

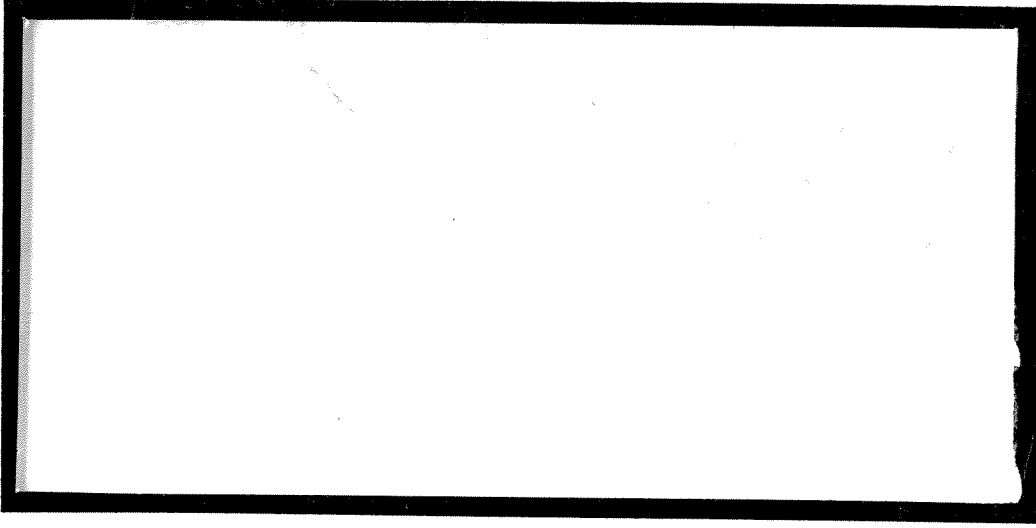
1997-132

Dup



TÜRKİYE BİLİMSEL VE
TEKNİK ARAŞTIRMA KURUMU

THE SCIENTIFIC AND TECHNICAL
RESEARCH COUNCIL OF TURKEY



Elektrik, Elektronik ve Enformatik Araştırma Grubu

Electric, Electronics and Informatics Research
Grant Committee

NESNESEL VERİ TABANI YÖNETİM SİSTEMİ
PROTOTİPİ

PROJE NO: YAZILIM 1

191E013

PROF. DR. ASUMAN DOĞAÇ
ÇETİN ÖZKAN
CEM EVRENDİLEK
MEHMET ALTINEL
BUDAK ARPINAR

EYLÜL 1993
ANKARA

TÜBİTAK YAZILIM ARAŞTIRMA ve GELİŞTİRME ÜNİTESİ

YAZILIM 1 PROJESİ SONUÇ RAPORU

Önsöz

Yazılım 1 projesi kapsamında geliştirilen MOOD (METU Object-Oriented DBMS: ODTÜ Nesneye-Yönelik Veritabanı Yönetim Sistemi), 1991 Yılı Ekim ayında başlamıştır. Bu proje TÜBİTAK tarafından ODTÜ Bilgisayar Mühendisliği bölümünde kurulan Yazılım Araştırma ve Geliştirme Ünitesinde gerçekleştirilmiştir. Nesne-yönelimi teknolojisinin veri tabanı teknolojisine uyarlanması sonucu, mühendislik uygulamalarının gerektirdiği grafik içeren karmaşık veri yapılarını modellemek ve bunlara etkin ulaşım sağlayacak veri tabanı sistemini geliştirmek mümkün olmuştur.

İçindekiler

1. Öz.....	1
2. Proje Ana Metni	2
3. Projenin Türkiye'deki yazılım araştırma ve geliştirme çalışmalarına katkıları.....	5
4. Projede gerçekleştirilen yayınların listesi	5
5. Yayınlanmak üzere gönderilen teknik raporlar.....	6
6. Proje kapsamında tamamlanmış olan tezler	6
7. Proje kapsamında tamamlanmak üzere olan tezler	7
8. MOOD projesinin birlikte demo edildiği diğer sistemler	7
9. Ekler.....	9
Ek1. MOOD User Manual	
Ek2. METU Object-Oriented DBMS Kernel	
Ek3. A Heuristic Approach for Optimization of Object-Oriented Query Languages	
Ek4. Nesne-yönelimli veri tabanları konusunda dünyada yapılan çalışmaların bir özeti	

Öz

Nesne-yönelimi yaklaşımı kullanan MOOD sisteminin sağladığı başlıca teknik olanaklar şöyledir:

- 1) Grafik, fotoğraf, ses gibi çoklu ortam verisi de içerebilen karmaşık veri yapılarının temsil ve etkin bir şekilde işleme olanağı
 - 2) Bu verilere grafik ortamda erişimi sağlayan ve Motif yazılımı kullanılarak geliştirilmiş bir grafik kullanıcı arabirimi (Graphical User Interface), MoodView
 - 3) Yine veri erişimi ve güncleme amacıyla kullanılan bir nesne yönelimli SQL dili, MoodSQL
 - 4) C++ dili ile tanımlanmış olan fonksiyonların, SQL dili içerisinden dinamik olarak çağırılmasını sağlayan bir dinamik fonksiyon bağlayıcısı (Dynamic Function Linker)
 - 5) SQL sorgularının sistemde kullandığı kaynakları en aza indirmek amacıyla geliştirilmiş bir SQL sorgu en iyileyicisi (Query Optimizer). Bu sistem Colorado Üniversitesinde geliştirilmiş bulunan Volcano Query Optimizer Generator yazılımı kullanılarak geliştirilmiştir.
 - 6) Sistem kataloglarını yöneten birim, MOOD Catalog Manager
 - 7) Veri erişim komutlarını disk yönetim sistemi komutlarına dönüştüren MOOD Algebra
 - 8) Çok boyutlu verilere etkin erişimi sağlamak amacıyla, R ve R* tree ve ikonik indeksleme yöntemleri
- Bütün bu sistemler Wisconsin-Madison Üniversitesinde geliştirilmiş bulunan Exodus Storage Manager isimli disk

yönetimi yapan sistem üzerine geliştirilmiştir. Projenin ilk safhalarında ayrıca Borland C++ "persistent" hale getirilmiştir

Proje Ana Metni

Projeyi oluşturan belli başlı sistemlerin özet olarak fonksiyonları şöyledir:

1. MOOD sistemi için Motif yazılımı üzerinde, kullanıcıların sisteme grafik ortamda İngilizce veya Almanca ulaşmalarını sağlayacak bir grafik kullanıcı arabirimi geliştirilmiştir. MOODView ile ilgili olarak yapılan çalışmaların detayları aşağıda belirtilen Yüksek Lisans tezinde ve makalede verilmiştir:

Tez:

- Arpınar, B., " An Advanced Graphical User Interface for Object- Oriented DBMSs: MoodView", Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Eylül 1993.

Makale:

- Arpınar, B., Doğaç, A., Evrendilek, C., "MoodView: An Advanced Graphical User Interface for OODBMSs", in ACM SIGMOD Record, Vol.22, No.4, December 1994.

2. MOOD projesinin nesne yönelimli veri erişim dili olan MOODSQL tasarlanmış, gerçekleştirilmiş ve bu dilin en iyilenmesini sağlayan iki yazılım, biri proje kapsamında geliştirilen yöntemlerle, diğeri Colorado Üniversitesinin ürünü olan Volcano Query Optimizer kullanılarak gerçekleştirilmiştir. Bu konuda yapılan çalışmaların detayları aşağıda belirtilen Yüksek Lisans tezleri ile tebliğlerde verilmiştir.

Tezler:

- İlker Durusoy, MOOD Query Optimizer, Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Şubat 1994.

- Cetin Ertan, Design and Implementation of an Object-Oriented Query Language, MOODSQL, and its Optimizer, Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Eylül 1993. **Bu tez 1993 yılı Mustafa Parlar Vakfı tez ödülünü almıştır.**

- Pınar Köksal, Query Optimization Through Optimization Regions, Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, bu tez çalışması yazma aşamasındadır.

Tebliğler:

- Ç. Özkan, A. Doğaç, C. Evrendilek, T. Geşli, "Efficient Ordering of Path Traversals in Object-Oriented Query Optimization", in Proc. of Eight Intl. Symposium on Computer and Information Sciences, Istanbul, November, 1993.

- Ç. Özkan, A. Doğaç, Y. Saygın, "A Heuristic Approach for Optimization of Path Expressions in Object-Oriented Query Languages", submitted for publication.

3. Nesne yönelimli bir veri tabanı çekirdek sistemi (Object- Oriented Database Kernel): Bu sistem Exodus Storage Manager üzerinde çalışmakta ve Exodus Storage Manager'ın sağladığı fonksiyonlara ilave olarak şu fonksiyonların gerçekleştirimini içermektedir:

- i. SQL komutlarının interpret edilmesi
- ii. Metodların dinamik olarak sisteme bağlanması
- iii. Katalog yönetimi

Bu konuda yapılan çalışmaların detayları aşağıda belirtilen tezler ve tebliğler içerisinde verilmiştir:

Tezler:

- Okay, T., "Design and Implementation of an Object-Oriented Database Management System Kernel", Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Eylül 1993.
- Altinel, M., "MOOD Algebra and Dynamic Function Linking in MOOD", Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Eylül 1994.

Tebliğler:

- Evrendilek, C., Toroslu, H., "Inheritance and Object Algebra in an Object-Oriented Data Model", 6th Intl. Conference on Computing and Information, Peterborough Canada, May 1994.
- A. Doğaç, M. Altinel, I. Durusoy, Ç. Özkan, "METU Object-Oriented DBMS Kernel", submitted for publication.

4. Nesne yönelimli veri tabanı yönetim sistemlerinin etkin çalışmasının sağlanabilmesi, birlikte erişilen verilerin disk üzerinde fiziksel olarak yakın depo edilmesini gerektirmektedir. Bu konuda yapılan çalışmaların detayı aşağıda belirtilen yayınlar içerisinde verilmiştir:

Tez:

- Kadir Koç, "Comparison of Clustering Algorithms in a Single User Environment", Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Haziran 1993.

