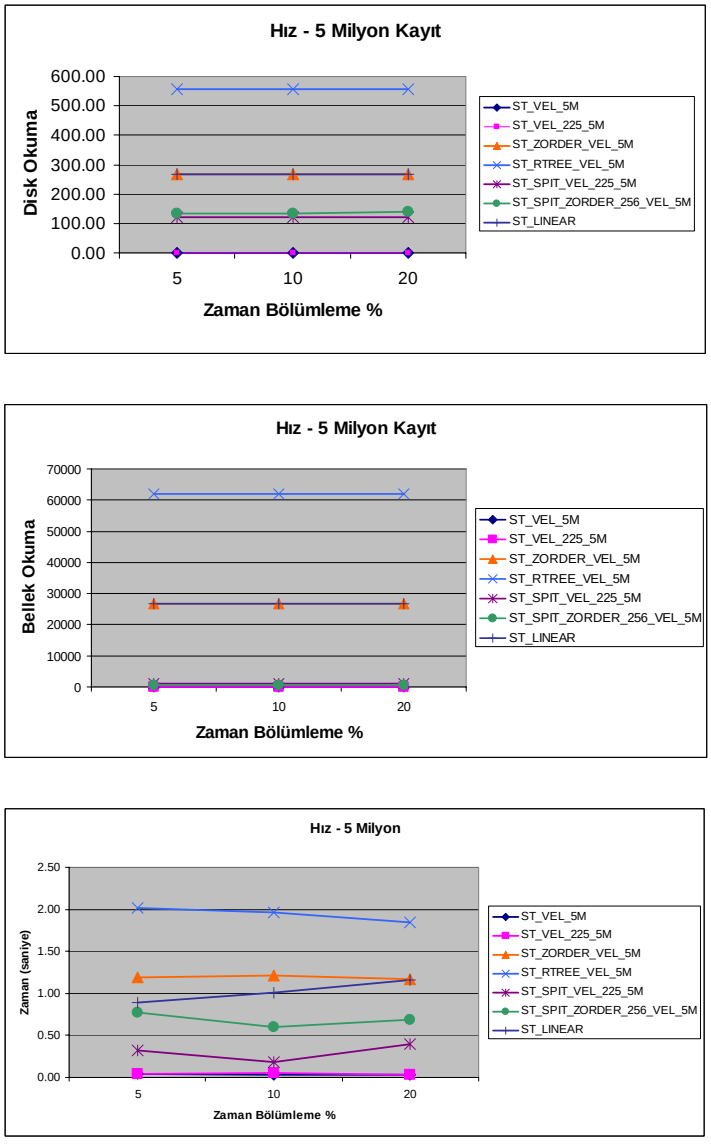


Şekil 7.14 Farklı sorgu uzaylarında hız için endeks performansları

Brinkhoff'un üreteci ile hareketli nesneler için 9 farklı hız oluşturulmaktadır. Bu nedenle 9 grid hücreli St_vel ile hıza göre partition oluşturulmuştur.

Diğer yeni bir yöntem de önceki bölümde tanımlanan hız için endeksleme yöntemidir. Buna göre 225 grid hücreli SPIT yönteminde 225nci partition'a hız yapmış hareket eden nesneler yerleştirilmiştir. Bu, Şekil 7.14'te St_vel_225 ile gösterilmiştir. Farklı zaman aralıkları alınarak oluşturulan deneylerin sonuçları da Şekil 7.15'te verilmiştir. Tüm hız deneylerinde en yüksek hız değeri 1197 ve bu nesnelerin sayısı da 13275'tir.

Hız deneylerinde en iyi performansı partition tabanlı St_vel ve St_vel_225 vermiştir. Fakat St_vel, 9 grid hücreli olduğu ve partitionlara hemen hemen eşit dağılımlı olmadığından St_vel_225'in daha iyi olduğu değerlendirilmektedir. Sonuç olarak, hareketli nesnenin konumu dışında ilgilenilen özelliklerin tüm uzayda saklanması yerine, ayrı bir partitionda tutulmasının performansı artıracağı değerlendirilmektedir.



Şekil 7.15 Farklı zaman aralıklarında hız için endeks performansları

7.11 MOORA Veri Kartuşu

Bu bölümde ismini MOORA (Moving Objects within ORAcle) olarak verip, tanımladığımız hareket eden nesneler sorgu sistemi için sorgu dili Oracle'da veri kartuşu olarak oluşturulmuştur. Tez çalışması, serbest hareket eden nesnelerin hareketlerini incelemektedir. Veri tabanında saklandığı gibi her hareket eden nesnenin, bir zaman anındaki durumu veri kartuşunu oluştururken de aynı şekilde ele alınmıştır.

MOORA veri kartuşunda hareket eden nesneler için uzaya, zamana ve hem uzaya hem zamana bağlı özel veri tipleri ve metotları tanımlanmıştır. Bu yetenekler tanımlanırken üçüncü bölümde tanımlanmış MADS kavramsal veri modelleme dili temel alınmıştır. Tanımladığımız uzaya bağlı metotlarda, MADS modelinin uzaya bağlı topolojik ilişkileri dikkate alınmıştır (Çizelge 7.10).

Çizelge 7.10 MADS uzaya bağlı topolojik ilişkileri

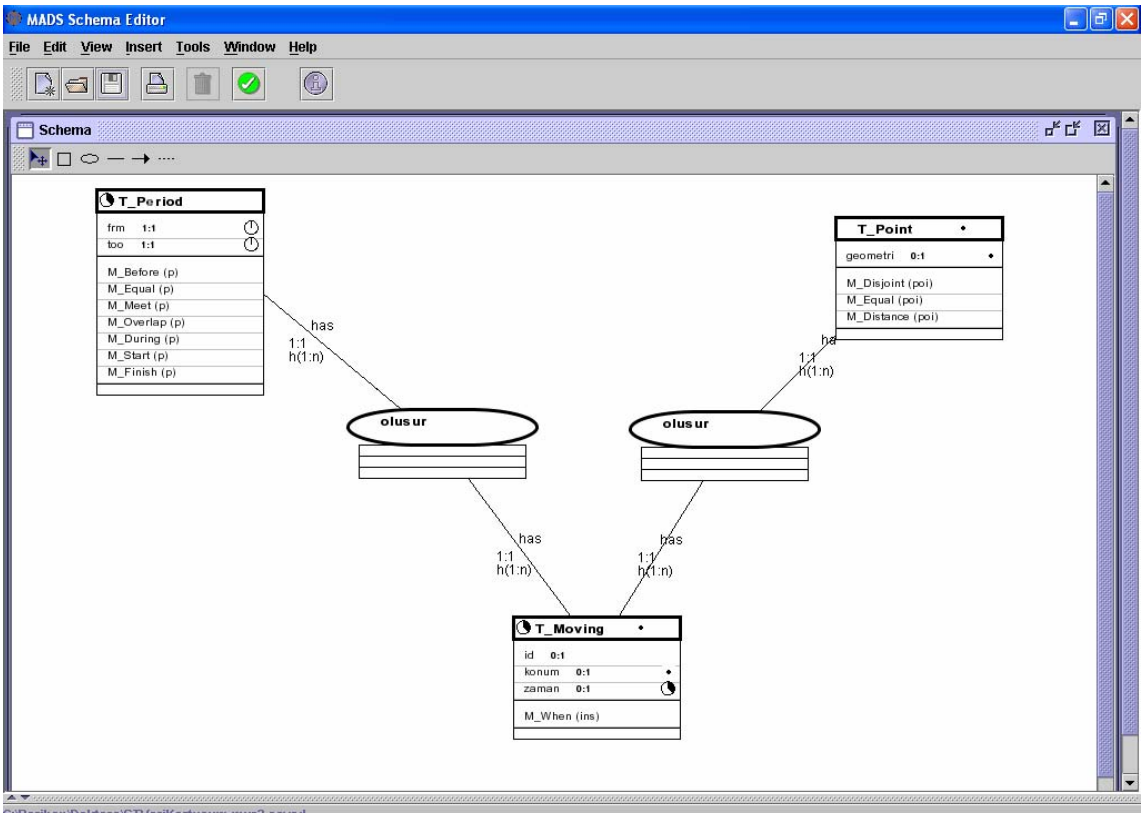
Topological kind	Icon	Topological kind	Icon
TopoDisjoint		TopoTouch	
TopoOverlap		TopoCross	
TopoWithin		TopoEqual	
TopoGeneric			

Aynı şekilde hareketli nesneler için tanımladığımız zamana bağlı metotlarda, MADS modelinin zamana bağlı senkronizasyon ilişkileri temel alınmıştır (Çizelge 7.11).

Çizelge 7.11 MADS zamana bağlı senkronizasyon ilişkileri

Synchronization kind	Icon	Synchronization kind	Icon
SyncPrecede		SyncMeet	
SyncWithin		SyncOverlap	
SyncStart		SyncFinish	
SyncEqual		SyncDisjoint	
SyncGeneric			

MOORA veri kartuşunda üç tane nesne tanımlanmıştır. Bunlar sırasıyla zamana bağlı veri tiplerinin ve metotlarının tanımlandığı T_Period (type period), uzaya bağlı veri tiplerinin ve metotlarının tanımlandığı T_Point ve hem uzaya hem zamana bağlı veri tiplerinin ve metotlarının tanımlandığı T_Moving'dir. MOORA veri kartuşunun MADS kavramsal veri gösterimi Şekil 7.16'da, PL/SQL tanımlamaları Ek 5'te verilmiştir.



Şekil 7.16 MOORA veri kartuşunun MADS gösterimi

T_Period ile bir zaman aralığı tanımlanmıştır. Bu nesnenin zaman aralığının başlangıç ve bitişini gösteren iki özelliği bulunmaktadır: frm ve too. Bu özellikler Oracle'da zaman anını gösteren date tipinde tanımlanmıştır. Bunlara ek olarak T_Period nesnesinin 7 metodu bulunmaktadır: M_Before (Method Before), M_Equal, M_Meet, M_Overlap, M_During, M_Start ve M_Finish. Bu metotlar t_period tipinde p parametresini alarak, boolean sonuç göndermektedir.

T_Point ile bir nokta geometrisi tanımlanmıştır. Bu nesnenin sadece <x, y> koordinatları sdo_geometry veri tipinde tanımlanmış geometri özelliği bulunmaktadır. T_Point nesnesinin metotları ise M_Disjoint, M_Equal ve M_Distance'tır. M_Disjoint ve M_Equal; T_Point tipinde poi parametresini alarak boolean sonuç göndermekte, M_Distance ise T_Point tipinde poi parametresini alarak float sonuç göndermektedir.

T_Moving nesnesi ile hareket eden nesne tanımlanmaktadır. Bu nesnenin özellikleri; number tipinde id, t_point tipinde konum ve t_period tipinde zaman'dır. Metodu ise zaman anı olarak ins parametresini alan ve t_point tipinde sonuç gönderen M_When'dir.

Açıklanan tüm nesnelerin özellikleri Ek 5'teki gibi tanımlanmaktadır. Metotları ise "type body" bölümünde tanımlanmaktadır.