Tebliğler:

- Koc, K., Evrendilek, C., Dogac, A., "Comparison of Clustering Algorithms in a Single User Environment through Sun Benchmark", in Proc. of European Joint Conference on Engineering Systems Design and Analysis, London, July 1994.
- Koc, K., Dogac, A., Evrendilek, C., "Comparison of Clustering Algorithms in a Single User Environment through OO7 Benchmark", in Proc. of East-West Database Workshop, Klagenfurt, September 1994.

5. MOOD projesinde çok boyutlu verilerin etkin bir şekilde işlenmesine olanak sağlamak üzere "spatial indexing" teknikleri olarak bilinen R ve R* tree veri yapıları ile "iconic indexing" tekniği EXODUS sistemi üzerinde gerçekleştirilmiştir. Bu çalışmaların detayı aşağıda belirtilen yayınlarda verilmiştir:

Tezler:

- Yürüten, F." Indexing Methods for Spatial Data Objects and their Implementation on the Exodus Storage Manager", Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Haziran 1993.

- Ulu, C., "Iconic Indexing on Exodus Storage Manager", Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Eylül 1994.

MOOD sistemi ile görüntü erişimini kolaylaştırmak amacıyla geliştirilen destek yazılımlarının detayı aşağıda belirtilen Yüksel Lisans tezinde verilmiştir:

Tez:

- Tolga Gesli, Image Data Management in MOODS: METU Object- Oriented Database Management System, Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Eylül 1992.

6. Nesnesel veri tabanları için veri tabanı tasarımı yapan bir yazılım geliştirilmiştir. Bu konuda yapılan çalışmaların detayı aşağıda belirtilen tezde verilmiştir:

Tez:

- Gökmenler, S., "MOOD Database Design Tool", TÜBİTAK Software Research and Development Center, Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, bu tez çalışması yazma aşamasındadır.

7. Projenin ilk safhasında nesne-yönelimli veri tabanı geliştirmeye diğer bir yaklaşım olan C++ dili nesnelerine kalıcılık ("persistency") kazandırmak yaklaşımı denenmiş ve bu amaçla Borland C++ "persistent" hale getirilmiştir. Bu konuda yapılan çalışmaların detayları aşağıda belirtilen Yüksek Lisans tezleri ile tebliğlerde verilmiştir.

Tezler:

- Cem Evrendilek, Persistent C++ in DOS Environment, Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Eylül 1992. **Bu tez 1992 yılı Mustafa Parlar Vakfı tez ödülünü almıştır.**

- Yüksel Saygın, MOODS Storage System, Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü, ODTÜ, Ağustos 1992.

Tebliğ:

- C. Evrendilek, A. Doğaç, İ. Durusoy, Ç. Özkan, Y. Saygın, and M. Altınel, " Persistent C++ in DOS Environment ", in Proc. of Seventh Intl. Symposium on Computer and Information Sciences, Antalya, November, 1992.

7. Bütün bu sistemlerin entegrasyonu ile MOOD sisteminin bir prototipi elde edilmiştir. Sistemin tamamının anlatıldığı yayınlar şöyledir:

Tebliğler:

- Doğaç, A., Özsu, T., Biliris, A., Sellis, T., (Edtrs.) Object-Oriented Database Systems, Springer-Verlag, 1994.

- Doğaç, A., Evrendilek, C., Okay, T., Ozkan, C., "METU Object- Oriented DBMS", in Object-Oriented Database Systems, edited by Doğaç, A., Özsu, T., Biliris, A., Sellis, T., pp.172-198.

- Doğaç, A., Arpınar, B. Evrendilek, C., Ozkan, C., Altıntaş, I., Durusoy, I., Altınel, M., Okay, T., Saygın, Y., "METU Object- Oriented Database System", Demo description, in Proc. of ACM SIGMOD Intl. Conf. on Management of Data, Minneapolis, May 1994.

- Doğaç, A., "The MOOD User Manual", Mayıs 1994.

Projenin Türkiye'deki yazılım araştırma ve geliştirme çalışmalarına katkıları:

Yaklaşık 90.000 satır kod içeren bu büyük çaplı projenin Türkiye'deki yazılım araştırma ve geliştirme çalışmalarına katkıları özet olarak şöyledir:

1. İleri teknoloji alanında kapsamlı bir araştırma prototipi bütün fonksiyonları ile çalışır halde tamamlanmıştır. Bu proje 1994 Tübitak Hüsamet'tin Tuğaç Araştırma ödülünü almıştır.
2. İlk defa ülkemizde geliştirilen bir yazılım, yurt dışında demo edilmiştir. MOOD yazılımı 23-26 Mayıs tarihleri arasında ABD'nin Minneapolis kentinde yapılan "ACM SIGMOD International Conference on Management of Data" konferansına demo edilmek üzere gönderilen 30 sistemden kabul edilen 15'i arasına girmiştir. Bu konferansta ürünleri demo edilen şirketler arasında IBM, AT&T ve Texas Instruments gibi büyük şirketler ve Avrupa'da yürümekte olan ESPRIT projesi ürünleri bulunuyordu.
3. Bu projede yapılan çalışmalar sonucunda 1 makale, 9 tebliğ yayınlanmış, ve yine projedeki gelişmeleri uluslararası düzeyde tanıtmak üzere 6-15 Ağustos 1993 tarihleri arasında, proje grubu tarafından düzenlenen "Object-Oriented Database Systems" konulu NATO Yaz Okulunun sonucunda MOOD projesinin de yer aldığı kitap 1994 yılında Springer Verlag tarafından basıma girmiştir. Ayrıca proje ile ilgili tamamlanmış araştırmaların yayın çalışmaları devam etmektedir.
4. Bu proje kapsamında 11 Yüksek Lisans tezi tamamlanmış, tezlerden iki tanesi 1992, ve 1993 yılları Mustafa Parlar Vakfı tez ödülleri almışlardır. Ayrıca 2 Yüksek Lisans tezide tamamlanma aşamasındadır.

Proje'nin "source code"u da dahil olmak üzere bütün makale ve tebliğler ile bazı tezlerin kopyalarına WWW'deki URL:<http://www.srdc.metu.edu.tr/> adresinden ulaşılabilir.

Projede gerçekleştirilen yayınların listesi

ÖDÜL

- Proje 1994 Tübitak Hüsametlin Tuğaç Araştırma ödülünü almıştır.

KİTAP

- Doğaç, A., Özsu, T., Biliris, A., Sellis, T., Object-Oriented Database Systems, Springer-Verlag, 1994.

MAKALE

- Arpınar, B., Doğaç, A., Evrendilek, C., "MoodView: An Advanced Graphical User Interface for OODBMSs", in ACM SIGMOD Record, Vol.22, No.4, December 1994.

TEBLİĞLER

- 1- Doğaç, A., Arpınar, B., Evrendilek, C., Ozkan, C., Altıntaş, I., Durusoy, I., Altınel, M., Okay, T., Saygın, Y., "METU Object- Oriented Database System", Demo description, in Proc. of ACM SIGMOD Intl. Conf. on Management of Data, Minneapolis, May 1994.
- 2- Doğaç, A., Evrendilek, C., Okay, T., Ozkan, C., "METU Object- Oriented DBMS", in Object-Oriented Database Systems, edited by Doğaç, A., Özsu, T., Biliris, A., Sellis, T., pp.172-198, 1994.
- 3- Koc, K., Evrendilek, C., Dogac, A., "Comparison of Clustering Algorithms in a Single User Environment through Sun Benchmark", in Proc. of European Joint Conference on Engineering Systems Design and Analysis, London, July 1994.
- 4- Koc, K., Dogac, A., Evrendilek, C., "Comparison of Clustering Algorithms in a Single User Environment through OO7 Benchmark", in Proc. of East-West Database Workshop, Klagenfurt, September 1994.
- 5- Evrendilek, C., Toroslu, H., "Inheritance and Object Algebra in an Object-Oriented Data Model", 6th Intl. Conference on Computing and Information, Peterborough Canada, May 1994.
- 6- Ç. Özkan, A. Doğaç, C. Evrendilek, T. Geşli, "Efficient Ordering of Path Traversals in Object-Oriented Query Optimization", in Proc. of Eight Intl. Symposium on Computer and Information Sciences, Istanbul, November, 1993.
- 7- C. Evrendilek, A. Doğaç, İ. Durusoy, Ç. Özkan, Y. Saygın, and M. Altınel, " Persistent C++ in DOS Environment ", in Proc. of Seventh Intl. Symposium on Computer and Information Sciences, Antalya, November, 1992.
- 8- Doğaç, A., "The MOOD User Manual", Mayıs 1994.
- 9- Aytekin, H., Dogac, A., "A distributed Parallel Object manager for Smalltalk", in Proc. of ninth Intl. Symposium on Computer and Information Sciences, Antalya, November, 1994.

Yayınlanmak üzere gönderilen teknik raporlar:

1. A. Doğaç, M.Altinel, I. Durusoy, Ç. Özkan, "METU Object-Oriented DBMS Kernel", submitted for publication.
2. Ç. Özkan, A. Doğaç, Y. Saygın, "A Heuristic Approach for Optimization of Path Expressions in Object-Oriented Query Languages", submitted for publication.