7.11.1 MOORA veri kartuşu zamana bağlı metotları

T_Period nesnesinin "type body"inde tanımlanan metotlarından M_Before, PL/SQL ile aşağıdaki gibi tanımlanmaktadır. Eğer p parametresinin başlangıcından önce bitiyorsa, zaman olarak önce bittiğini göstermektedir.

```
MEMBER FUNCTION M_Before (p IN T_Period) return boolean
IS
BEGIN
  if too < p.frm then
    return true;
  else
    return false;
  end if;
END;
```

Daha sonra tanımlanan tüm metotlar, önce function sonra da operator olarak tanımlanarak SQL cümlesinde WHERE kelimesinden sonra kullanılması sağlanmaktadır. MOORA_Before operatörü aşağıda tanımlandığı gibi parametre olarak iki zaman aralığı almakta ve number tipinde veri göndermektedir.

```
CREATE OR REPLACE FUNCTION MOORA_Before_Func (object t_period, p T_Period)
return NUMBER
  AS
  BEGIN
    if object.M_Before (p) then
      return 1;
    else
      return 0;
    end if;
  END;

CREATE OPERATOR MOORA_Before BINDING(t_period, t_period)
  RETURN NUMBER USING MOORA_Before_Func;
```

7.11.2 MOORA veri kartuşu uzaya bağlı metotları

T_Point nesnesinin "type body"inde tanımlanan metotlarından M_Disjoint, PL/SQL ile aşağıdaki gibi tanımlanmaktadır. Eğer poi parametresinin x ve y koordinatlarından farklı ise, uzay olarak birbirinden ayrık olduğunu göstermektedir.

```
  MEMBER FUNCTION M_Disjoint (poi IN T_Point) return boolean
  IS
  BEGIN
    if geometri.sdo_point.x<>poi.geometri.sdo_point.x and
       geometri.sdo_point.y<>poi.geometri.sdo_point.y then
      return true;
    else
      return false;
    end if;
  END;
```

Uzayda ayrık olup olmadıklarını kontrol eden function ve operator tanımları aşağıdaki gibi yapılmıştır. MOORA_Disjoint_Space operatörü görüldüğü gibi parametre olarak iki uzay noktasını almakta ve number tipinde veri göndermektedir.

```
CREATE OR REPLACE FUNCTION MOORA_Disjoint_Func(object t_point, poi T_Point)
return NUMBER
  AS
  BEGIN
    if object.M_Disjoint(poi) then
      return 1;
    else
      return 0;
    end if;
  END;

CREATE OPERATOR MOORA_Disjoint_Space BINDING(t_point, t_point)
  RETURN NUMBER USING MOORA_Disjoint_Func;
```

7.11.3 MOORA veri kartuşu hem uzaya hem zamana bağlı metotları

T_Moving nesnesinin "type body"inde tanımlanan metotlarından M_When, PL/SQL ile aşağıdaki gibi tanımlanmaktadır. Eğer ins parametresi ile bildirilen zaman anı nesnenin zaman aralığının içinde ise nesnenin konumu gönderilmektedir. Hareket eden nesneler için en önemli metottur.

```
CREATE OR REPLACE TYPE BODY T_MOVING
IS

  MEMBER FUNCTION M_when (ins IN date) return t_point
  AS
    i number;
  BEGIN
    if ins> zaman.frm and ins < zaman.too then
       return konum;
    else
       return(null);
    end if;

  END;

END;
```

Zaman aralığının içinde olanların konumlarını geri gönderen function ve operator tanımları yapılmıştır. MOORA_M_When operatörü görüldüğü gibi parametre olarak hareketli bir nesneyi ve zaman anını almakta ve t_point tipinde uzay verisi göndermektedir.

```
CREATE OR REPLACE FUNCTION MOORA_M_When_Func(object t_moving, ins date)
return t_point
  AS
  BEGIN
    return(object.M_When(ins));
  END;

CREATE OPERATOR MOORA_M_When BINDING(t_moving, date)
  RETURN t_point USING MOORA_M_When_Func;
```

Tüm function ve operatörler tanımlandıktan sonra veri ekleyebileceğimiz bir tablo yaratılmaktadır. Tabloyu tanımlamadan önce, ORACLE'ın sdo_geometry'sinde bir noktayı kolayca tanımlamamızı sağlayacak nokta fonksiyonu aşağıdaki gibi tanımlanmıştır.

```
CREATE OR REPLACE FUNCTION nokta(x NUMBER, y NUMBER, srid NUMBER DEFAULT
8307)
  RETURN SDO_GEOMETRY DETERMINISTIC
  AS
  BEGIN
    RETURN SDO_GEOMETRY (2001, srid, SDO_POINT_TYPE (x,y,NULL), NULL, NULL);
  END;
```

7.11.4 MOORA veri kartuşu sorgulama işlemleri

Tablo tanımı ve yaptığımız tanımlamaları denemek için 4 adet veri ekleme için PL/SQL kodu aşağıda verilmiştir.

```
create table MOORA
(MovObj t_moving);

insert into MOORA values(
    t_moving(1, t_point(nokta(5,4)),
    t_period(to_date('01-01-09:12:00:00','dd-mm-yy:hh24:mi:ss'),
                to_date('02-02-09:12:00:10','dd-mm-yy:hh24:mi:ss') )));

insert into MOORA values(
    t_moving(2, t_point(nokta(5,4)),
       t_period(to_date('01-01-09:12:00:10','dd-mm-yy:hh24:mi:ss'),
                to_date('02-02-09:12:00:30','dd-mm-yy:hh24:mi:ss') ) ));

insert into MOORA values(
    t_moving(3, t_point(nokta(8,8)),
       t_period(to_date('01-01-09:12:00:20','dd-mm-yy:hh24:mi:ss'),
                to_date('02-02-09:12:01:00','dd-mm-yy:hh24:mi:ss') ) ));

insert into MOORA values(
    t_moving(4, t_point(nokta(9,8)),
       t_period(to_date('01-01-09:12:00:20','dd-mm-yy:hh24:mi:ss'),
                to_date('03-02-09:12:01:00','dd-mm-yy:hh24:mi:ss') ) ));
```

MOORA_BEFORE zaman operatörünün SELECT SQL cümlesindeki kullanımı ve sorgu sonuçları aşağıda verilmiştir. Buna göre tablodaki her satırın movObj.zaman ile verilen özelliğinin 03 Şubat 2009 tarihinden önce gelenler listelenmiştir.

```
select * from moora b where MOORA_BEFORE(b.movObj.zaman,
t_period(to_date('03-02-09:13:01:00','dd-mm-yy:hh24:mi:ss'),
              to_date('02-03-09:12:00:10','dd-mm-yy:hh24:mi:ss') ))=1;

MOVOBJ(ID, KONUM(GEOMETRI(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z),
SDO_ELEM_INFO, SDO_ORDINATES)), Z
--------------------------------------------------------------------------
-------------------------
T_MOVING(1, T_POINT(SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(5, 4, NULL),
NULL, NULL)), T_PERIOD('01-
T_MOVING(2, T_POINT(SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(5, 4, NULL),
NULL, NULL)), T_PERIOD('01-
T_MOVING(3, T_POINT(SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(8, 8, NULL),
NULL, NULL)), T_PERIOD('01-
T_MOVING(4, T_POINT(SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(9, 8, NULL),
NULL, NULL)), T_PERIOD('01-
```

MOORA_DISJOINT_SPACE uzay operatörünün SELECT SQL cümlesindeki kullanımı ve sorgu sonuçları aşağıda verilmiştir. Buna göre tablodaki her satırın movObj.konum ile verilen <5,4> noktasından ayrık olanlar (farklı olanlar) listelenmiştir.

```
select * from moora b where MOORA_DISJOINT_SPACE(b.movObj.konum,
t_point(nokta(5,4)))=1;

MOVOBJ(ID, KONUM(GEOMETRI(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z),
SDO_ELEM_INFO, SDO_ORDINATES)), Z
--------------------------------------------------------------------------
-------------------------
T_MOVING(3, T_POINT(SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(8, 8, NULL),
NULL, NULL)), T_PERIOD('01-
T_MOVING(4, T_POINT(SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(9, 8, NULL),
NULL, NULL)), T_PERIOD('01-
```

Diğer operatörlerden farklı olduğu için uzay operatörlerinden iki nokta arasındaki uzaklığı hesaplayan MOORA_DISTANCE'ın aşağıda kullanımı ve sorgu sonuçları verilmiştir.

```
select MOORA_DISTANCE(b.movObj.konum, t_point(nokta(5,4))) from moora b;


MOORA_DISTANCE(B.MOVOBJ.KONUM,T_POINT(NOKTA(5,4)))
--------------------------------------------------
                                                 0
                                                 0
                                                 5
                                       5.65685425
```

Hem uzaya hem zamana bağlı MOORA_EQUAL_SPACE operatörünün SELECT SQL cümlesindeki kullanımı ve sorgu sonuçları aşağıda verilmiştir. Bu operatör, predicate olmadığından diğer predicate operatörlerden biri olan MOORA_EQUAL_SPACE ile kullanılarak sonuçlar elde edilmiştir. Buna göre 01 Şubat 2009 tarihi zaman aralığının içinde olan nesnelerden, konumu <5, 4> olanlar listelenmiştir.

```
select * from moora b where
  MOORA_Equal_Space(MOORA_M_When(b.movObj,(to_date('01-02-09:12:00:00','dd-
mm-yy:hh24:mi:ss'))),t_point(nokta(5,4)))=1;

MOVOBJ(ID,   KONUM(GEOMETRI(SDO_GTYPE,   SDO_SRID,   SDO_POINT(X,   Y,   Z),
SDO_ELEM_INFO
--------------------------------------------------------------------------
-----
T_MOVING(1,  T_POINT(SDO_GEOMETRY(2001,  8307,  SDO_POINT_TYPE(5,  4,  NULL),
NULL, N
ULL)), T_PERIOD('01-JAN-09', '02-FEB-09'))

T_MOVING(2,  T_POINT(SDO_GEOMETRY(2001,  8307,  SDO_POINT_TYPE(5,  4,  NULL),
NULL, N
ULL)), T_PERIOD('01-JAN-09', '02-FEB-09'))
```

7.12 Değerlendirme

Bu bölümde maliyet modeli normal dağılıma göre oluşturulmuş SPIT yapısı ve endeksleri oluşturularak farklı veri ve partition büyüklüklerinde deneyler yapılmıştır. SPIT maliyet modelinde veri kümesinin büyüklüğü önemliyken, gerçekleştirilen deneylerle, sorgu penceresi ile örtüşen partition sayısının en etkin sorgulamayı gerçekleştirdiği ispatlanmıştır.

Oracle veri tabanında yapılan ayarlar ile eski model olmasına rağmen dizüstü bilgisayarın, Mallett'in deneylerinden disk erişimleri olarak iyi sonuçlar verdiği saptanmıştır. Sorgu işleme zamanının daha büyük elde edilmesinin nedeni olarak da sunucu makinelerinde bulunan hızlı sabit disklerin dizüstü bilgisayarlarda olmaması değerlendirilmektedir.

SPIT algoritması z-order'a göre sıralanmış ve farklı sorgu algoritması oluşturulmuştur. Bu deneylerde sorgu penceresinin rastgele konumlara yerleştirilmesi yerine yakın noktalarda gezdirilmesi ile iyileştirmeler elde edilmiştir. Z-order sorgu algoritması ile oluşturulan zorder-ref ile de performans artışı oluşturulmuştur.

Yapılan hız deneyleri ile hareketli nesnelerin konum ve zamanının önemli olmadığı veya tüm uzaya yayılmış ve çok az sayıda hareketli nesnede bulunan ilgilenilen özelliklerin boş bir partitionda saklanmasıyla iyileştirmeler gerçekleştirilmiştir.

Son olarak da endeksleme yöntemi oluşturulmuş MOORA sorgu sistemi için sorgulama dilinin özellikleri Oracle veri kartuşu ile tanımlanmıştır. Tez çalışması kapsamında incelenen MADS kavramsal modeli ile veri kartuşunun nesneleri ve ilişkileri gösterilmiştir.

8. SONUÇ VE ÖNERİLER

Teknolojinin ilerlemesi, elektronik cihazların ucuzlaması ile günümüzde, pek çok elektronik cihaz yaygın olarak kullanılmaktadır. GPS cihazlarının; kablosuz cihazlara eklenmesi, maliyetinin düşük olması ve bulunulan konum hakkında doğru ölçümler yapmasından dolayı, çok fazla veri bilgi sistemleriyle toplanmaktadır. Bu veriler, uzay ve zamana bağlı olarak çok boyutlu oldukları için çok boyutlu veriler veya hem uzaya hem zamana bağlı veriler olarak isimlendirilmişlerdir.

Bilgi sistemleriyle bu kapsamda, çoğunlukla arabalar, gemiler, uçaklar, askeri birlikler vb. gibi nesneler izlenmekte ve görüntülenmektedir. Bu nesneler de genel olarak, hareket eden nesneler olarak isimlendirilmektedir. İzlenen hareket eden nesnelerden elde edilen bu tip verilerin gizliliği olması nedeniyle, ticari firmalar bu bilgileri araştırmacıların kullanımına sunamamaktadırlar. Bundan dolayı, hareket eden nesne verileri üreten çeşitli üreteç çalışmaları gerçekleştirilmiştir. Araştırmacıların, hareket eden nesneleri izleme ve görüntülemeyi sağlayarak bir uygulamayı gerçekleştirmeleri için bir çok teknolojiyi bütünleşik olarak kullanmaları gerekmektedir. Konunun güncel olması nedeniyle literatürdeki çalışmalarda, hareket eden nesneler için tek problem ele alınarak incelenmiştir.