Proje kapsamında tamamlanmış olan tezler:

1. Ilker Durusoy, MOOD Query Optimizer, M.S. Thesis, Dept. of Computer Eng., February 1994.
2. Tansel Okay, Design and Implementation of an Object-Oriented Database Management System Kernel, M.S. Thesis, Dept. of Computer Eng., September 1993.
3. Budak Arpınar, An Advanced Graphical User Interface for Object-Oriented DBMSs: MoodView, M.S. Thesis, Dept. of Computer Eng., September 1993.
4. Cetin Ertan, Design and Implementation of an Object-Oriented Query Language, MOODSQL, and its Optimizer, M.S. Thesis, Dept. of Computer Eng., September 1993. **Bu tez 1993 yılı Mustafa Parlar Vakfı tez ödülünü almıştır.**
5. Fusun Yuruten, Indexing Methods for Spatial Data Objects and their Implementation on the Exodus Storage Manager, M.S. Thesis, Dept. of Computer Eng., June 1993.
6. Kadir Koç, Comparison of Clustering Algorithms in a Single User Environment, M.S. Thesis, Dept. of Computer Eng., June 1993.
7. Cem Evrendilek, Persistent C++ in DOS Environment, M.S. Thesis, Dept. of Computer Eng., September 1992. **Bu tez 1992 yılı Mustafa Parlar Vakfı tez ödülünü almıştır.**
8. Tolga Gesli, Image Data Management in MOODS: METU Object-Oriented Database Management System, M.S. Thesis, Dept. of Computer Eng., September 1992.
9. Yüksel Saygın, MOODS Storage System, M.S. Thesis, Dept. of Computer Eng., August 1992.
10. Altinel, M., "MOOD Algebra and Dynamic Function Linking in MOOD", M.S. Thesis, Dept. of Computer Eng., September 1994.
11. Ulu, C., "Iconic Indexing on Exodus Storage Manager", M.S. Thesis, Dept. of Computer Eng., September 1994.

Proje kapsamında tamamlanmak üzere olan tezler:

1. Gökmenler, S., "MOOD Database Design Tool", TÜBİTAK Software Research and Development Center, M.S. Thesis, Dept. of Computer Eng., bu tez çalışması yazma aşamasındadır.

2. Pınar Köksal, Query Optimization Through Optimization Regions, M.S. Thesis, Dept. of Computer Eng., bu tez çalışması yazma aşamasındadır.

MOOD projesinin birlikte demo edildiği diğer sistemler:

Ayrıca MOOD, 23-26 Mayıs tarihleri arasında ABD'nin Minneapolis kentinde yapılmış bulunan "ACM SIGMOD International Conference on Management of Data" konferansına demo edilmiştir. Bu konferansta demo edilen sistemlerden bazıları şöyledir:

"METU Object-Oriented DBMS", Asuman Dogac, Budak Arpinar, Cetin Ozkan, Cem Evrendilek, Ilker Altintas, Ilker Durusoy, Mehmet Altinel, Tansel Okay and Yuksel Saygin (**Tübitak & Middle East Technical University**)

"EOS: an Extensible Object Store", Alexandros Biliris and Euthimios Panagos (**AT&T Bell Laboratories**)

"Quest: A Project on Database Mining", Rakesh Agrawal (**IBM Almaden Research Center**)

"Relaxed Transaction Processing", Munindar P. Singh, Christine Tomlinson and Darrell Woelk (**Microelectronics and Computer Technology Corporation**)

"The MEDUSA Project: Autonomous Data Management in a Shared Nothing Parallel Database Machine", George M. Bryan, Wayne E. Moore, B. J. Curry, K. Lodge and J. Geyer (**University of Western Sidney, Nepean**)

"A Language Based Multidatabase System", Wva Kuhn, Konrad Schwarz and Thomas èTschernko (**Technische Universitaet Wien**)

"Ptool: A Scalable Persistent Object Store", Robert L. Grossman (**University of Illinois at Chicago**)

"The ORES Temporal Database Management System", Babis Theodoulidis, Aziz Ait-Braham, George Andrianopoulos, Jayant Chaudhary, George Karvelis and Simon Sou (**UMIST**)

"The MYRIAD Federated Database Prototype", S-Y. Hwang, E-P. Lim, H-R. Yang, K.Mediretta, M. Ganesh, D. Clements, J. Stenoien and J. Srivastava (**University of Minnesota**)

"GENESYS: A System for Efficient Spatial Query Processing", Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider and Bernhard Seeger (**University of Munich**)

The MOOD User Manual

Edited by Asuman Dogac

Software Research and Development Center
Scientific and Technical Research Council of Turkiye
Middle East Technical University
06531, Ankara Turkiye

e-mail: asuman@vm.cc.metu.edu.tr

1. OVERVIEW

This document is the user manual for the MOOD (METU Object-Oriented DBMS) under development at the Software Research and Development Center of the Scientific and Technical Research Council of Turkiye which is established at Middle East Technical University, Ankara, Turkiye. This project is led by Prof. Dr. Asuman Dogac and the implementation team includes the following researchers: Cetin Ozkan, Budak Arpinar, Yuksel Saygin, Ilker Durusoy, Mehmet Altinel, Ilker Altintas, Cem Evrendilek, Erhan Pasa, Gokhan Ozhan and Tansel Okay.

This manual describes Version 1.0 of MOOD. MOOD runs on Sun workstations with SunOS 4.

2. ABSTRACT

MOOD (METU Object-Oriented DBMS) is developed on the Exodus Storage Manager (ESM) [ESM 92], and supports a SQL-like object-oriented query language (MOODSQL)[Ozk 93, Dog 94] and a graphical user interface, called MoodView [Arp 93], developed using Motif. MoodView supports both English and German. The system is coded in C++ on Sun Sparc 2 workstations and has a type system derived from C++, eliminating the impedance mismatch between MOOD and C++.

ESM provides the MOOD the following kernel functions :

- storage management
- concurrency control
- backup and recovery of data.

Additionally, the MOOD kernel provides the following functions :

- catalog management
- optimization and interpretation of SQL statements. During this interpretation, functions (which have been previously compiled with C++) within SQL statements, are dynamically linked and executed. The advantage of this approach is that by eliminating the interpretation of the functions the overall efficiency of the system is improved.

Each object is given a unique Object Identifier (OID) at object creation time by the ESM which is the disk start address of the object returned by the ESM. The object encapsulation properties are supported through the public and private declarations of C++. Objects are grouped in the abstraction level of a class, in other words, classes have extensions. Class inheritance mechanism of the MOOD is multiple inheritance. The name resolution is handled as in standard C++. Aggregate definitions are handled in the MOOD system by introducing type constructors (Set, List, Ref and Tuple). Aggregate classes can be constructed by recursive use of these type constructors.

The MOOD Catalog Manager uses three classes to store data definitions. These are *MoodsTypes*, *MoodsAttributes* and *MoodsFunctions* classes. The *MoodsTypes* class instances keep definitions of classes, indices and data types of the database. Basic data types (integer, float, string, etc.) and type constructors (i.e., set, list, ref, tuple) defined by the system, are instances of *MoodsTypes* class. Instances of *MoodsTypes* class may have pointers to *MoodsAttributes* class instances which stores the information about the attributes. The *MoodsFunctions* class instances keep track of the member function definitions of classes to support dynamic definition and linking of functions.

The query optimizer of the MOOD [Dur 93] is generated by using the Volcano Query Optimizer Generator. The Volcano Query Optimizer Generator [McK 93] provides for very fast and easy development of a query optimizer. The MOOD Optimizer uses database statistics obtained from the MOOD catalog in computing the selectivities and the costs for each optimization step. The set transformation and implementation rules is given in [Dur 93] and the set of MOOD Algebra operators is given in [Dog 94].

A graphical user interface, namely MoodView is implemented. MoodView provides the database programmer with tools and functionalities for every phase of OODBMS application development. Current version of MoodView allows a database user to design, browse, and modify database schema interactively. Furthermore, a database administration tool, a full screen text-editor, a SQL based query manager, and a graphical indexing tool for the spatial data, i.e., R Trees are also implemented.

References

[Dog 94] Dogac, A., Ozkan, C., Arpinar, B., Okay, T., Evrendilek, C., "METU Object-Oriented DBMS", in *Advances in Object-Oriented Database Systems*, Dogac, A., Ozsu, T., Biliris, A., Sellis, T., eds., Springer-Verlag 1994.

[Arp 93] Arpinar, I. B., Dogac, A., Evrendilek, C., "MoodView: An Advanced Graphical User Interface for OODBMSs", *SIGMOD Record*, Vol. 22, No. 4, December 1993.

[Ozk 93] Ozkan, C., Dogac, A., Evrendilek, C., Gesli, T., "Efficient Ordering of Path Traversals in Object-Oriented Query Optimization", in *Proc. of the Intl. Symp. on Computer and Information Sciences*, Istanbul, November 1993.

[Dur 93] Durusoy, I., Dogac, A., "Query Optimization in MOOD Using Volcano Extensible Query Optimizer Generator", TUBITAK Software R&D Center Tech. Rep. No. 18, June 1993.

[ESM 92] Using the Exodus Storage Manager V2.1.1, June 1992.

[McK 93] McKenna, W. J., "Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer Generator", PhD thesis, Department of Computer Science, University of Colorado, 1993.

3. MOOD Gaphical User Interface, MoodView

3.1. Introduction to MoodView

MoodView provides the database programmer with tools and functionalities for every phase of object oriented database application development. Current version of MoodView allows a database user to design, browse, and modify database schema interactively and to display class inheritance hierarchy as a directed acyclic graph. MoodView can automatically generate graphical displays for complex and multimedia database objects which can be updated through the object browser. Furthermore, a database administration tool, a full screen text-editor, a SQL based query manager, and a graphical indexing tool for the spatial data, i.e., R Trees are also implemented.

3.2. MoodView Environment

In this section we present a typical MoodView session with the help of MoodView screens' snapshots.

3.2.1. Initial Window

Upon entering the programming environment, an initial window that contains the icons for each of the MoodView tools is displayed as shown in Figure 1(a).

Initial window contains a browser icon for both the class hierarchy browser and the object browser, a query icon for the query formulation tool and also an R Tree icon for the graphical indexing tool.

3.2.2. Database Design and Schema Updates

A database schema in MOOD contains class types, their methods and relationships between those classes. Their inheritance relationships is represented as a dag and MoodView uses a dag placement algorithm that minimizes crossovers and makes drawings for graph nodes.

3.2.3. Data Definition

MoodView can display a class hierarchy defined in MOODSQL Data Definition Language which is stored in MOOD Catalog. MoodView uses the catalog information maintained by the MOOD kernel and displays class hierarchy graphically.

Upon clicking on the browser icon in the initial window, MoodView displays the class hierarchy representing the database schema. All nodes representing the classes have standard menus activated by clicking on them. MoodView supports the primitive actions on the class hierarchy graph such as adding a new class, dropping an existing class, changing the name of a class etc. Figure 1 shows the inheritance graph for a sample database.