Bu tez çalışmasında hareket eden nesneler için, bütünleşik ve özgün bir sorgu sistemi oluşturma hedeflenmiştir. Bunun için literatürde yapılan en son çalışmalar kapsamlıca incelenerek, özetlenmiştir. Hareketli nesneler için etkin erişim yöntemleri konusunda gerçekleştirilen erişim yöntemleri; mevcut, gelecek ve geçmiş zaman başlıkları altında incelenmiştir. Çalışmada kullanmak üzere, Türkiye'nin yollarına ait sayısal harita araştırılmıştır. Fakat gizlilik ve ticari nedenlerle, bu tür herkese açık bir harita bulunamamıştır. İstanbul Büyükşehir Belediyesinin Şehir Rehberi kapsamında ticari kuruluşlara yaptırdığı çalışmalar öğrenildikten sonra, belediyeden araştırma için Eminönü-İstanbul sayısal haritası temin edilmiştir. Daha sonra hareket eden nesneler için oluşturulmuş veri üreteçleri denenerek, veriler oluşturulmuştur. Bu üreteçler içinde sayısal haritayı girdi olarak alıp, trafik verisi üreten ve bir çok araştırmada kullanılan Brinkhoff'un üreteci bu çalışmada kullanılmıştır.

Hareket eden nesneler için oluşturulan veri yapıları, hem uzaya hem de zamana bağlıdır. Çok az sayıda araştırmada veri tabanı sistemi kullanılarak hareketli nesneler konusu araştırıldığı için, bu çalışmada uzaya bağlı ilişkisel veri tabanı yönetim sistemi kullanılmıştır. Böylece çalışmada zamana bağlı veri tipleri ve işlemleri gerçekleştirmeye yönelerek, uzaya bağlı

Oracle Spatial 10g veri tabanı yönetim sistemi kullanılarak hareket eden nesneler için sorgu sistemi geliştirilmiştir.

Hareket eden nesneler için araştırmalarda Visual Basic, Java ve C++ programlama dilleri kullanılmıştır. Bu çalışmanın başlangıcında coğrafi sistemler için gerçekleştirilmiş uygulama yeteneklerini daha hızlı görmek maksadıyla, Visual Basic ile Esri firmasının geliştirdiği MapObjects yazılım parçası (component) kullanılarak, Oracle'ın sitesinden temin edilen USA haritası üzerinde çeşitli deneme çalışmaları gerçekleştirilmiştir. Daha sonra Oracle veri tabanı sistemi üzerindeki sayısal harita üzerinde hareketli nesneler oluşturmak için Java programlama dili kullanılmıştır.

Ayrıca, günümüzde kullanılan kavramsal veri modelleri incelenerek, nesneye yönelik modellemeyi sağlayan ve uzun yıllar boyunca kullanılan MADS modeli bu çalışmada kullanılmıştır. Kavramsal düzeyde tanımlanan sorgu dilleri çalışmaları da kapsamlı olarak çalışmada incelenmiştir. Bu tür bir sorgu dilini Oracle veri tabanı yönetimi sisteminin, veri kartuşları ile tanımladığı tespit edilmiştir.

Bu tez çalışması ile incelenen aralık sorgularıdır. Bir hem uzaya hem zamana bağlı aralık (range) sorgusu Q, $Q = \langle R, T \rangle$ şeklinde gösterilmektedir. Burada R, uzaya bağlı bir alanı ve T de zaman aralığını ifade etmektedir. Q, R içinde bulunan $\langle x, y \rangle$ noktalarını ve T ile kesişen $\langle t_s, t_e \rangle$ zamanları içeren kayıtların farklı nesne id'lerini geri göndermektedir. Bu çalışmada incelenen problem, hem uzaya hem zamana bağlı geçmişe ait nokta verilerinin endekslenmesi ve en kısa zamanda aralık sorgu sonucunun elde edilmesidir.

MOORA sorgu sistemi için; SPIT yöntemi, R-tree + zamana bağlı B-tree ve Z-değerleri + B-tree endeks yapıları oluşturulmuştur. En etkin sorgulama SPIT yöntemi ile elde edilmiştir. Bu yaklaşımlarda hareket eden nesnelerin hız bilgisi kullanılmamıştır. Bu endeks yapıları incelendiğinde, z-değerleri + B-tree algoritmasının istenen sorgu penceresi alanı dışında daha geniş alanları taradığı görülmüştür. Bu nedenle, istenen sorgu penceresi alanı içinde hareketli nesnelerin araştırılması için, her grid hücrenin z-order değeri ayrı bir tabloda saklanmıştır. Daha sonra, sorgu penceresi ile çakışan hücrelerin z-order değerleri veri tabanından alan yeni bir algoritma geliştirilmiş ve performans artışı elde edilmiştir.

Bu tez çalışması kapsamında gerçekleştirilen bilime diğer bir katkı da, SPIT bölümleme algoritmasını z-order sıralamasına uygun olarak gerçekleştirilmesidir.

Bir trafik simülasyonunda, gezingesinin bir bölümünde hızlı hareket etmiş araçların sayıları az olduğu için bu nesnelerin ayrı bir bölümde (partition) endekslenmesi bu çalışmada teklif edilmiştir. Bunun için MOORA sorgu sisteminde oluşturulan endeks yapıları için tablolara hareketli nesnelerin hızları için ayrı bir sütun eklenmiştir.

Bunlara ek olarak MOORA sorgu dili yeteneklerini belirlemek ve gerçekleştirmek için oluşturulan örnek veri kartuşu özellikleri ve işlemleri MADS kavramsal modeli ile birleştirilerek incelenmiştir. Daha sonra MOORA sorgulama dilinin özellikleri veri kartuşu ile tanımlanmıştır.

Genel olarak değerlendirildiğinde, MOORA sorgu sistemi deneyleri aşağıdaki aşamaları izleyerek gerçekleştirilmiştir.

- Normal dağılım için oluşturulmuş maliyet modeline göre, SPIT algoritmasını gerçekleştirme ve farklı veri büyüklüklerinde ve partition sayılarında değerlendirme,
- SPIT endeksleme yöntemlerini diğer endeksleme yöntemleri ile karşılaştırma,
- İstanbul haritası üzerinde hareket eden nesneler için SPIT algoritması gerçeklendiğinde, en uygun partition sayısını deneysel olarak belirleme,
- Hızlı hareket etmiş nesneler için en etkin endeks yapısını oluşturma,
- MOORA veri kartuşunu oluşturarak hareket eden nesneler için veri tipleri yaratma ve sorgu dili oluşturma.

Deney sonuçlarına göre SPIT maliyet modelinde veri kümesinin büyüklüğü önemliyken, sorgu penceresi ile örtüşen partition sayısının en etkin sorgulamayı gerçekleştirdiği ispatlanmıştır.

SPIT algoritması z-order'a göre sıralanmış ve farklı sorgu algoritması oluşturulmuştur. Bu deneylerde sorgu penceresinin rastgele konumlara yerleştirilmesi yerine yakın noktalarda gezdirilmesi ile iyileştirmeler elde edilmiştir.

Yapılan hız deneyleri ile hareketli nesnelerin konum ve zamanının önemli olmadığı veya tüm uzaya yayılmış ve çok az sayıda hareketli nesnede bulunan ilgilenilen özelliklerin boş bir partitionda saklanmasıyla iyileştirmeler gerçekleştirilmiştir.

Gerçekleştirilen kapsamlı inceleme sonucunda, tez çalışması kapsamında oluşturulan bu özgün sorgu sisteminin literatüre beş katkısının olduğu söylenebilir:

1. **Veri modeli:** Hareket eden nesnelerin bir ağ üzerinde, geçmişe yönelik verilerini saklamak için veri modeli ve veri tabanı için de kavramsal veri modeli MADS ile oluşturulmuştur.

2. **Trafik verisi:** Eminönü-İstanbul sayısal haritası kullanılarak, hareket eden nesneler için veriler Brinkhoff'un üreteci ile oluşturulmuştur.

3. **Erişim yöntemleri:** Hareket eden nesne bilgilerine hızlı ulaşmak için SPIT yöntemi, z-order sıralı SPIT, R-tree + zamana bağlı B-tree ve Z-değerleri + B-tree endeks yapıları oluşturularak performans ölçümleri elde edilmiştir. Zorder için yeni bir sorgulama algoritması yaratılmıştır. Oluşturulan bu sistemde yörüngesinde herhangi bir zamanda hız yapmış nesnelerin sayısının az olmasından dolayı, hızlı nesneler için yeni bölümlenme algoritması tasarlanarak gerçekleştirilip, performans iyileştirmesi elde edilmiştir.

4. **Sorgu dili tasarımı:** Hareket eden nesneler sorgu dili, soyut bir model üzerinde tasarlanmıştır.

5. **Sorgu dili gerçekleştirimi:** Veri kartuşu kullanılarak, tasarlanan sorgu dili özellikleri, işlemleri ve algoritmaları gerçekleştirilmiştir.

İleriye dönük olarak, hareket eden nesnelerin gezingelerinin veri modellerini oluşturma, kavramsal modelini oluşturma, gezingeleri endeksleme, gezingelerden elde edilen bilgileri daha az yer kaplayacak şekilde sabit bellekte saklama için bu bilgilerin sıkıştırılması, benzer gezingeleri bulma konularında daha pek çok araştırma gerçekleştirilebilir. Ayrıca veri kartuşu için yeni endeksleme yöntemi, optimizasyon işlemleri ve maliyet modeli oluşturulabilir.

Son olarak veri üreteçleri ile üretilmiş veriler dışında, gerçek-hayat bilgi sistemleri ile toplanmış verilerin araştırmacılara sunulmasından sonra, pek çok araştırma alanlarının oluşacağı değerlendirilmektedir.

KAYNAKLAR

Abdelguerfi, M., Givaudan, J., Shaw, K.ve Ladner, R. (2002), "The 2-3 TR-tree, A Trajectory-Oriented Index Structure for Fully Evolving Valid-time Spatio-temporal Datasets", Geographic Information Systems, November 2002, ACM, 29-34.

Abraham, T.ve Roddick, J. F. (1999), "Survey of Spatio-Temporal Databases", GeoInformatica 3(1): 61-99.

Acker, R., Pieringer, R.ve Bayer, R. (2005), Towards Truly Extensible Database Systems, Database and Expert Systems Applications: 596-605.

Almeida, V. T. D. (2006), "Moving Objects in Networks Databases", EDBT2006.

Almeida, V. T. D.ve Güting, R. H. (2005), "Indexing the Trajectories of Moving Objects in Networks", GeoInformatica 9(1): 33-60.

Almeida, V. T. d., Guting, R. H.ve Behr, T. (2006), "Querying Moving Objects in SECONDO", Mobile Data Management, IEEE.

Almeida, V. T. d., Guting, R. H.ve Duntgen, C. (2007), "Multiple Entry Indexing and Double Indexing", Proceedings of the 11th International Database Engineering and Applications Symposium, IEEE Computer Society.

Alvares, L. O., Bogorny, V., Kuijpers, B., Macedo, J. A. F. d., Moelans, B.ve Vaisman, A. (2007), "A model for enriching trajectories with semantic geographical information", Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, Seattle, Washington, ACM.

An, K., Lee, J.ve Kim, K. (2005), "Design of Query Language for Tracking Moving Objects", Geoscience and Remote Sensing Symposium, IEEE, 3506-3509.

Artale, A., Parent, C.ve Spaccapietra, S. (2007), "Evolving objects in temporal information systems", Ann Math Artif Intell.

Athitsos, V., Hadjieleftheriou, M., Kollios, G.ve Sclaroff, S. (2007), "Query-sensitive embeddings", ACM Trans. Database Syst. 32(2): 8.

Baars, M. (2004), "Moving Objects in a geo-DBMS", Delft University.

Behr, T., Almeida, V. T. d.ve Güting, R. H. (2006), "Representation of periodic moving objects in databases", Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems, Arlington, Virginia, USA, ACM.

Bertino, E.ve Ooi, B. C. "The Indispensability of Dispensable Indexes", IEEE Transactions on Knowledge and Data Engineering 11(1).

Botea, V., Mallett, D., Nascimento, M.ve Sander, J. (2008), "PIST: An Efficient and Practical Indexing Technique for Historical Spatio-Temporal Point Data", GeoInformatica 12(2): 143-168.

Brakatsoulas, S., Pfoser, D.ve Theodoridis, Y. (2002), "Revisiting R-Tree Construction Principles", ADBIS, Springer-Verlag, 149-162.