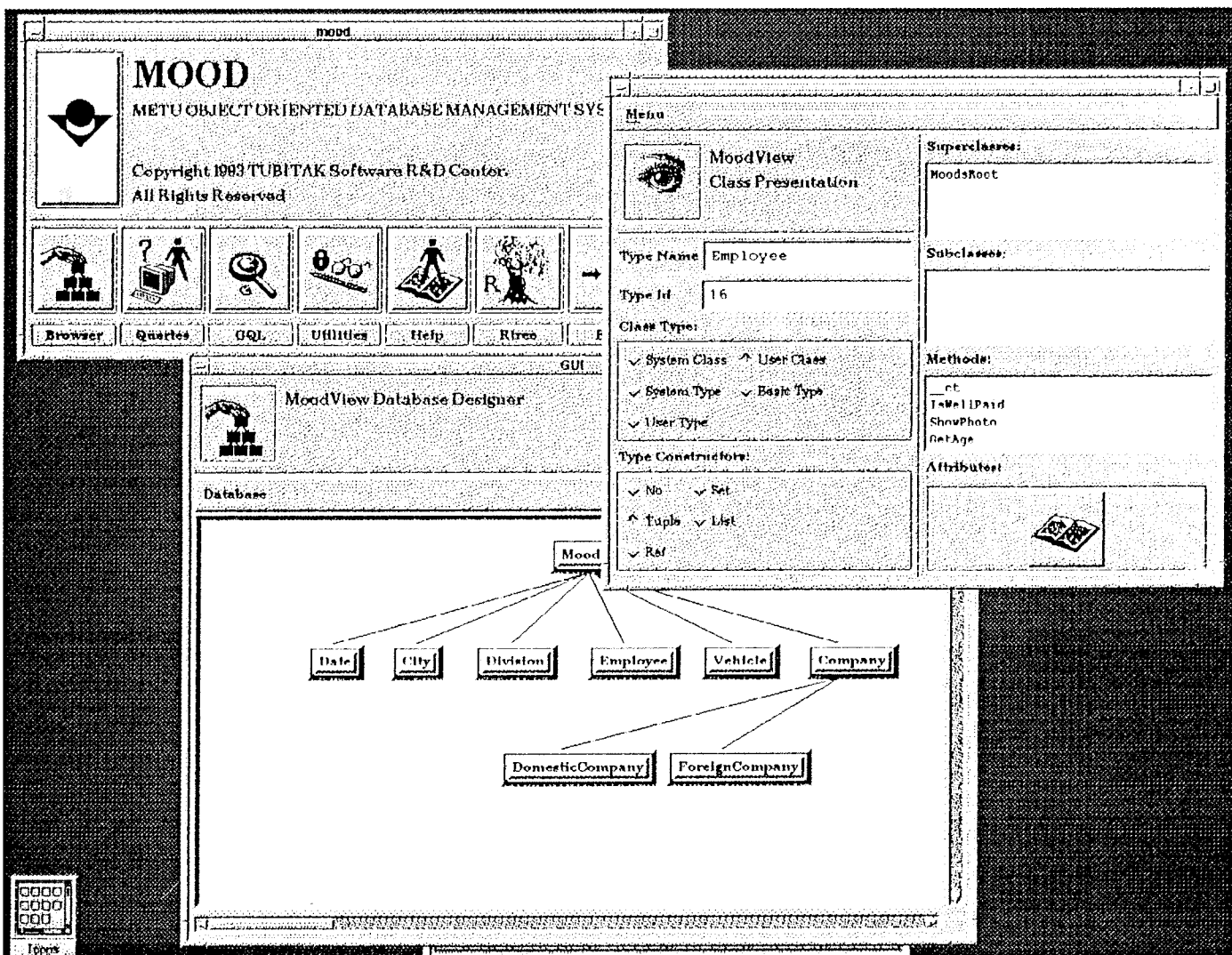
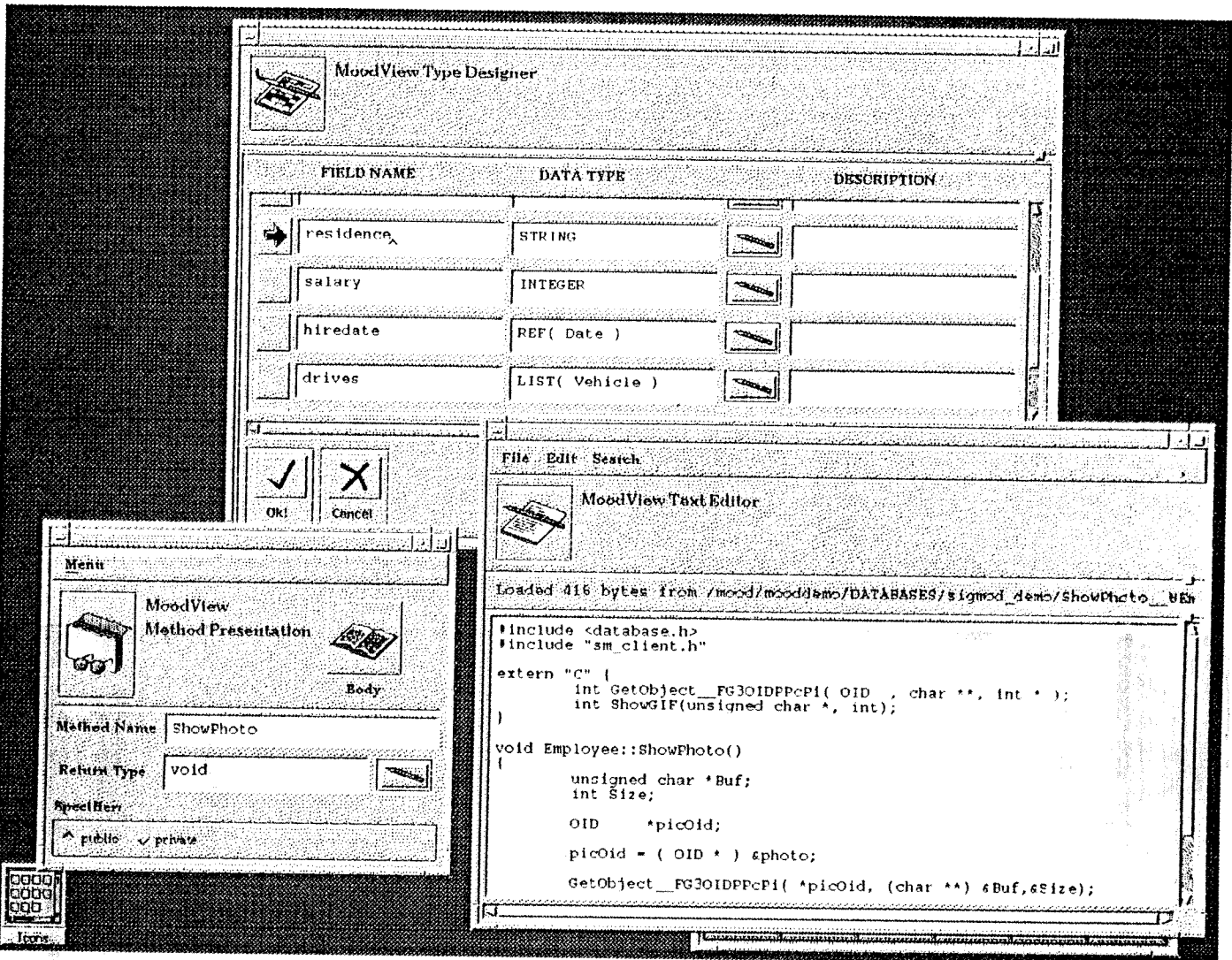


Figure 1. Initial MoodView Window, Class Hierarchy Browser, and Class Presentation



3.2.4. Class Presentation

Assume that database designer wants to extend his database to keep the information about every employee works in his office.

The database designer designs the Employee class as follows:

```
class Employee{
    ssno integer,
    name string,
    age integer,
    residence string,
    salary integer,
    hiredate Date,
    drives list(Vehicle),
    manager ref(Employee),
```

photo ref(Picture))

Figure 2 shows the attributes of the Employee class. Note that the hiredate, drives and manager fields have complex types. Photo is a reference to a system defined class Picture. The database designer can add this new class to his database by selecting "Add subclass" item from the standard menu of Moodsroot node in class hierarchy browser. This results in a pop-up template window which represents the new class. Each MoodView class presentation window contains a standard menu for schema updates, class type updates and class methods updates. MoodView class presentation shows a window that contains fields for the name of the class, its type id given by MOOD, its type constructor, superclasses, subclasses, and public and private methods and also a field that indicates if the class is a system or user defined class as shown in Figure 1.