Brakatsoulas, S., Pfoser, D.ve Tryfona, N. (2004), "Modeling, Storing and Mining Moving Object Databases", Database Engineering and Applications Symposium(IDEAS'04).

Brinkhoff, T. (2000), "Generating Network-Based Moving Objects", Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM'00), IEEE Computer Society.

Brinkhoff, T. (2002), "A Framework for Generating Network-Based Moving Objects", GeoInformatica 6(2): 153-180.

Brinkhoff, T. (2003), "Generating Traffic Data", Bulletin of the Technical Commitee on Data Engineering, IEEE Computer Society, 19-25.

Cai, M.ve Revesz, P. (2000), "Parametric R-Tree: An Index Structure for Moving Objects", International Conference on Management of Data, COMAD, December 2000.

Chakka, V. P., Everspaugh, A.ve Patel, J. M. (2003), "Indexing Large Trajectory Data SetsWith SETI", First Biennial Conference on Innovative Data Systems Research, January 2003, VLDB-ACM SIGMOD.

Chen, S., Jensen, C. S.ve Lin, D. (2008), "A benchmark for evaluating moving object indexes", Proc. VLDB Endow. 1(2): 1574-1585.

Cui, B., Lin, D.ve Tan, K.-L. (2005), "IMPACT: A twin-index framework for efficient moving object query processing", Data Knowl. Eng. 59: 63-85.

Ding, H., Trajcevski, G.ve Scheuermann, P. (2008), "Efficient Maintenance of Continuous Queries for Trajectories", GeoInformatica 12(3): 255-288.

Dumas, M., Fauvet, M.-C.ve Scholl, P.-C. (2004), "TEMPOS: A Platform for Developing Temporal Applications on Top of Object DBMS", IEEE Trans. on Knowl. and Data Eng., 354-374.

Düntgen, C., Behr, T.ve Güting, R. H. (2008), "BerlinMOD: a benchmark for moving object databases", Hagen, Germany, FernUniversitat.

Fengli, Z., Jiebang, Y., Zhiguang, Q.ve Mingtian, Z. (2003), "Performance Assessment of Retrieving Information of Moving Objects with Spatio-Temporal Index", Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003, IEEE, 863-867.

Forlizzi, L., Güting, R. H., Nardelli, E.ve Schneider, M. (2000), "A Data Model and Data Structures for Moving Objects Databases", Proc.ACM SIGMOD International Conference on Management of Data, Dallas.

Frentzos, E. (2003), "Indexing Objects Moving on Fixed Networks", SSTD 2003, Springer-Verlag

Frentzos, E., Gratsias, K., Pelekis, N.ve Theodoridis, Y. (2005), "Nearest Neighbor Search on Moving Object Trajectories", SSTD 2005, Springer-Verlag.

Frentzos, E., Gratsias, K.ve Theodoridis, Y. (2007a), "Index-based Most Similar Trajectory Search", ICDE 2007.

Frentzos, E., Gratsias, K.ve Theodoridis, Y. (2007b), "Towards the Next Generation of Location-Based Services", W2GIS 2007, Springer.

Frentzos, E., Gratsias, K.ve Theodoridis, Y. (2008a), "On the Effect of Location Uncertainty in Spatial Querying", IEEE Transactions on Knowledge and Data Engineering.

Frentzos, E., Pelekis, N.ve Theodoridis, Y. (2008b), "Cost Models and Efficient Algorithms on Existentially Uncertain Spatial Data", Panhellenic Conference on Informatics, IEEE.

Frentzos, E., Pelekis, N.ve Theodoridis, Y. (2008c), "Cost Models and Efficient Query Processing over Existentially Uncertain Spatial Data", Hellas, University of Piraeus.

Frentzos, E. K. (2008), "Trajectory Data Management in Moving Object Databases", University of Piraeus.

Freytag, J.-C., Flasza, M.ve Stillger, M. (2000), "Implementing Geospatial Operations in an Object-Relational Database System", Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM'00), IEEE Computer Society.

Giannotti, F., Mazzoni, A., Puntoni, S.ve Renso, C. (2005), "Synthetic generation of cellular network positioning data", Geographic Information Systems'05, New York, NY, USA, ACM.

Griffiths, T., Fernandes, A. A. A., Paton, N. W.ve Barr, R. (2004), "The Tripod spatio-historical data model", Data Knowl. Eng. 49(1): 23-65.

Güting, H., Almeida, T. d.ve Ding, Z. (2006), "Modeling and querying moving objects in networks", The VLDB Journal 15(2): 165-190.

Guting, R. H., Almeida, V., Ansorge, D., Behr, T., Ding, Z., Hose, T., Hoffmann, F., Spiekermann, M.ve Telle, U. (2005), "SECONDO: An Extensible DBMS Platform for Research Prototyping and Teaching", Proceedings of the 21st International Conference on Data Engineering, IEEE Computer Society.

Guting, R. H., Almeida, V. T. d., Ansorge, D., Behr, T., Spiekermann, M.ve Düntgen, C., (2007a), "SECONDO Programmer's Guide" Retrieved December 2008, 2008.

Guting, R. H., Ansorge, D., Behr, T.ve Spiekermann, M. (2007b), "SECONDO User Manual", Hagen, Germany, Praktische Informatik IV, Fernuniversitat Hagen.

Güting, R. H., Böhlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M.ve Vazirgiannis, M. (2000), "A foundation for representing and querying moving objects", ACM Trans. Database Syst. 25(1): 1-42.

Guting, R. H.ve Schneider, M. (2005), Moving Objects Databases, Boston, Morgan

Kaufmann.

Guttman, A. (1984), "R-trees: a dynamic index structure for spatial searching", Proceedings of the 1984 ACM SIGMOD international conference on Management of data, Boston, Massachusetts, ACM.

Hadjieleftheriou, M., Hoel, E.ve Tsotras, V. J. (2005), "SaIL: A Spatial Index Library for Efficient Application Integration", GeoInformatica 9(4): 367-389.

Iwerks, G. S., Samet, H.ve Smith, K. P. (2006), "Maintenance of K-nn and spatial join queries on continuously moving points", ACM Trans. Database Syst. 31(2): 485-536.

Jensen, C., Tiešytė, D.ve Tradišauskas, N. (2006), The COST Benchmark—Comparison and Evaluation of Spatio-temporal Indexes, Database Systems for Advanced Applications: 125-140.

Jensen, C. S., Lin, D.ve Ooi, B. C. (2004), "Query and Update Efficient B^+-Tree Based Indexing of Moving Objects", 30th VLDB Conference, Toronto, Canada.

Jensen, C. S., Tieste, D.ve Tradisajskas, N. (2006), "Robust B^+-tree based indexing of moving objects", MDM, 12-20.

Jin, P.ve Sun, P. (2008), "OSTM: a Spatiotemporal Extension to Oracle", Fourth International Conference on Networked Computing and Advanced Information Management, IEEE.

Jin, P., Wan, S.ve Yue, L. (2008), "Conceptual Modeling for Moving Objects Database Applications", Proceedings of the The Ninth International Conference on Mobile Data Management (mdm 2008) IEEE Computer Society.

Karimi, H. A.ve A.H. (2004), Telegeoinformatics Location-Based Computing & Services, CRC Press.

Kilimci, P.ve Kalıpsız, O. (2007a), "Geometric Modelling and Visualization of Moving Objects on a Digital Map", 2nd International Conference on Geometric Modelling and Imaging, 4-6 July 2007, Zurich, 39-43.

Kilimci, P.ve Kalıpsız, O. (2007b), "Moving Objects Databases in Space Applications", 3rd International Conference on Recent Advances in Space Technologies, 14-16 June 2007, Istanbul, Turkey, 106-108.

Kilimci, P.ve Kalıpsız, O. (2008a), "Hareket Eden Nesnelerin MADS Belirtim Dili ile Modellenmesi ve Gerçeklenmesi", 2. Ulusal Yazılım Mimarisi Konferansı'08, 11-12 Eylül 2008, İzmir, Ege Üniversitesi, 125-134.

Kilimci, P.ve Kalıpsız, O. (2008b), "Konuma ve Zamana Bağlı Veri Temelli Yazılım Mimarileri için Kavramsal Modelleme ve Gerçekleme", Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, 9-10 Ekim 2008, İstanbul, 41-46.

Kothuri, R. K. V., Godfrind, A.ve Beinat, E. (2007), Pro Oracle Spatial for Oracle Database 11g, Springer-Verlag.

Kothuri, R. K. V., Hanckel, R.ve Yalamanchi, A. (2008), "Using Oracle Extensibility Framework for Supporting Temporal and Spatio-Temporal Applications", 15th International Symposium on Temporal Representation and Reasoning, IEEE.

Kothuri, R. K. V.ve Ravada, S. (2002), "Spatio-Temporal Indexing in Oracle: Issues and Challenges", IEEE TCDE Bulletin, 56-60.

Kothuri, R. K. V., Ravada, S.ve Abugov, D. (2002), "Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data", Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, ACM.

Kothuri, R. K. V., Ravada, S.ve An, N. (2004), "Incorporating Updates in Domain Indexes: Experiences with Oracle Spatial R-trees", IEEE Computer Society(Proceedings of the 20th International Conference on Data Engineering(ICDE'04)).

Kothuri, R. K. V., Ravada, S.ve Xu, W. (2003), "Spatial Processing using Oracle Table Functions", 19th International Conference on Data Engineering, IEEE.

Koubarakis, M.ve T.S.e.a. (2003), Spatio-temporal databases : the CHOROCHRONOS approach, Berlin; New York, Springer.

Krajzewicz, D., Hertkorn, G., Rössel, C.ve Wagner, P. (2002), "SUMO (Simulation of Urban MObility): An open-source traffic simulation", 4th Middle East Symposium on Simulation and Modeling (MESM2002), SCS European Publishing House, 183-187.

Krassimir Markov, K. I., Ilia Mitov, Stefan Karastanev (2008), "Advance of the Access Methods", Information Technologies and Knowledge.

Kriegel, H.-P., Pfeifle, M., Pötke, M.ve Seidl, T. (2003), "The Paradigm of Relational Indexing: A Survey", 10. GI-Fachtagung Datenbanksysteme für Business Technologie und Web, Leipzig.

Kriegel, H.-P., Pötke, M.ve Seidl, T. (2000), "Managing Intervals Efficiently in Object-Relational Databases", VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, Cairo, Egypt, Morgan Kaufmann, 407-418.

Kriegel, H.-P., Pötke, M.ve Seidl, T. (2001), "Interval Sequences: An Object-Relational Approach to Manage Spatial Data", 7th International Symposium on Spatial and Temporal Databases, Springer-Verlag.

Kwon, D., Lee, S.ve Lee, S. (2002), "Indexing the Current Positions of Moving Objects Using the Lazzy Update R-tree", Mobile Data Management, January 2002, 113-120.

Ladner, R.ve K.S. (2002), Mining Spatio-Temporal Information Systems, Kluwer Academic Publishers.

Le, Y. (2004), "A Feature-Based Temporal Representation and Its Implementation with Object-Relational Schema for Base Geographic Data in Object-Based Form", UCGIS Assembly, Adelphi.

Lee, M., Hsu, W., Jensen, C., Cui, B.ve Teo, K. (2003), "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach", International Conference on Very Large Data Bases, September 2003.

Lema, J. A. C., Forlizzi, L., Güting, R. H., Nardell, E.ve Schneider, M. (2001), "Algorithms for Moving Objects Databases", The Computer Journal 46(6): 680-712.

Li, W.ve Mozes, A. (2004), "Computing Frequent Itemsets Inside Oracle 10g", 30th VLDB Conference.

Lin, D., Jensen, C. S., Ooi, B. C.ve Saltenis, S. (2005), "Efficient indexing of the historical, present, and future positions of moving objects", Proceedings of the 6th international conference on Mobile data management, Ayia Napa, Cyprus, ACM.

Mallett, D., Nascimento, M. A., Botea, V.ve Sander, J. (2005), "RDBMS Support for Efficient Indexing of Historical Spatio-Temporal Point Data", Aalborg University.

Mallett, D. J. (2004), "Relational Database Support for Spatio-Temporal Data", University of Alberta.

Manolopoulos, Y., Nanopoulos, A.ve Papadopoulos, A. N. (2000), "R-trees Have Grown Everywhere", ACM Computing Surveys.

Mokbel, M. F. (2004), "Continuous Query Processing in Spatio-Temporal Databases", EDBT 2004 Workshops, LNCS 3268, 100-111.

Mokbel, M. F., Aref, W. G.ve Kamel, I. (2002), "Performance of multi-dimensional space-filling curves", Proceedings of the 10th ACM international symposium on Advances in geographic information systems, McLean, Virginia, USA, ACM.

Mokbel, M. F., Ghanem, T. M.ve Aref, W. G. (2003), "Spatio-Temporal Access Methods", IEEE Data Eng. Bull. 26(2): 40-49.

Mokbel, M. F., Xiong, X., Hammad, M. A.ve Aref, W. G. (2005), "Continuous Query Processing of Spatio-Temporal Data Streams in PLACE", GeoInformatica 9(4): 343-365.

Myllymaki, J.ve Kaufman, J. (2003), DynaMark: A Benchmark for Dynamic Spatial Indexing, Mobile Data Management: 92-105.

Nascimento, M. A., Pfoser, D.ve Theodoridis, Y. (2003), "Synthetic and Real Spatiotemporal Datasets", IEEE TCDE Bulletin 26(1): 26-32.

Nascimento, M. A.ve Silva, J. R. O. (1998), "Towards historical R-trees", Symposium on Applied Computing, February 1998, 235-240.

Nascimento, M. A., Silva, J. R. O.ve Theodoridis, Y. (1999), "Evaluation of Access Structures for Discretely Moving Points", International Workshop on Spatio-Temporal Database Management, September 1999, 171-188.

Ni, J. (2007), "Indexing Spatio-Temporal Trajectories with Efficient Polynomial

Approximations", IEEE Trans. on Knowl. and Data Eng. 19(5): 663-678.

Obalı, M. (2007), Oracle 10g, Pusula.

Oracle, (1999), "Developing Stored Procedures in Java".

Oracle (2000), "Oracle Spatial Java Library User's Guide".

Oracle (2001a), "Oracle9i: Program with PL/SQL Additional Practices".

Oracle (2001b), "Oracle9i: Program with PL/SQL Student Guide Volume 1".

Oracle (2001c), "Oracle9i: Program with PL/SQL Student Guide Volume 2".

Oracle (2001d), "Oracle 9i: Spatial Student Guide Volume 1".

Oracle (2001e), "Oracle 9i: Spatial Student Guide Volume 2".

Oracle (2001f), "Oracle Spatial Relational Model Guide and Reference".

Oracle (2003), "Oracle Spatial Quadtree Indexing".

Oracle (2004a), "Oracle Database 10g Developing Spatial Applications Using Oracle Spatial and MapViewer".

Oracle (2004b), "Oracle Database 10g Empowering Applications with Spatial Analysis and Mining".

Oracle (2004c), Oracle Database 10g: PL/SQL Fundamentals.

Oracle (2005a), "GeoRaster".

Oracle (2005b), "Oracle Spatial Topology and Network Data Models".

Oracle (2006), Oracle Spatial User' Guide and Reference 10g Release 2 (10.2). Oracle.

Oracle (2007), "Java Developer's Guide".

Oracle (2008a), "Data Cartridge Developer's Guide 11g Release 1 (11.1). Oracle Database Documentation Library ".

Oracle (2008b), "Oracle Database SQL Language Reference 11g Release 1".

Oracle (2008c), "Oracle SQL Paralel Execution".

Parent, C., Spaccapietra, S.ve Zimányi, E. (2006), Conceptual Modeling for Traditional and Spatio-Temporal Applications, Berlin, Heidelberg, Springer.

Patel, J. M., Chen, Y.ve Chakka, V. P. (2004), "STRIPES: An Efficient Index for Predicted

Trajectories ", SIGMOD 2004, Paris, France, ACM.

Pelanis, M., Saltenis, S.ve Jensen, C. S. (2006), "Indexing the past, present, and anticipated future positions of moving objects", ACM Trans. Database Syst. 31(1): 255-298.

Pelekis, N., Frentzos, E., Giatrakos, N.ve Theodoridis, Y. (2008), "HERMES: aggregative LBS via a trajectory DB engine", Proceedings of the 2008 ACM SIGMOD international conference on Management of data, Vancouver, Canada, ACM.

Pelekis, N.ve Theodoridis, Y. (2006), "Boosting location-based services with a moving object database engine", Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access, Chicago, Illinois, USA, ACM.

Pelekis, N.ve Theodoridis, Y. (2007), "An Oracle Data Cartridge for Moving Objects", University of Piraeus.

Pelekis, N., Theodoridis, Y., Vosinakis, S.ve Panayiotopoulos, T. (2006), Hermes - A Framework for Location-Based Data Management, Advances in Database Technology - EDBT 2006: 1130-1134.

Pelekis, N.ve Theodoulidis, B. (2002), "TAU TLL Data Cartridge", Manchester, UK.

Pelekis, N., Theodoulidis, B., Kopanakis, I.ve Theodoridis, Y. (2004), "Literature review of spatio-temporal database models", Knowl. Eng. Rev. 19(3): 235-274.

Pfoser, D. (2000), "Issues in the Management of Moving Point Objects", Aalborg, Aalborg University.

Pfoser, D. (2002), "Indexing the Trajectories of Moving Objects", Bulletin of the Technical Committee on Data Engineering 25(2).

Pfoser, D.ve Jensen, C. S. (2003), "Indexing of Network Constrained Moving Objects", GIS'03 New Orleans, ACM.

Pfoser, D., Jensen, C. S.ve Theodoridis, Y. (2000), "Novel Approaches in Query Processing for Moving Object Trajectories", Proceedings of the 26th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 395-406.

Pfoser, D.ve Theodoridis, Y. (2003), "Generating semantics-based trajectories of moving objects", Computers, Environment and Urban Systems 27(3): 243-263.

Porkaew, K., Lazaridis, I.ve Mehrotra, S. (2001), "Querying Mobile Objects in Spatio-Temporal Databases", International Symposium on Advances in Spatial and Temporal Databases, July 2001, Redondo Beach, CA, 59-78.

Prabhakar, S., Xia, Y., Kalashnikov, D. V., Aref, W. G.ve Hambrusch, S. E. (2002), "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects", IEEE Trans. Comput. 51(10): 1124-1140.

Procopiuc, C. M., Agarwal, P. K.ve Har-Peled, S. (2002), "STAR-Tree: An Efficient Self-

Adjusting Index for Moving Objects", Workshop on Alg. Eng. and Experimentation, ALENEX, January 2002, 178-193.

Ralston, B. A. (2002), Developing GIS Solutions with MapObjects and Visual Basic, Onward Press.

Ravada, S.ve Sharma, J. (1999), Oracle8i Spatial: Experiences with Extensible Databases, Advances in Spatial Databases: 355-359.

Ravi Kanth V Kothuri, S. R., Ning An (2004), "Incorporating Updates in Domain Indexes: Experiences with Oracle Spatial R-trees", IEEE Computer Society(Proceedings of the 20th International Conference on Data Engineering(ICDE'04)).

Saglio, J. M.ve Moreira, J. (2001), "Oporto: A Realistic Scenario Generator for Moving Objects", GeoInformatica 5(1): 71-93.

Saltenis, S.ve Jensen, C. S. (2002), "Indexing of Moving Objects for Location-Based Services", International Conference on Data Engineering, ICDE, IEEE.

Saltenis, S., Jensen, C. S., Leutenegger, S. T.ve Lopez, M. A. (2000), "Indexing the Positions of Continuously Moving Objects", International Conference on Management of Data, May 2000, SIGMOD, 331-342.

Scheugenpflug, S.ve Scilcher, M. (2004), "Object-Relational Features for Modeling and Analysis of Spatio-temporal data", ISPRS 2004.

Shekhar, S.ve S.C. (2003), Spatial Databases A Tour, USA, Prentice Hall.

Sistla, A. P., Wolfson, O., Chamberlain, S.ve Dao, S. (1997), "Modeling and Querying Moving Objects", Proceedings of the Thirteenth International Conference on Data Engineering, IEEE Computer Society.

Song, Z.ve Roussopoulos, N. (2001), "Hashing Moving Objects", Mobile Data Management, January 2001, 161-172.

Song, Z.ve Roussopoulos, N. (2003), "SEB-tree: An Approach to Index Continuously Moving Objects", Mobile Data Management, January 2003, 340-344.

Spaccapietra, S., Parent, C., Damiani, M. L., Macedo, J. A. d., Porto, F.ve Vangenot, C. (2008), "A conceptual view on trajectories", Data Knowl. Eng. 65(1): 126-146.

Srinivasan, J., Murthy, R., Sundara, S., Agarwal, N.ve DeFazio, S. (2000), "Extensible Indexing: a Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i", Proceedings of the 16th International Conference on Data Engineering, IEEE Computer Society.

Stojanovic, D.ve Dordevic-Kajan, S. (2003), "Modeling and Querying Mobile Objects in Location-Based Services", Facta Universitatis Ser. Math. Inform 18: 59-80.

Sun, J., Papadias, D., Tao, Y.ve Liu, B. (2004), "Querying about the Past, the Present, and the

Future in Spatio-Temporal Databases", Proceedings of the 20th International Conference on Data Engineering, IEEE Computer Society.

Tao, Y.ve Papadias, D. (2001a), "Efficient Historical R-trees", International Conference on Scientific and Statistical Database Management, July 2001, 223-232.

Tao, Y.ve Papadias, D. (2001b), "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries", Proceedings of the 27th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc.

Tao, Y., Papadias, D.ve Sun, J. (2003), "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries", 29th VLDB Conference, Berlin, Germany, ACM.

Tayeb, J., Ulusoy, O.ve Wolfson, O. (1998), "A Quadtree Based Dynamic Attribute Indexing Method", The Computer Journal.

Theodoridis, Y. (2003), "Ten Benchmark Database Queries for Location-based Services", The Computer Journal 46(6): 713-725.

Theodoridis, Y., Silva, J.ve Nascimento, M. (1999), On the Generation of Spatiotemporal Datasets, Advances in Spatial Databases, R.H.Güting, D. Papadiasve F. H. Lochovsky, 1651: 147-164.

Theodoridis, Y., Vazirgiannis, M.ve Sellis, T. (1996), "Spatio-Temporal Indexing for Large Multimedia Applications", ICMCS'96.

Trajcevski, G., Wolfson, O., Hinrichs, K.ve Chamberlain, S. (2004), "Managing uncertainty in moving objects databases", ACM Trans. Database Syst. 29(3): 463-507.

Tryfona, N., Andersen, S., Mogensen, S. R.ve Jensen, C. S. (1999), "A Methodology and a Tool for Spatiotemporal Database Design", Helenic Conference on Informatics, University of Ioannina, 53-60.

Tzouramanis, T., Vassilakopoulos, M.ve Manolopoulos, Y. (2002), "On the Generation of Time-Evolving Regional Data", GeoInformatica 6(3): 207-231.

Wang, X., Zhou, X.ve Lu, S. (2000), "Spatiotemporal Data Modeling and Management: A Survey", Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Asia'00), IEEE Computer Society.

Wolfson, O., Jiang, L., Sistla, A. P., Chamberlain, S., Rishe, N.ve Deng, M. (2002), "Moving Objects Information Management: The Database Challenge(Vision Paper)".

Wolfson, O., Sistla, P., Xu, B., Zhou, J.ve Chamberlain, S. (1999), "DOMINO: Databases fOr MovINg Objects tracking", SIGMOD '99: 547-549.

Wolfson, O., Xu, B., Chamberlain, S.ve Jiang, L. (1998), "Moving Objects Databases: Issues and Solutions", Proceedings of the 10th International Conference on Scientific and Statistical Database Management, IEEE Computer Society, 111-122.

Wood, J. (2002), Java Programming for Spatial Sciences, New York, Taylor & Francis.

Xiong, X.ve Aref, W. G. (2006), "R-trees with update memos", ICDE, 22.

Xu, X., Han, J.ve Lu, W. (1990), "RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Database", International Symposium on Spatial Data Handling, July 1990, 1040-1049.

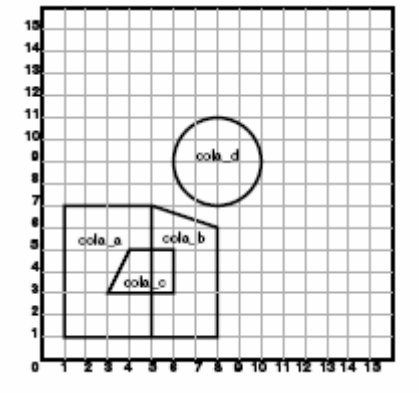
Yiu, M. L., Tao, Y.ve Mamoulis, N. (2008), "The B^{dual}-Tree: indexing moving objects by space filling curves in the dual space", VLDB Journal 17: 379-400.