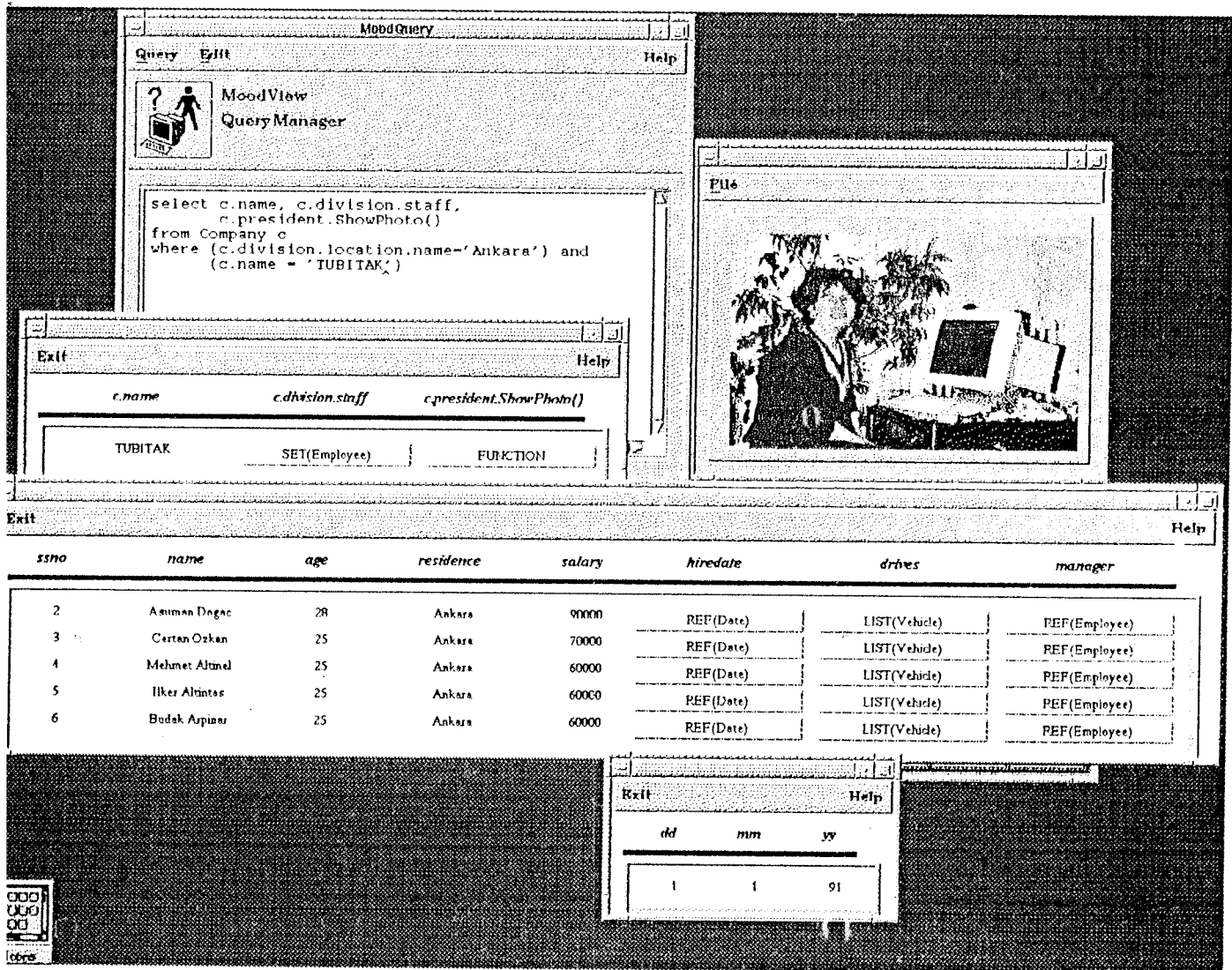


Figure 3. Query Manager and Objects displayed as a result of the Specific Query

3.2.5. Class Design

Class attributes can be updated by clicking on attributes button. This is an entry point for a tool for designing object-oriented data types. One can add, drop attributes, change the name or the type of an attribute by using this tool (Figure 2).

At any time during the database design process, user can cancel the operation using the Cancel button on the presentation or commit the transaction and add the newly created class to database schema by clicking on Save button in the menu. After saving, the new class appears immediately on the class hierarchy window.

3.3. Method Presentation and Class Methods Updates

Now suppose that the database designer wants to display the photograph of each employee in X/Windows environment. He can write a private method attached to the new class Employee for the picture display. Each class presentation of MoodView contains a menu for class methods' updates. Supported actions are adding a new method, dropping an existing method, and changing the body of a method. Class designer can call up the standard menu of the Employee class and select the "Add Method" item.

A method template is used for the new method creation as shown in Figure 2. Updates to existing method bodies or creation of the new method bodies can be done through MoodView text editor.

3.4. Object Browsing

A user can access database objects through the query manager of MoodView. MoodView allows complex operations against a set of objects. These include creation, deletion, update and automatic display of complex and multimedia objects, and the invocation of methods. Projection, selection and complex query specification can be done on the objects through the SQL based query manager. Updates to objects from query displays are not allowed in MoodView.

3.4.1 Generic Object Presentations

Any complex type in MOOD can be created by using basic types and recursive application of the type constructors (such as set, list or ref). Therefore MOOD objects constitute graphs connecting atoms and constructors and these graphs can be cyclic and large. MoodView has a generic display algorithm for displaying these object graphs and walking through the referenced objects. Referenced objects are represented as the drawn buttons and items of set and list are displayed in a scrolled window. Multimedia data such as images in different formats and sound are defined through the system classes. As an example, the Company presentation shown in Figure 3 contains references to a set of Employee objects.

3.4.2. Interactive Method Activation

Methods are attached to object presentations and can be activated interactively. For example, the user can display photograph of an Employee by clicking on function button representing ShowPhoto method.

3.5. Query Formulation

In MOOD, we have a uniform SQL-based interface in accessing the database. Query manager provides a query editor with facilities for accessing previous queries in a session. Through queries, objects with specific characteristics (selection) or selected portions of the objects (projection) can be displayed graphically.

MOOD Kernel interprets SQL statements and provides all the functions needed by MoodView to manage schema and instance levels as shown in Figure 4. During this interpretation, functions (which have been previously compiled with C++) within SQL statements, are dynamically linked and executed.

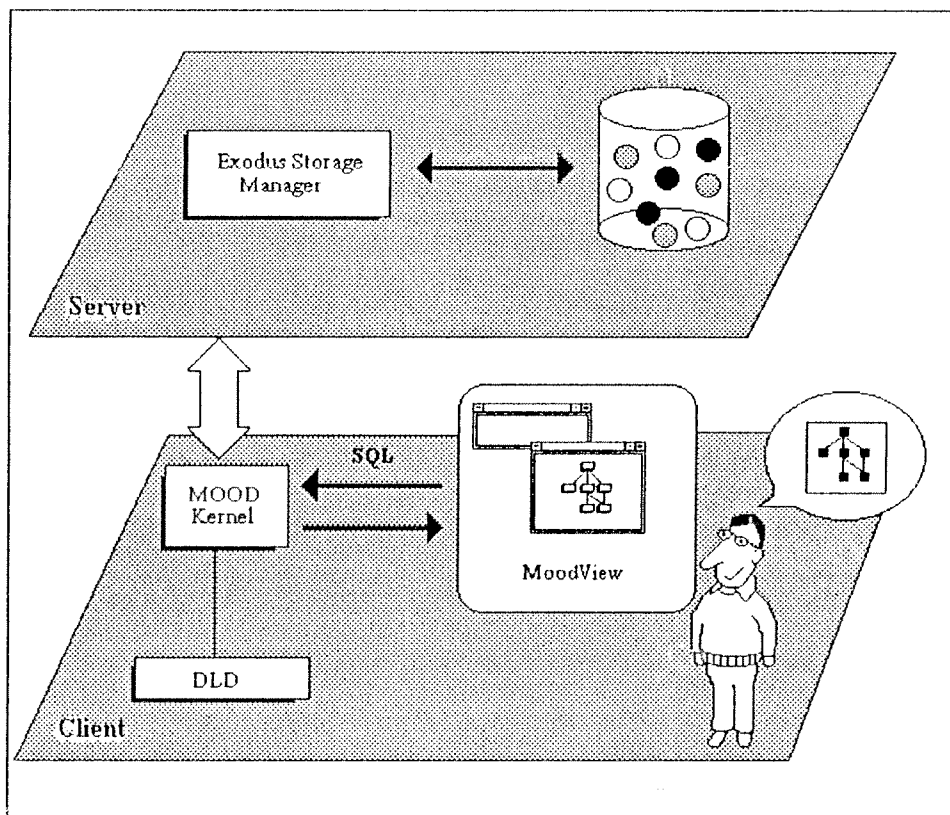


Figure 4. Overall MOOD Structure

4. MOOD Query Language, MOODSQL

Our reserved words are case insensitive.

4.1 Data Definition Language of MOODSQL

The following commands are used for creating, opening, or dropping a database :

```
CREATE DATABASE database-name
USE DATABASE database-name
DROP DATABASE database-name.
```

In order to create a class hierarchy for a database, BEGIN DECLARATION is needed to check the consistency of the definition, since although there are no cycles in the class hierarchy, there may be cycles in the class composition hierarchy. Therefore the common syntax for the declarations is as follows :

```
BEGIN DECLARATIONS {
    type or class declarations;
    ...
    type or class declarations;
}
```

In order to remove a class from the leaf of class hierarchy, the following command is used;

```
DROP CLASS class-name(s).
```

When a class is to be deleted all the classes that give reference to this class and all the classes that inherit from

this class must be deleted.

The indexing is handled by secondary B+-tree indices and hash indices supported through Exodus Storage Manager. The syntax of the command for creating a secondary B+-tree index on a set of atomic attributes of a class is as follows;

```
CREATE [UNIQUE] INDEX index-name
      ON class_name
      attribute [ASCDESC], ..., attribute [ASCDESC]
```

The syntax of the command for creating a hash index on a set of atomic attributes of a class is as follows;

```
CREATE [UNIQUE] HASH INDEX index-name
      ON class_name
```

INDEXES ARE NOT AVAILABLE IN V1.0.

In Figure 4.1 and 4.2 two example database schema definitions are provided which will be used in the example queries.

Begin Declaration {

```
create type Date
tuple(
    dd      integer,
    mm      integer,
    yy      integer
)
```

Create Class Vehicle

```
Tuple(
    id      integer,
    color   string[12],
    manufacturer ref( Company )
)
```

Create Class Company

```
Tuple(
    name    String[20],
    location String[20],
    president REF( Employee ),
    division SET( Division )
)
```

Create Class Employee

```
Tuple(
    ssno    Integer,
    name    String[32],
    age     Integer,
    residence String[20],
    salary  Integer,
    hiredate Date,
    drives  LIST( Vehicle ),
    manager REF( Employee )
)
```

Create Class Division

```
Tuple(
    functionality String[20],
)
```

```

    staff      SET( Employee ),
    location   REF( City )
)

Create Class City
Tuple(
    name       String[20],
    population  Integer
)

Create Class DomesticCompany inherits from Company
Tuple(
    ExportTax Integer
)

Create Class ForeignCompany inherits from Company
Tuple(
    ImportTax Integer
)

```

Figure 4.1. Example Schema Definition 1

```

Begin Declaration {

Create Class Course
Tuple(
    name       String[20],
    number     String[12],
    sections    LIST(Section),
    has_prereq  SET(Course),
    is_prereq_for SET(Course)
)

Create Class Section
Tuple(
    number     String[20],
    taught_by   Ref(Professor)
)

Create Class Employee
Tuple(
    id          Integer,
    name        String[32],
    age         Integer,
    salary       Integer
)

Create Class Professor inherits from Employee
Tuple(
    enum        String[20],
    rank        String[20] ,
    teaches     Set(Section)
)

Create Class Student
Tuple(

```

```

        name          String[20],
        student_id     String[10],
        dorm_address    Dorm,
        takes           Set(Section)
    )

```

Create Class TA inherits from Employee, Student

```

Tuple(
    assigned_to Ref(Professor)
)

```

Create Class Person

```

Tuple(
    name String[30],
    age Integer
)

```

Create Type Dorm

```

Tuple(
    college String[30],
    room_no String[20]
)

```

```

);

```

Figure 4.2. Example Schema Definition 2 (An Example database from ODMG-93 Definition)

4.1.1 Populating a Class with Instances

For Example Schema Definition 1:

```

new City('New York',15000000);
new City('Ankara',6000000);
new City('Istanbul',12000000);
new City('San Fransisco',10000000);
new City('London',10000000);
new City('Los Angeles',10000000);

New Employee(1,'Tansu Ciller',35, *new Date(1,1,91));
new Employee(2,'Asuman Dogac',28,*new Date(10,1,91));
new Employee(3,'Certan Ozkan',25, *new Date(1,1,91));
new Employee(4,'Mehmet Altinel',25,*new Date(1,1,91));
New Employee(5,'Ilker Altintas',25, *new Date(1,1,91));
New Employee(6,'Budak Arpinar',25,*new Date( 1,1,91));
New Employee(8,'Suha Sevuk',45, *new Date(1,1,91));
New Employee(9,'Tuncay Birand',45,*new Date(1,1,91));
New Employee(10,'Tosun Terzioglu',45, *new Date(1,1,91));
New Employee(11,'Ersin Tulunay', 45, *new Date(1,1,91));
New Employee(13,'Robert Redford',50, *new Date(1,1,91));
New Employee(14,'Michael Douglas',25, *new Date(1,1,91));
New Employee(15,'Sharon Stone',25, *new Date(1,1,91));
New Employee(16,'Daniel Day Lewis',30, *new Date(4,3,93));
New Employee(17,'Catherine Deneuve',35, *new Date(5,2,87));
New Employee(18,'Harrison Ford',25, *new Date(1,1,91));
New Employee(19,'Kevin Costner',25, *new Date(1,1,91));
New Employee(20,'John Wayne',25, *new Date(1,1,91));
New Employee(21,'Marylin Monroe',25, *new Date(1,1,91));
New Employee(22,'Al Pacino',25, *new Date(1,1,91));

```

```

New Employee(23,'Dustin Hoffman',25, *new Date(1,1,91));
New Employee(24,'Paul Newman',25, *new Date(1,1,91));
New Employee(25,'Cindy Crawford',25, *new Date(1,1,91));

New Division('top management',{select e from Employee e where
e.ssno=1},
    select d from City d where d.name='Ankara');
New Division('Software R&D Center',{ select e from Employee e
where (e.ssno>1) and (e.ssno<7) },
    select d from City d where d.name='Ankara');
New Division('management',{select e from Employee e where
e.ssno=4},
    select d from City d where d.name='Ankara');
New Division('development',{select e from Employee e where
e.ssno=3},
    select d from City d where d.name='Ankara');
New Division('coding',{select e from Employee e where e.ssno=6},
    select d from City d where d.name='Ankara');
New Division('star',{select e from Employee e where e.ssno>11},
    select d from City d where d.name='Los Angeles');

new Company('TUBITAK','Ankara',select e from Employee e where
e.ssno=10,
    {select d from Division d where d.functionality='Software
R&D Center'});
new Company('METU','Ankara',select e from Employee e where
e.ssno=8,
    {select d from Division d } );
new Company('MGM','Hollywood',select e from Employee e where
e.ssno=17,
    {select d from Division d where d.functionality='star' }
);
new Company('GM','New York',select e from Employee e where
e.ssno=24,
    {select d from Division d where d.location.name='Ankara' }
);

new DomesticCompany('TOFAS','Bursa',select e from Employee e
where e.ssno = 9,
    {select d from Division d where d.functionality =
'development' }, 10 );

new ForeignCompany('NISSAN','Tokyo',select e from Employee e
where e.ssno = 25,
    {select d from Division d where d.functionality =
'development' }, 50 );

New Vehicle(1,'blue',select c from Company c where c.name='GM');
New Vehicle(2,'yellow',select c from Company c where
c.name='GM');
New Vehicle(3,'green',select c from Company c where c.name='GM');
New Vehicle(4,'red',select c from Company c where c.name='GM');
New Vehicle(5,'black',select c from Company c where c.name='GM');

Create Function For Class Employee
With Prototype 'Employee( integer, string, integer, REF_Date)'
using './Employee.c';

```

Create Function For Class Vehicle
 With Prototype 'Vehicle(integer, string)'
 using './Vehicle.c';

Updating the database to include cyclic references

In MOOD V1.0 it is not possible to create cyclic data through the new statement, it is necessary to create cyclic references through the Update command.

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 1)
 where e.ssno = 1;

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 11)
 where e.ssno = 2;

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 2)
 where (e.ssno > 2) and (e.ssno < 8);

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 1)
 where e.ssno = 8;

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 8)
 where e.ssno = 9;

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 1)
 where e.ssno = 10;

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 10)
 where e.ssno = 11;

Update Employee e
 set e.manager = (select e from Employee e where e.ssno = 7)
 where (e.ssno > 11);

Update Employee e
 set e.drives = [select v from Vehicle v where (v.id < 3)]
 where e.ssno < 12;

Update Employee e
 set e.drives = [select v from Vehicle v where (v.id > 2)]
 where e.ssno > 12;

4.2. Data Manipulation Language of MOODSQL

General syntax of the query language is as follows;

```

SELECT projection-list
FROM class-name r1,
    class-name r2,
    ...
    class-name rn

```

[WHERE search-expression]

Operators used in MOODSQL are classified as;

1. Arithmetic Operators (+, *, -, /),
2. Boolean Operators (AND, OR, NOT) .
3. Comparison Operators (=, >, <, <>, >=, <=)

OR/NOT operators are not available in V1.0.

4.3 Example Queries in MOODSQL

The following examples use the example schema in Figure 4.1.

```
MoodSQL>select c.name, c.division.staff.name
>from Company c, Company o, Employee e
>where (c.division.staff.name=o.division.staff.name) and
>(o.name='METU') and (o.division.staff.name=e.name) and
>(e.age=45);
```

QUERY RESULTS

```
=====
METU Dustin Hoffman
MGM Dustin Hoffman
```

2 Objects are listed !

```
MoodSQL>select e.name, s.division.location.name
>from Company s, Employee e
>where (s.name = 'MGM') and (s.president = e) and (e.age > 20);
```

QUERY RESULTS

```
=====
Catherine Deneuve Los Angeles
```

1 Objects are listed !

```
MoodSQL>select c.president.name
>from Company c
>where c.president.drives.manufacturer=c;
```

QUERY RESULTS

```
=====
Paul Newman
Paul Newman
Paul Newman
```

3 Objects are listed !

```
MoodSQL>select v.manufacturer.division.staff.name
>from Vehicle v
>where (v.color = 'yellow') and (v.manufacturer.name = 'GM') and
> (v.manufacturer.division.staff.age = 28);
```

QUERY RESULTS

```
=====
Aşuman Dogac
```

1 Objects are listed !

```
MoodSQL>select c.name, c.division.location.name, o.name,  
o.division.location.name  
>from Company o, Company c  
>where (c.name='MGM') and  
(c.division.location=o.division.location);
```

QUERY RESULTS

=====

MGM Los Angeles METU Los Angeles
MGM Los Angeles MGM Los Angeles

2 Objects are listed !

```
MoodSQL>select v.manufacturer.name  
>from Vehicle v  
>where (v.color = 'red') and (v.manufacturer.president.drives.id  
= v.id);
```

QUERY RESULTS

=====

GM

1 Objects are listed !

```
MoodSQL>select v.manufacturer.name, c.president.name  
>from Vehicle v, Company c  
>where (v.color = 'red') and  
(v.manufacturer.president=c.president)  
>and (c.name='GM');
```

QUERY RESULTS

=====

GM Paul Newman

1 Objects are listed !

The following examples use the example schema in Figure 4.2.

```
SELECT x.age  
FROM Person x  
WHERE x.name='Pat'
```

```
SELECT x.name, y.name  
FROM Students x, x.takes.taught_by z  
WHERE z='Full Professor'
```

4.4 User Defined MOOD Functions and their Use in MOODSQL

Syntax :

```
CREATE [PUBLIC | PRIVATE] FUNCTION FOR CLASS className  
WITH PROTOTYPE 'prototype declaration'  
USING 'fileName';
```

```
MoodSQL>create function for class Employee  
>with prototype 'int UpdateSal()'   
>using './UpdateSal.c';
```



```

int Employee::UpdateSal() {
    float pinc;

    if (ssno == 1)
        pinc = 1.3;
    else if (ssno == 2)
        pinc = 1.2;
    else
        pinc = 1.1;

    return( (int) (salary * pinc) );
}

```

```

MoodSQL>select s.name, s.salary, s.UpdateSal()
>from Employee s
>where s.ssno < 10;

```

QUERY RESULTS

```

=====
Tansu Ciller 50000 64999
Asuman Dogac 50000 60000
Certan Ozkan 50000 55000
Mehmet Altinel 50000 55000
Ilker Altintas 50000 55000
Budak Arpinar 50000 55000
Suha Sevik 50000 55000
Tuncay Birand 50000 55000

```

8 Objects are listed !

```

MoodSQL>create function for class Division
>with prototype 'SET_Employee GetStaff()'
>using './GetStaff.c';

```

```

SET_Employee Division::GetStaff() {
    return staff;
}

```

```

MoodSQL>select c.division.GetStaff().name
>from Company c
>where (c.president.age = 45) and (c.president.name = 'Tosun
Terzioglu');