Zhang, W., Li, J.ve Zhang, W. (2006), "Spatio-temporal Pattern Query Processing based on Effective Trajectory Splitting Models in Moving Object Database", First International Multi-Symposiums on Computer and Computational Sciences(IMSCCS'06), IEEE.

EKLER

Ek 1 Oracle Spatial 10g Veri Tabanı Üzerine İnceleme ve Deneme Çalışmaları

Oracle Spatial 10g veri tabanı yeteneklerini kullanabilmek için bu bölümde Oracle Spatial kaynaklarında kullanılan örnek gerçekleştirilmiştir. Bu örnekte, pazarda en çok satılan kola markaları, Şekil Ek 1.1'de verilen harita üzerinde konumlarına göre belirlenmiştir.



Şekil Ek 1.1 Kola pazarı

Bu haritada bulunan kola firmalarını Oracle 10g veri tabanında tanımlamak ve çeşitli sorgular yapmak için aşağıdaki işlemler sırayla gerçekleştirilecektir:

- COLA_MARKETS isimli tablonun, uzaya ait verileri saklaması için yaratılması,
- İlgilenilen kola markalarının konum bilgilerine göre tabloya eklenmesi (cola_a, cola_b, cola_c, cola_d),
- USER_SDO_GEOM_METADATA'nın boyut bilgilerini tutmak için güncelleştirilmesi,
- Uzaya göre endeksin(COLA_SPATIAL_IDX) yaratılması,
- Uzay sorgularının gerçekleştirilmesi.

Oluşturulacak olan COLA_MARKETS isimli tablodaki her satır, bir kola markası için ilgilenilen bilgiyi tutmaktadır. İlgilenilen bilgi, o coğrafyadaki sakinler tarafından en çok tercih edilen kola markasının olduğu konum veya üretici firmanın potansiyel gördüğü konumlar olabilir. Aşağıda bu tabloyu yaratmak için gerekli SQL verilmiştir:

```
CREATE TABLE cola_markets (
      mkt_id NUMBER PRIMARY KEY,
      name VARCHAR2(32),
      shape SDO_GEOMETRY);
```

Daha sonra A kola markası için ilgilenilen konum, tabloya aşağıdaki şekilde eklenir. Bu alan dikdörtgendir. Dikdörtgeni tanımlamak için iki nokta gerekir: sol alt (1,1) ve sağ üst (5,7) noktaları.

```
INSERT INTO cola_markets VALUES (
   1,
   'cola_a',
      SDO_GEOMETRY(
          2003,                  --2 boyutlu poligon
          NULL,
          NULL,
          SDO_ELEM_INFO_ARRAY(1, 1003, 3)  -- dikdörtgen 1003 dışı
          SDO_ORDINATE_ARRAY(1,1,  5,7)
      )
   );
```

Aynı şekilde B ve C kola markalarının da konumları aşağıdaki şekilde tabloya eklenir. Eklenen bu iki alan birer poligondur. Bir poligon tanımlanırken, başlangıç noktası tekrar tanımlanarak poligon noktaları bitirilir. B kola markası için bu değerler; (5,1, 8,1, 8,6, 5,7 5,1). C kola markası için bu değerler; (3,3, 6,3, 6,5, 4,5 3,3).

```
INSERT INTO cola_markets VALUES (
   2,
   'cola_b',
   SDO_GEOMETRY(
      2003,                  --2 boyutlu poligon
      NULL,
      NULL,
      SDO_ELEM_INFO_ARRAY(1, 1003, 1) --poligon dışı poligon halka
      SDO_ORDINATE_ARRAY(5,1,  8,1,  8,6,  5,7  5,1)
   )
);
INSERT INTO cola_markets VALUES (
   3,
   'cola_c',
   SDO_GEOMETRY(
      2003,                  --2 boyutlu poligon
      NULL,
      NULL,
      SDO_ELEM_INFO_ARRAY(1, 1003, 1)-- poligon dışı poligon halka
      SDO_ORDINATE_ARRAY(3,3,  6,3,  6,5,  4,5  3,3)
   )
);
```

Son olarak da D kola markasının konumu aşağıdaki gibi eklenir. Bu alan, çapı 2 olan bir dairedir. Bir daire üç nokta ile belirlenir.

```
INSERT INTO cola_markets VALUES (
      4,
      'cola_d',
      SDO_GEOMETRY(
            2003,                  --2 boyutlu poligon
            NULL,
            NULL,
            SDO_ELEM_INFO_ARRAY(1, 1003, 4)  -- bir daire
            SDO_ORDINATE_ARRAY(8,7  10,9,  8,11)
      )
   );
```

Uzaya ait endeks yaratılmadan önce, metadata bilgileri güncelleştirilmelidir. Her katman (tablo-sütun ilişkisi. Burada COLA_MARKETS ve SHAPE) için bu işlemin sadece bir defa yapılması gerekmektedir.

```
INSERT INTO user_sdo_geom_metadata
      (TABLE_NAME,
      COLUMN_NAME,
      DIMINFO,
      SRID)
  VALUES (
      'cola_markets',
      'shape',
      SDO_DIM_ARRAY(  -- 20 x 20 grid
            SDO_DIM_ELEMENT ('X', 0, 20, 0.005),
            SDO_DIM_ELEMENT ('Y', 0, 20, 0.005)
            ),
      NULL,   --SRID
);
```

Uzaya ait R_Tree endeksi aşağıdaki şekilde yaratılır;

```
CREATE INDEX cola_spatial_idx
      ON cola_markets(shape)
      INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Endeks yapısı ile Oracle Spatial'ın sunduğu sorgulama kolaylıkları kullanılabilmektedir. İki kola a ve b firmalarının kesişim alanı aşağıdaki SQL ile elde edilmektedir.

```
Select SDO_GEOM.SDO_INTERSECTION ( c_a.shape, c_c.shape,0.005)
   from cola_markets c_a, cola_markets c_c
   WHERE c_a.name='cola_a' AND c_c.name='cola_c';
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
SDO_GEOM.SDO_INTERSECTION(C_A.SHAPE,C_C.SHAPE,0.005)(SDO_GTYPE,
SDO_SRID, SDO_PO
--------------------------------------------------------------------
SDO_GEOMETRY(2003,  NULL,  NULL,  SDO_ELEM_INFO_ARRAY(1,  1003,  1),
SDO_ORDINATE_ARR
AY(4, 5, 3, 3, 5, 3, 5, 5, 4, 5))
```

Belirli bir alanı hesaplama SDO_AREA fonksiyonu ile hesaplanır.

```
select SDO_GEOM.SDO_AREA(shape, 0.005) FROM cola_markets;
```

Bu komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
SDO_GEOM.SDO_AREA(SHAPE,0.005)
------------------------------
                            24
                          16.5
                             5
                   12.5663706
```

Verilen iki geometri arasındaki uzaklığı ölçmek için SDO_DISTANCE fonksiyonu kullanılır.

```
select SDO_GEOM.SDO_DISTANCE(c_b.shape, c_d.shape,0.005)
   from cola_markets c_b, cola_markets c_d
   WHERE c_b.name='cola_b' AND c_d.name='cola_d';
```

Bu komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
SDO_GEOM.SDO_DISTANCE(C_B.SHAPE,C_D.SHAPE,0.005)
-----------------------------------------------
                                    .846049894
```

Verilen bir geometrinin geçerli olması VALIDATE_GEOMETRY_WITH_CONTEXT fonksiyonu ile gerçekleştirilir.

```
select c.name, SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(c.shape,0.005)
   from cola_markets c WHERE c.name='cola_c';
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
NAME
--------------------------------
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(C.SHAPE,0.005)
--------------------------------------------------------------------------------
cola_c
TRUE
```

Verilen bir geometriye en yakın 3 komşuluk aşağıdaki sorgu ile elde edilebilir.

```
select c.mkt_id, c.name from cola_markets c WHERE SDO_NN(c.shape,
   SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(10,7,NULL),
   NULL,NULL),'SDO_NUM_RES=3')='TRUE';
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
    MKT_ID NAME
---------- --------------------------------
         2 cola_b
         3 cola_c
         4 cola_d
```

Verilen bir geometriye en yakın 2 komşunun uzaklığı aşağıdaki sorgu ile elde edilir.

```
select
   c.mkt_id, c.name, SDO_NN_DISTANCE(1) dist
      from cola_markets c
      WHERE SDO_NN(c.shape, SDO_GEOMETRY(2001, NULL,
            SDO_POINT_TYPE(10,7,NULL), NULL, NULL),
            'sdo_num_res=2', 1) ='TRUE' order by dist;
```

Bu komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
    MKT_ID NAME                                 DIST
---------- -------------------------------- ----------
         4 cola_d                          .828427125
         2 cola_b                          2.23606798
```

Verilen bir geometri ile topolojik bir ilişkinin olup olmadığı aşağıdaki sorgu ile elde edilir.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
   WHERE SDO_ANYINTERACT(c.shape,
      SDO_GEOMETRY(2003, NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(4,6, 8,8))
   ) = 'TRUE';
```

Bu komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
    MKT_ID NAME
---------- --------------------------------
         2 cola_b
         1 cola_a
         4 cola_d
```

Verilen bir geometride CONTAINS topolojik ilişkisi aşağıdaki sorgu ile elde edilir.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
   WHERE SDO_CONTAINS(c.shape,
      SDO_GEOMETRY(2003, NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(2,2, 4,6))
   ) = 'TRUE';
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
    MKT_ID NAME
---------- --------------------------------
         1 cola_a
```

Aynı şekilde SDO_COVEREDBY, SDO_COVERS, SDO_EQUAL, SDO_INSIDE, SDO_ON, SDO_OVERLAPBDYDISJOINT, SDO_OVERLAPS, SDO_TOUCH topolojik ilişkileri benzer sorgular ile incelenebilir. 10 birimlik alan içinde olan geometriler aşağıdaki sorgu ile elde edilebilir.

```
SELECT c.name FROM cola_markets c WHERE SDO_WITHIN_DISTANCE(c.shape,
   SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
   SDO_ORDINATE_ARRAY(4,6, 8,8)),
   'distance=10') = 'TRUE';
```

İki çizgi aşağıdaki gibi yaratılarak, daha sonra birleştirilebilir.

```
INSERT INTO cola_markets VALUES(1001, 'line_1', SDO_GEOMETRY(2002,
NULL, NULL,
SDO_ELEM_INFO_ARRAY(1,2,1), SDO_ORDINATE_ARRAY(1,1, 5,1)));
INSERT INTO cola_markets VALUES(1002, 'line_2', SDO_GEOMETRY(2002,
NULL, NULL,
SDO_ELEM_INFO_ARRAY(1,2,1), SDO_ORDINATE_ARRAY(5,1, 8,1)));
-- Perform aggregate concatenation of all line geometries in layer.
SELECT SDO_AGGR_CONCAT_LINES(c.shape) FROM cola_markets c
WHERE c.mkt_id > 1000;
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
SDO_AGGR_CONCAT_LINES(C.SHAPE)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y,
Z), SDO_ELEM
--------------------------------------------------------------------
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
SDO_ORDINATE_ARRAY(
1, 1, 5, 1, 8, 1))
```

Convex hull alanı aşağıdaki grup(aggregate) fonksiyon ile elde edilebilir.

```
SELECT SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(shape, 0.005))
FROM cola_markets;
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(SHAPE,0.005))(SDO_GTYPE,     SDO_SRID,
SDO_POINT(X,
--------------------------------------------------------------------------
SDO_GEOMETRY(2003,  NULL,  NULL,  SDO_ELEM_INFO_ARRAY(1,  1003,  1),
SDO_ORDINATE_ARR
AY(8, 1, 10, 7, 10, 11, 8, 11, 6, 11, 1, 7, 1, 1, 8, 1))
```

MBR, aşağıdaki grup fonksiyon ile elde edilebilir.