```

QUERY RESULTS

```

=====
Asuman Dogac
Certan Ozkan
Mehmet Altinel
Ilker Altintas
Budak Arpinar

```

5 Objects are listed !

```

MoodSQL>create function for class Employee
>with prototype 'boolean IsWellPaid(integer)'
>using './IsWellPaid.c';

```

```

integer Employee::IsWellPaid(integer Criterion) {

```

```

if (salary > Criterion)
    return 1;
return 0;
}

```

```

MoodSQL>select e.name, e.salary
>from Employee e
>where e.IsWellPaid(80000)=FALSE;

```

QUERY RESULTS

```

=====
Certan Ozkan 70000
Mehmet Altinel 60000
Ilker Altintas 60000
Budak Arpinar 60000
John Wayne 80000

```

5 Objects are listed !

4.4.1 Deleting User Defined Functions

```

MoodSQL>Is fnc of Employee;

```

FUNCTIONS

```

-----
__ct__8EmployeeFiPcT18REF_Date
Well_Paid_Emp__8EmployeeFi

```

```

MoodSQL>drop fnc with signature 'Well_Paid_Emp__8EmployeeFi'
>from class Employee;

```

```

MoodSQL>Is fnc of Employee;

```

FUNCTIONS

```

-----
__ct__8EmployeeFiPcT18REF_Date

```

4.5 Update in MOODSQL

```

MoodSQL>update Company c
>set c.president=(select e from Employee e
>where (e.drives.color='red') and (e.ssno=2))
>where c.name='METU';

```

```

MoodSQL>select c.president.name
>from Company c
>where c.name='METU';

```

QUERY RESULTS

```

=====
Asuman Dogac

```

1 Objects are listed !

```

MoodSQL>delete c from Company c
>where ( c.division.staff.salary>100000) and (c.division.staff.ssno=14);

```

Object deleted successfully...

MoodSQL>select c.name from Company c;

QUERY RESULTS

=====

TUBITAK

GM

2 Objects are listed !

4.6 Querying the Catalog

MOODSQL can be used to query the Catalog. Additionally following command are provided:

MoodSQL>list database;

LIST OF DATABASES

sigmod_demo

asuman_demo

certain

yuksel_demo

ilker

MoodSQL>use db sigmod_demo;

Database in use

MoodSQL>describe Employee;

User Class Employee

Inherits from MoodsRoot

TUPLE (

ssno INTEGER ,

name STRING[32] ,

age INTEGER ,

residence STRING[20] ,

salary INTEGER ,

hiredate REF(Date),

drives LIST(Vehicle),

manager REF(Employee),

photo OID ,

)

MoodSQL>Describe DomesticCompany;

User Class DomesticCompany

Inherits from Company

TUPLE (

ExportTax INTEGER ,

)

MoodSQL>list function of Employee;

FUNCTIONS

__ct__8EmployeeFiPcN218REF_DateT2

IsWellPaid__8EmployeeFi

ShowPhoto__8EmployeeFv

GetAge__8EmployeeFi

MoodSQL>list type;

TYPE NAME SIZE ID TYPE TYPE!

TYPE	NAME	SIZE	ID	TYPE	TYPE!
INTEGER		4	1	BASIC TYPE	
FLOAT		4	2	BASIC TYPE	
BOOLEAN		1	3	BASIC TYPE	
CHAR		1	4	BASIC TYPE	
STRING		1	5	BASIC TYPE	
OID		12	6	BASIC TYPE	
IID		8	7	BASIC TYPE	
FID		12	8	BASIC TYPE	
TUPLE		0	9	SYSTEM TYPE	
SET		12	10	SYSTEM TYPE	
LIST		12	11	SYSTEM TYPE	
REF		12	12	SYSTEM TYPE	
MoodsRoot		4	14	USER CLASS	
Vehicle		32	15	USER CLASS	
Employee		116	16	USER CLASS	
Date		16	18	USER TYPE	
City		28	20	USER CLASS	
Division		48	24	USER CLASS	
Company		68	28	USER CLASS	
DomesticCompany		72	30	USER CLASS	
ForeignCompany		72	32	USER CLASS	

5. Installation

```
/******  
* MOOD ( Metu Object-Oriented Database Management System )  
* Copyright (c) 1994 Scientific and Technical Research Council of Turkiye  
* Software Research and Development Center, Ankara  
* All Rights Reserved.  
*  
*Permission to use, copy, modify and distribute this software and its documentation is hereby granted, provided  
*that both the copyright notice and this permission notice appear in all copies of the software, derivative works  
*or modified versions, and any portions thereof, and that both notices appear in supporting documentation.  
*  
*THE SOFTWARE RESEARCH AND DEVELOPMENT CENTER ALLOWS FREE USE OF THIS  
*SOFTWARE IN ITS "AS IS" CONDITION. THE CENTER DISCLAIMS ANY LIABILITY OF ANY KIND  
*FOR ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.  
*  
* The MOODS Project Group requests users of this software to return any improvements or extensions that they  
*make to:  
*  
* MOODS Project Group  
* TUBITAK Software Research and Development Center  
* Department of Computer Engineering  
* Middle East Technical University  
* Ankara, 06531  
*  
* asuman@srcd.metu.edu.tr  
*  
*In addition, the MOODS Project Group requests that users grant the Software Research and Development  
*Center rights to redistribute these changes.  
*****
```

About the MOOD V1.0

This version of MOOD (V1.0) is a research prototype of an Object_Oriented DBMS. MOOD (METU Object-Oriented DBMS) is developed on the Exodus Storage Manager (ESM) and supports a SQL-like object-oriented query language (MOODSQL) and a graphical user interface, called MoodView developed using Motif. The system is coded in C++ on Sun Sparc 2 workstations and has a type system derived from C++, eliminating the impedance mismatch between MOOD and C++.

ESM provides the MOOD the following kernel functions :

- storage management
- concurrency control
- backup and recovery of data.

Additionally, the MOOD kernel provides the following functions :

- catalog management
- optimization and interpretation of SQL statements. During this interpretation, functions (which have been previously compiled with MOODCC) within SQL statements are dynamically linked and executed. The advantage of this approach is that by eliminating the interpretation of the functions the overall efficiency of the system is improved.

Each object is given a unique Object Identifier (OID) at object creation time by the ESM which is the disk address of the object returned by the ESM. The object encapsulation properties are supported through the public and private declarations of C++. Objects are grouped in the abstraction level of a class, in other words, classes have extensions. Class inheritance mechanism of the MOOD is multiple inheritance.

Aggregate definitions are handled in the MOOD system by introducing type constructors (Set, List, Ref and Tuple). Aggregate classes can be constructed by recursive use of these type constructors but there are some restrictions.

The MOOD Catalog Manager uses three classes to store data definitions. These are MoodsTypes, MoodsAttributes and MoodsFunctions classes. The MoodsTypes class instances keep definitions of classes, indices and data types of the database. Basic data types (integer, float, string, etc.) and type constructors (i.e., set, list, ref, tuple) defined by the system, are instances of MoodsTypes class. Instances of MoodsTypes class may have pointers to MoodsAttributes class instances which stores the information about the attributes. The MoodsFunctions class instances keep track of necessary information about functions and function definitions of classes to support dynamic definition and linking of functions.

The query optimizer of the MOOD is generated by using the Volcano Query Optimizer Generator. The Volcano Query Optimizer Generator provides for very fast and easy development of a query optimizer.

A graphical user interface, namely MoodView is implemented. MoodView provides the database programmer with tools and functionalities for every phase of OODBMS application development. Current version of MoodView allows a database user to design, browse, and modify database schema interactively. Furthermore, a database administration tool, a full screen text-editor, and a SQL based query manager is provided.

What MOOD V1.0 offers?

1. Data Definition Language

- a . creation and deletion of databases
- b . creation and deletion of classes and types
- c . insertion and deletion of member functions(including constructors)

2.The system has two interfaces:

- a. Textual user interface (moodsql)
- b. Graphical user interface (moodview)

The external software used in the MOOD V1.0 prototype

1. Exodus Storage Manager V3.0

2. Volcano Query Optimizer Generator
3. XTech/Motif (for graphical user interface)

Available platforms

Sun Workstations/SUN-OS

Features not yet implemented

1. View Management
2. Security
3. Aggregation operations (SUM, MIN, ...)
4. NOT/OR in where predicate
5. Member function calls with parameters other than constants is not supported by the current optimizer.

Disk Requirements

27MB including the binaries.

Prerequisites

The Exodus Storage Manager uses System V shared memory and semaphores, so your kernel must have them installed. The kernel should support a shared memory segment of at least 4 mega-bytes. If you are not sure whether your system is configured with shared memory, please see your system administrator. If you try to run the Storage Manager server on a system that does not have shared memory, the storage manager will print a message about this failure to allocate shared memory, and it will tell you how much shared memory it was trying to acquire; then it will exit.

Installation

To use MOODV1.0 the guidelines are as follows:

0. GETTING THE SOFTWARE

The MOOD (Metu Object-Oriented DBMS) is available without charge by anonymous FTP from <ftp.srdc.metu.edu.tr>. All of the MOOD software is located in the /pub/mood directory. The MOOD sources, documentation, and demo script programs are available in: moodV1.0.tar.Z. Note that MOOD runs on Exodus Storage Manager which is available from <ftp.cs.wisc.edu> free of charge.

To eliminate the need to compile the MOOD, we provide libraries, binaries for SPARCstations/SUN-OS in moodV1.0.tar.Z. The instructions below describe how to un-tar the release and install the Storage Manager by building binaries from the libraries provided. We also include instructions for compiling the MOOD for those who wish to do so.