```
SELECT SDO_AGGR_MBR(shape) FROM cola_markets;
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

```
SDO_AGGR_MBR(SHAPE)(SDO_GTYPE,    SDO_SRID,    SDO_POINT(X,    Y,    Z),
SDO_ELEM_INFO, SDO_
--------------------------------------------------------------------------
SDO_GEOMETRY(2003,  NULL,  NULL,  SDO_ELEM_INFO_ARRAY(1,  1003,  3),
SDO_ORDINATE_ARR
AY(1, 1, 10, 11))
```

İki marketin bileşimi aşağıdaki alt programla elde edilebilir.

```
SELECT SDO_AGGR_UNION(
SDOAGGRTYPE(c.shape, 0.005))
FROM cola_markets c
WHERE c.name < 'cola_d';
```

Kola marketteki topolojik ilişkiler aşağıdaki gibi elde edilebilir.

```
SELECT c.name,
  SDO_GEOM.RELATE(c.shape, 'determine', c_b.shape, 0.005) relationship
      FROM cola_markets c, cola_markets c_b WHERE c_b.name = 'cola_b';
```

Yukarıdaki komut kullanıldığında, aşağıdaki gibi sorgu cevabı elde edilir.

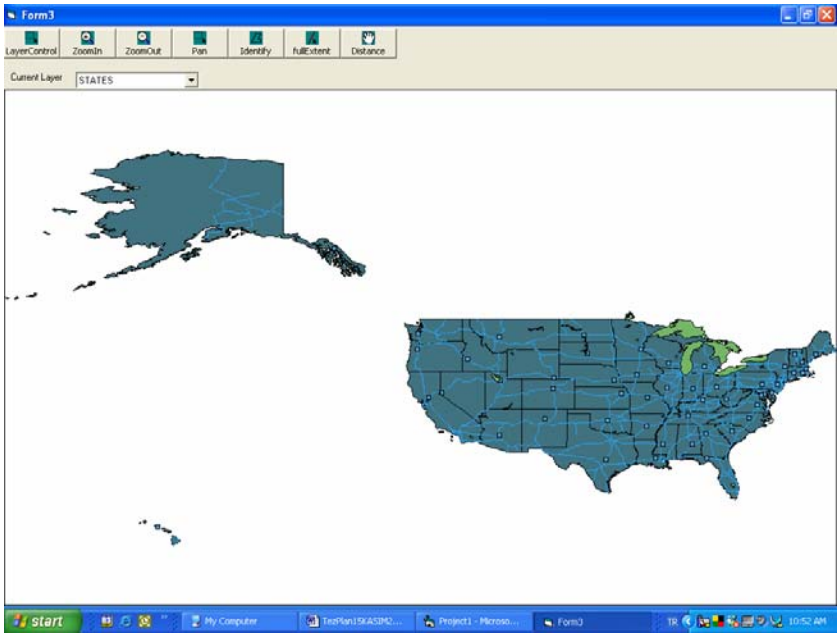
```
NAME
--------------------------------
RELATIONSHIP
--------------------------------------------------------------------------
cola_a
TOUCH

cola_b
EQUAL

cola_c
OVERLAPBDYINTERSECT


NAME
--------------------------------
RELATIONSHIP
--------------------------------------------------------------------------
cola_d
DISJOINT
```

Ek 2 VBasic ile MapObjects Üzerine İnceleme ve Deneme Çalışmaları

VisualBasic programlama dilinin kolay programlama özelliklerinin olmasından ve sayısal harita üzerinde gerçekleştirilen uygulamaları görsel olarak inceleyerek araştırmak için, tez çalışmasının başlangıcında deneme programları gerçekleştirilmiştir.

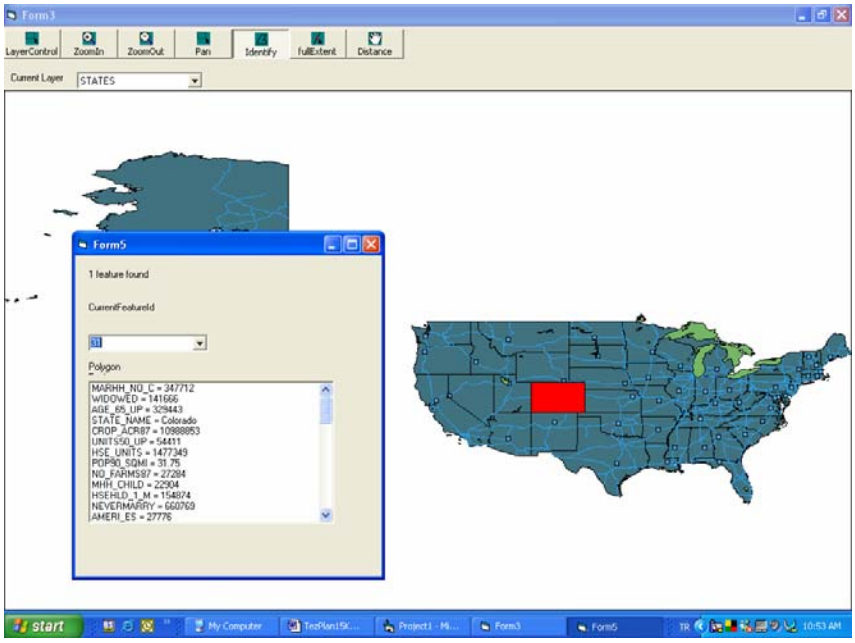
1997 yılından itibaren oluşturulan Coğrafi Bilgi Sistemleri verileri en çok ESRI firması tarafından geliştirilen ArcView uygulamasında ve Oracle veri tabanında saklanmıştır (Ralston, B. A. 2002). 1998 yılından itibaren de ArcView, Oracle formatına çevrilmiştir. Bu firma, MapObjects ile katmanlar şeklinde sayısal haritayı hazır alt programlarla görselleştirebilmektedir. Bu bölümde geliştirilen küçük uygulamanın olanakları tanıtılacaktır.

Visul Basic programlama dilinde eklenen yazılım bileşenleri (component) ile sayısal haritanın görüntülenmesi sağlanmaktadır. Amerika sayısal haritasına ait katmanlardaki verileri sorgulama ekranı Şekil Ek 2.1'de verilmiştir. Bu ekranın araç çubuklarında, **LayerControl**, **ZoomIn**, **ZoomOut**, **Pan**, **Identify**, **fullExtent** ve **Distance** düğmeleri bulunmaktadır.



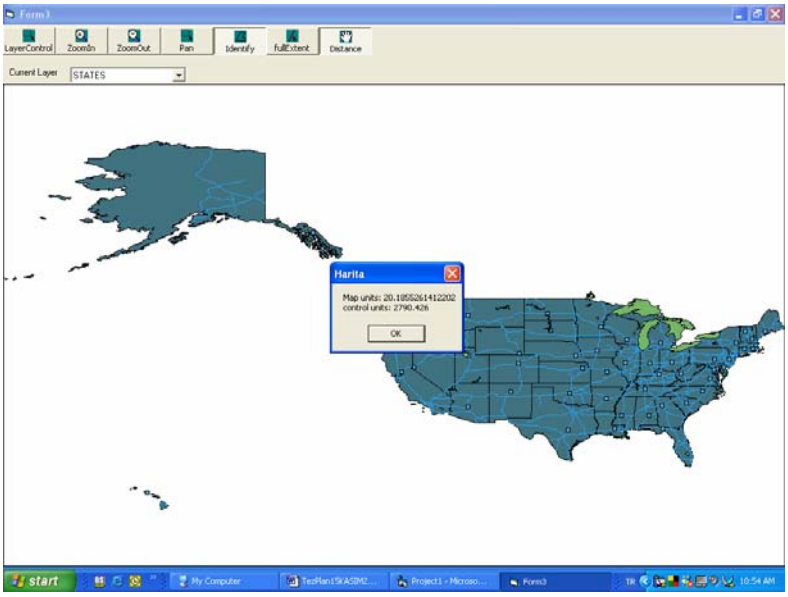
Şekil Ek 2.1 MapsObjects ile sayısal harita

Bir ilin adı, nüfus sayımı gibi genel bilgilerini görüntülemek için **Identify** düğmesi kullanılmaktadır (Şekil Ek 2.2).



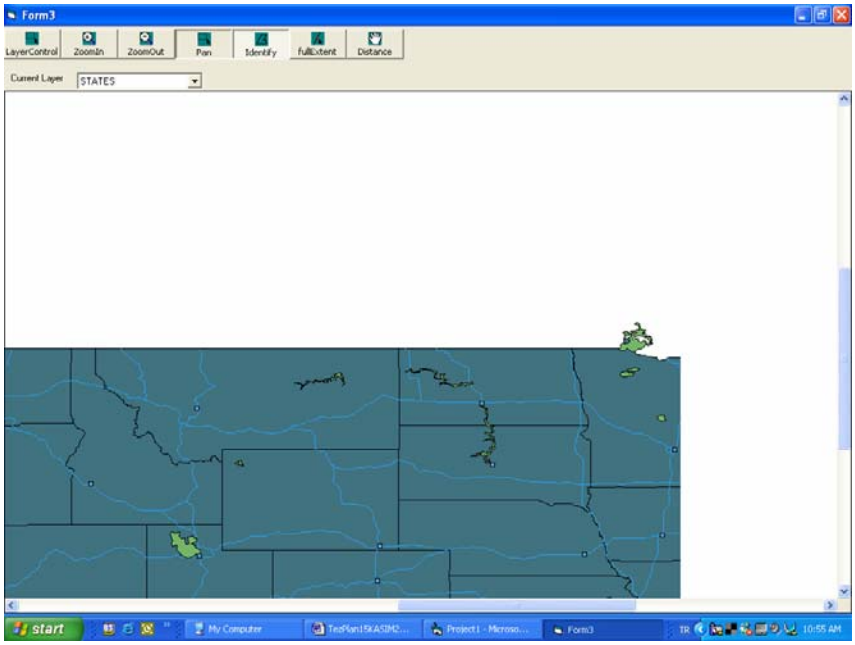
Şekil Ek 2.2 Bir ilin bilgileri

Fare ile belirlenen iki nokta arasındaki uzaklığı hesaplama **Distance** düğmesi seçilerek, oluşturulan ekran Şekil Ek 2.3'te verilmiştir.



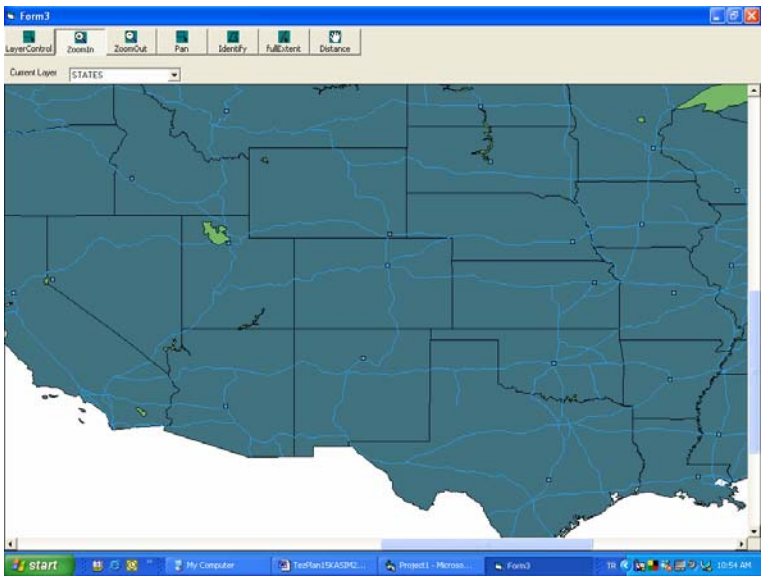
Şekil Ek 2.3 İki nokta arasındaki uzaklık

Pan düğmesi kullanılarak, haritanın bir bölümü bir yerden bir yere taşınabilmektedir. Bu ekran Şekil Ek 2.4'te verilmiştir.



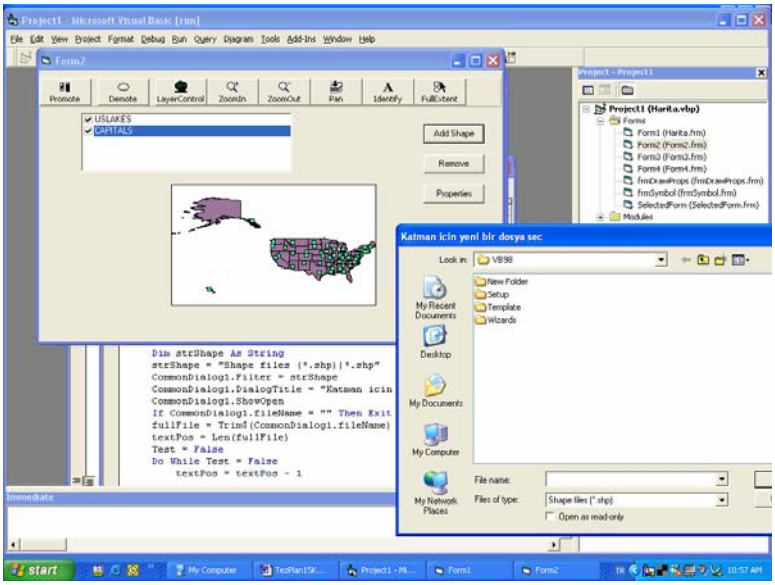
Şekil Ek 2.4 Pan özelliği

ZoomIn ve **ZoomOut** düğmeleri ile, haritanın belirlenen bir alanına daha yakından bakma veya uzaklaşma sağlanmaktadır. Yakınlaştırma ekranı, Şekil Ek 2.5'te verilmiştir.



Şekil Ek 2.5 Zoom özelliği

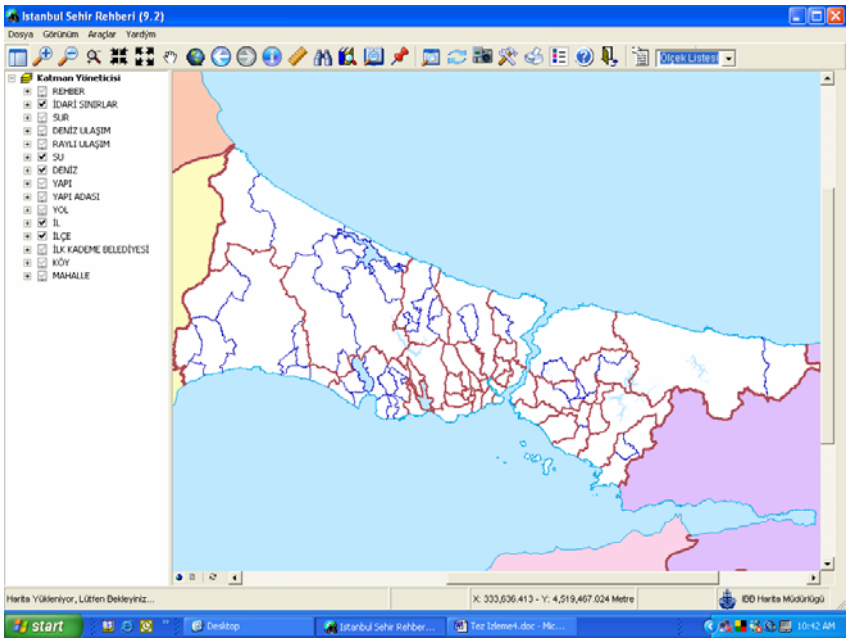
İstenen bir katmanı, ilgili dosyadan seçerek ekleme özelliği, Şekil Ek 2.6'da verilmiştir.



Şekil Ek 2.6 Haritaya katman ekleme

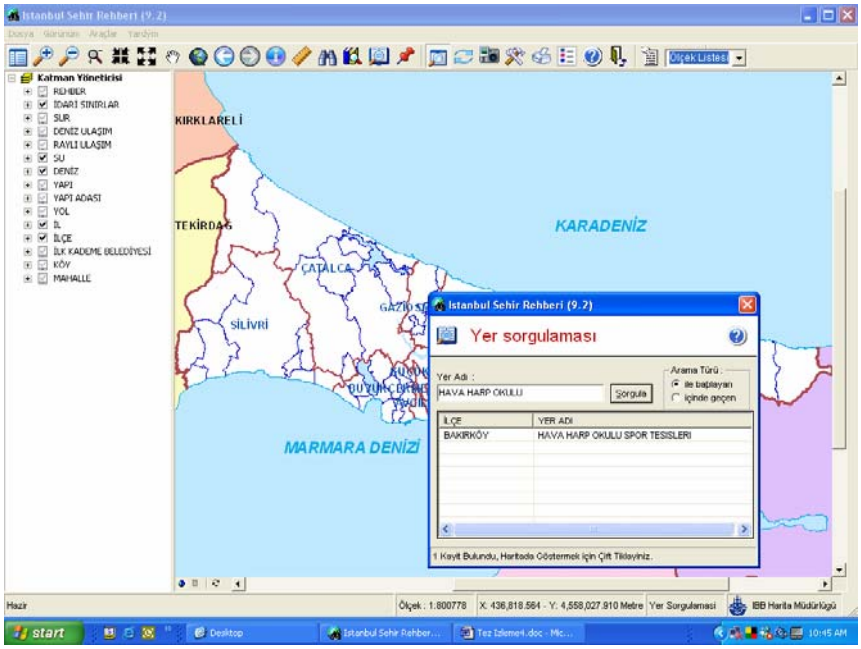
Ek 3 İstanbul Büyükşehir Belediyesinden Temin Edilen Konuma Bağlı Uygulama

Bu bölümde, İstanbul Büyükşehir Belediyesinden temin edilen konuma bağlı Şehir Rehberi uygulaması tanıtılmıştır (Şekil Ek 3.1).



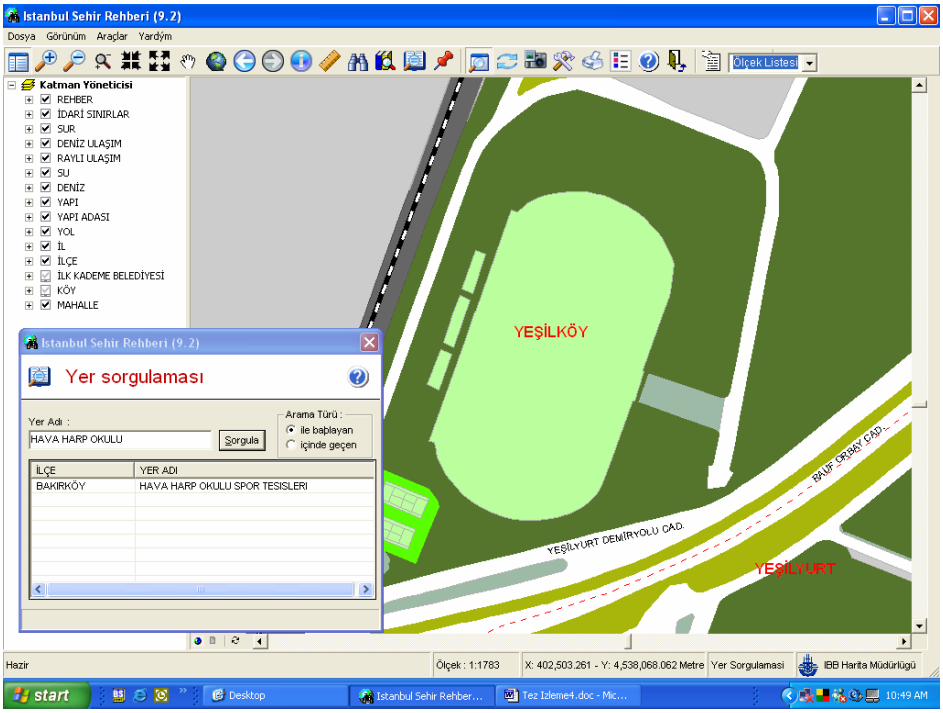
Şekil Ek 3.1 İstanbul şehir rehberi

Bu rehber, bir yer bilgisine göre sorgulamayı sağlamaktadır (Şekil Ek 3.2).



Şekil Ek 3.2 İstanbul şehir rehberi sorgulama ekranı

Sorgu sonucunda elde edilen ekran Şekil Ek 3.3'te gösterilmiştir.



Şekil Ek 3.3 İstanbul şehir rehberi sorgulama sonucu

Ek 4 Bazı Algoritmaların PL/SQL Karşılıkları

```
--find_pid fonksiyonu
CREATE OR REPLACE FUNCTION find_pid(x NUMBER, y NUMBER) return NUMBER
deterministic IS

x_grid number;
y_grid number;
n number;
bolum number;
sonuc number;
begin
   n:=15;       --8      15        16     32
   bolum:=937.5;--3750   2000      1875   937.5

   x_grid:=floor(x/bolum);
   y_grid:=floor(y/bolum);

   if x_grid=n then
      x_grid:=x_grid-1;
   end if;
   if y_grid=n then
      y_grid:=y_grid-1;
   end if;
   sonuc:=x_grid+y_grid*n;
   return sonuc;
end;
```

```
--St_zorder_5m random sorgu pencereleri ile t%20 sorgu
--yaratılması
DECLARE
i number;
xx number;
yy number;
t number;
n number;
tt number;
xxbas number;
yybas number;
ttson number;
uzayArtim number;
zamanArtim number;
z1 number;
z2 number;

BEGIN
   uzayArtim:=3000;--uzay(s) yuzde 0.25, 1, 4 :  1500,3000,6000
   zamanArtim:=4;  --zaman(t) yuzde 5, 10, 20:  1, 2, 4

   n:=100;
   FOR i in 1..n LOOP
      xx:=ceil(dbms_random.value(6000+uzayArtim,24000));
      yy:=ceil(dbms_random.value(6000+uzayArtim,24000));
      t:=ceil(dbms_random.value(zamanArtim,20-zamanArtim));
      tt:=t-zamanArtim;
      ttson:=t+zamanArtim;
      xxbas:=xx-uzayArtim;
      yybas:=yy-uzayArtim;
      z1:=z_order(xxbas, yybas);
      z2:=z_order(xx,yy);

      dbms_output.put_line(' SELECT unique id FROM ST_ZORDER_5M WHERE ');
      dbms_output.put_line('z_order(x,y) between ' || z1||' and '|| z2);
      dbms_output.put_line(' AND ts between '|| tt ||' and '||t||' AND ');
      dbms_output.put_line(' te between '|| t ||' and '||ttson);
      dbms_output.put_line(' AND x between '|| xxbas||' and '||xx);
      dbms_output.put_line(' AND y between '|| yybas||' and '||yy||';');

   END LOOP;

END;
```

Ek 5 MOORA Veri Kartuşu

Bu tez çalışmasında tasarlanıp, gerçekleştirilen veri kartuşu özelliklerinin ve işlemlerinin PL/SQL tanımlamaları aşağıda verilmiştir.

Zaman boyutu için tanımlar:

```
CREATE TYPE T_PERIOD AS OBJECT(
  FRM DATE,  --INSTANT
  TOO DATE,  --INSTANT
  MEMBER FUNCTION M_BEFORE(P T_PERIOD) RETURN BOOLEAN,
  MEMBER FUNCTION M_EQUAL(P T_PERIOD) RETURN BOOLEAN,
  MEMBER FUNCTION M_MEET(P T_PERIOD) RETURN BOOLEAN,
  MEMBER FUNCTION M_OVERLAP(P T_PERIOD) RETURN BOOLEAN,
  MEMBER FUNCTION M_DURING(P T_PERIOD) RETURN BOOLEAN,
  MEMBER FUNCTION M_START(P T_PERIOD) RETURN BOOLEAN,
  MEMBER FUNCTION M_FINISH(P T_PERIOD) RETURN BOOLEAN);
```

Uzay boyutu için tanımlar:

```
CREATE TYPE T_POINT AS OBJECT(
  GEOMETRI SDO_GEOMETRY,
  MEMBER FUNCTION M_DISJOINT(POI IN T_POINT) RETURN BOOLEAN,
  MEMBER FUNCTION M_EQUAL(POI IN T_POINT) RETURN BOOLEAN,
  MEMBER FUNCTION M_DISTANCE(POI IN T_POINT) RETURN FLOAT);
```

Hem Uzay Hem Zaman için tanımlar:

```
CREATE TYPE T_MOVING AS OBJECT(
  ID NUMBER,
  KONUM T_POINT,
  ZAMAN T_PERIOD,
  MEMBER FUNCTION M_WHEN(INS IN DATE) RETURN T_POINT);
```

ÖZGEÇMİŞ

Doğum tarihi	14.01.1974	
Doğum yeri	Elazığ	
Lise	1987-1990	İzmir Şemikler Lisesi
Lisans	1990-1995	Ege Üniversitesi Mühendislik Fakültesi, Bilgisayar Bilimleri Mühendisliği Bölümü
Yüksek Lisans	1997-2001	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Müh. Anabilim Dalı, Bilgisayar Müh. Programı

Çalıştığı Kurumlar

1995-1997	Göktepe Plastik San. ve Tic. A.Ş
1997	İZTEK A.Ş.
1997-Devam ediyor	Hava Harp Okulu Komutanlığı Bilgisayar Mühendisliği Bölümü Öğretim Elemanı