1. INSTALLATION OF MOOD

create a directory to place MOODV1.0 (we assume that this directory is "/mood" . If you install MOOD V1.0 to another directory, please make all the related changes in the following instructions)

```
cd /mood
uncompress moodV1.0.tar.Z
tar xvf moodV1.0.tar
```

The following directories are present in the moodV1.0.tar

bin - binary files of MOOD and Exodus Storage Manager

moodsql - textual interface of the MOOD
 moodview - graphical user interface of the MOOD
 VolOpt - optimizer of the MOOD
 MOODCC - precompiler of the MOOD
 moodsetup - script file to create necessary directories in the directory in which MOOD will be executed.
 browser & query - subprograms that are to be executed by MoodView

The following are ESM executables. Please refer to ESM documents for further information.

sm_server - Exodus Storage Manager server
 formatvol - To format volumes
 diskrw - to read write data from volumes
 shutserver- to shut down server

lib - library files of MOOD

libmoodsql.a - MOOD execution engine library
 libsm_client.a - ESM client library

doc - documents of MOOD

doc/esmdoc - documents of ESM

src - source files of MOOD

include - include files of MOOD

include/esminclude - include files of ESM

include/optinclude - include files of Volcano Optimizer Generator

examples-demo script files of moodsql

3. CONFIGURING ESM Server

3.1. Preparing Volumes

To run MOOD you should prepare three volumes, namely, log volume, data volume, and temporary volume. A volume can be a Unix file or a Unix raw disk partition. When a raw disk partition is used, data is transferred between the storage manager server's buffer pool and the disk by the disk process, bypassing the Unix file system's buffer pool. In the following we provide an example storage manager server configuration file. The actual name of this file should be .sm_config and it is advisable to put it in /mood/bin directory. Please keep in mind that you have to run /mood/bin/sm_server and /mood/bin/formatvol in the directory where this configuration file exists to start ESM server.

This example is provided in /mood/config/server_config.example file.

```

#set storage manager buffer pool size in pages
#(Please refer to ESM documents for further information)
server*bufpages: 400

#
# The server "sm_server" will run on this port. Notice that 8000
# matches the information in the client's mount option below.
#
server*portname: 8000

#set the place of the disk processor
#Notice that /mood is the installation directory!!!
server*diskproc: /mood/bin/diskrw

#
# Here we tell the server and formatvol about the log and data
# volumes.
# The "[sf]" prefix to the options makes them visible to

```

```

# servers and formatvol.
#
# The logformat option describes the log volume.
# The first part of the option is the name of the file containing
# the volume. Next is the volume ID, the number of cylinders
# in the volume, the number of tracks/per cylinder, the number
# of pages/track, and finally the size (in K) of log pages.
#
[st]*[rl].logformat: /dev/logvol : 2000 : 1: 1: 1000: 8

#
# The dataformat option describes the data volumes.
# It is identical to logformat except there is no log page size
# parameter. Note that colons or white space can be used to
# delimit the parts of the option value.
#
[st]*[rl].dataformat: /dev/datavol: 2001 1 1 1000

#
# The tempformat option describes the temporary volumes.
# It is identical to logformat except there is no log page size
# parameter. Note that colons or white space can be used to
# delimit the parts of the option value.
#
[st]*[rl].tempformat: /dev/tmpvol : 2002 1 1 1000

#
# This specifies that the server, "sm_server", uses volume 2000
# for a log.
#
# end of server configuration file
Run
server*logvolume: 2000

```

```
formatvol -vol 2000 -vol 2001 -vol 2002
```

For more information about volumes please refer to "USING STORAGE MANAGER SERVERS" section in doc/esmdoc/sm3doc.*.

4. TO RUN MOOD tools

To run MOOD tools, you may create another directory in which you will run the MOOD (optinal). We assume that this directory is "/moodworkspace"

Any user of MOOD has to define four environment variables :
for (csh)

```
setenv MOOD_HOME /moodworkspace
```

The MOOD_BIN environment is needed to find the place of MOOD executables, and necessary resource files for MoodView. However if you intend to use MOODSQL only, you do not need to define MOOD_BIN provided that it is placed in your path.

```
setenv MOOD_BIN /mood/bin
```

Two environment variables EVOLID, TMPVOLID to identify data volume and temporary volume respectively

```
setenv EVOLID 2001
setenv TMPVOLID 2002
```


After setting the above environment variables, please run moodsetup

```
$MOOD_BIN/moodsetup
```

It is a UNIX script file that creates three directories by using MOOD_HOME environment variable

```
$(MOOD_HOME)/DATABASES
$(MOOD_HOME)/tmp
$(MOOD_HOME)/config
```

\$(MOOD_HOME)/tmp directory is used for some temporary file operations by MOOD.

\$(MOOD_HOME)/DATABASES contains header files of created databases for method compilation.

\$(MOOD_HOME)/config should contain ".sm_config" file needed by MOOD as a ESM client. A sample ".sm_config" file (this file is presented in "/mood/config/client_config.example ") with respect to ".sm_config" file given for the server is as follows;

```
# --- EXAMPLE .sm_config FILE for mood users ---
# This is an example configuration file for SM 3.0
# The client has a 400 page buffer pool.
client*bufpages:      2200
#
# The client library finds out what volumes it can mount from the mount
# option. The first number in the option value is the volume ID. The
# second number is a port number for the server managing the volume.
# The address after "@" is the machine on which the server is running.
#
client*mount:         2001 8000@sariyer.srdc.metu.edu
client*mount:         2002 8000@sariyer.srdc.metu.edu
```

Please note that you have to change the server machine name!

When you are through with the above steps and after starting the ESM server (/mood/bin/sm_server), you can safely run one of the MOOD user interfaces

```
$MOODBIN/moodsql
```

```
$MOODBIN/moodview english (or german if you want to use MoodView in German)
```

Note that there is no on-line help. Therefore please refer to the user manual in the mood/doc directory.

To create example database

To create example database, a moodsql script is provided in "/mood/examples" directory. Please read README file in this directory.

Compilation of MOOD

For compilation, please make necessary changes in the files :

/mood/src/Makefile, /mood/src/Makefile.common (i.e., compiler, place of libraries) and /mood/bin/MOODCC.

to re-generate library for mood execution engine

```
cd /mood/src
make libmoodsql
```

To compile textual interface of MOOD

```
cd /mood/src
make moodsql
```

To compile graphical user interface of MOOD

```
cd /mood/src  
make moodview
```

To generate moodsql query optimizer

```
cd /mood/src/volsrc  
make optimizer
```

for whole compilation

```
cd /mood/src  
make all
```

APPENDIX

Screen snapshots in German.

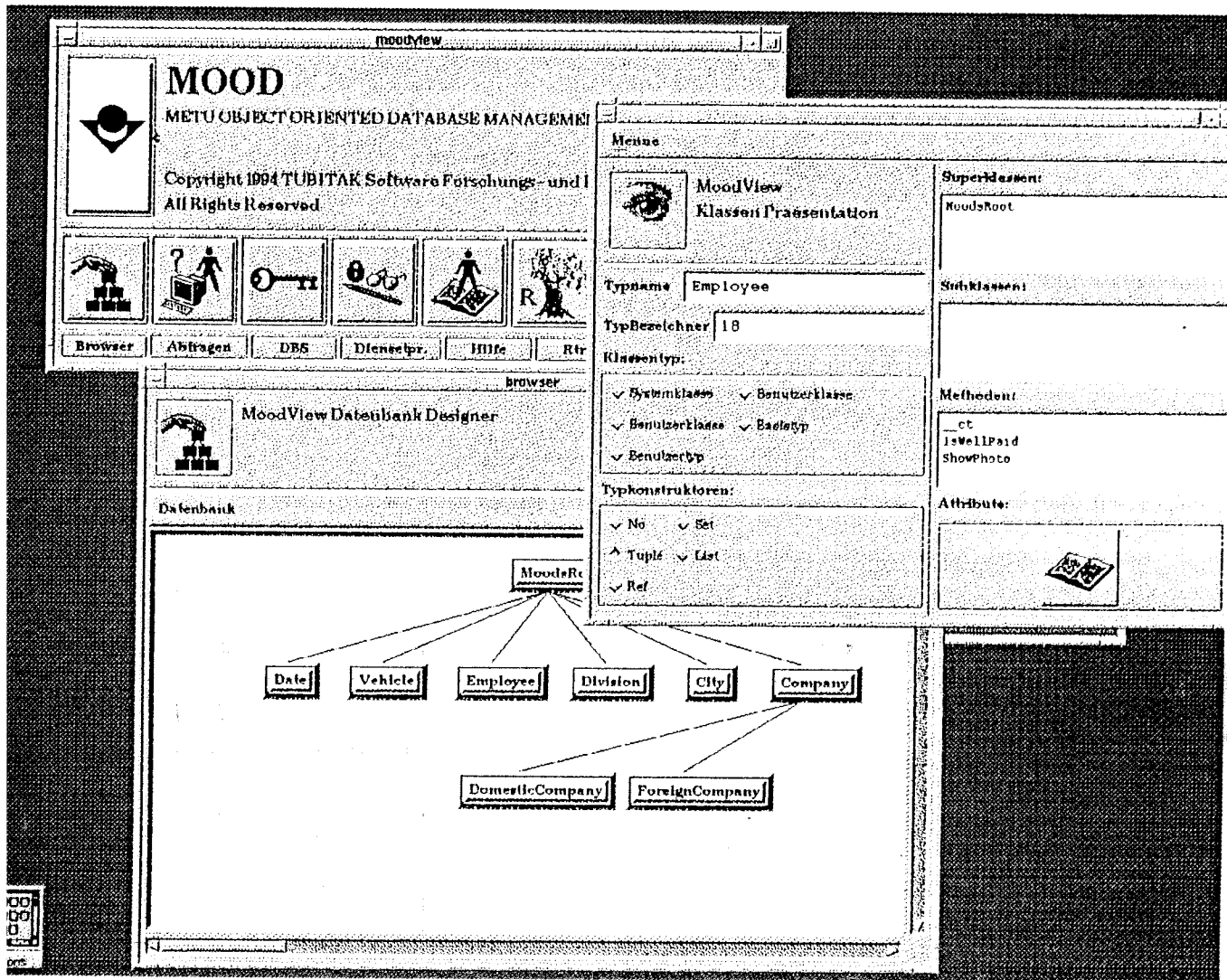


Figure 1. Initial MoodView Window, Class Hierarchy Browser, and Class Presentation

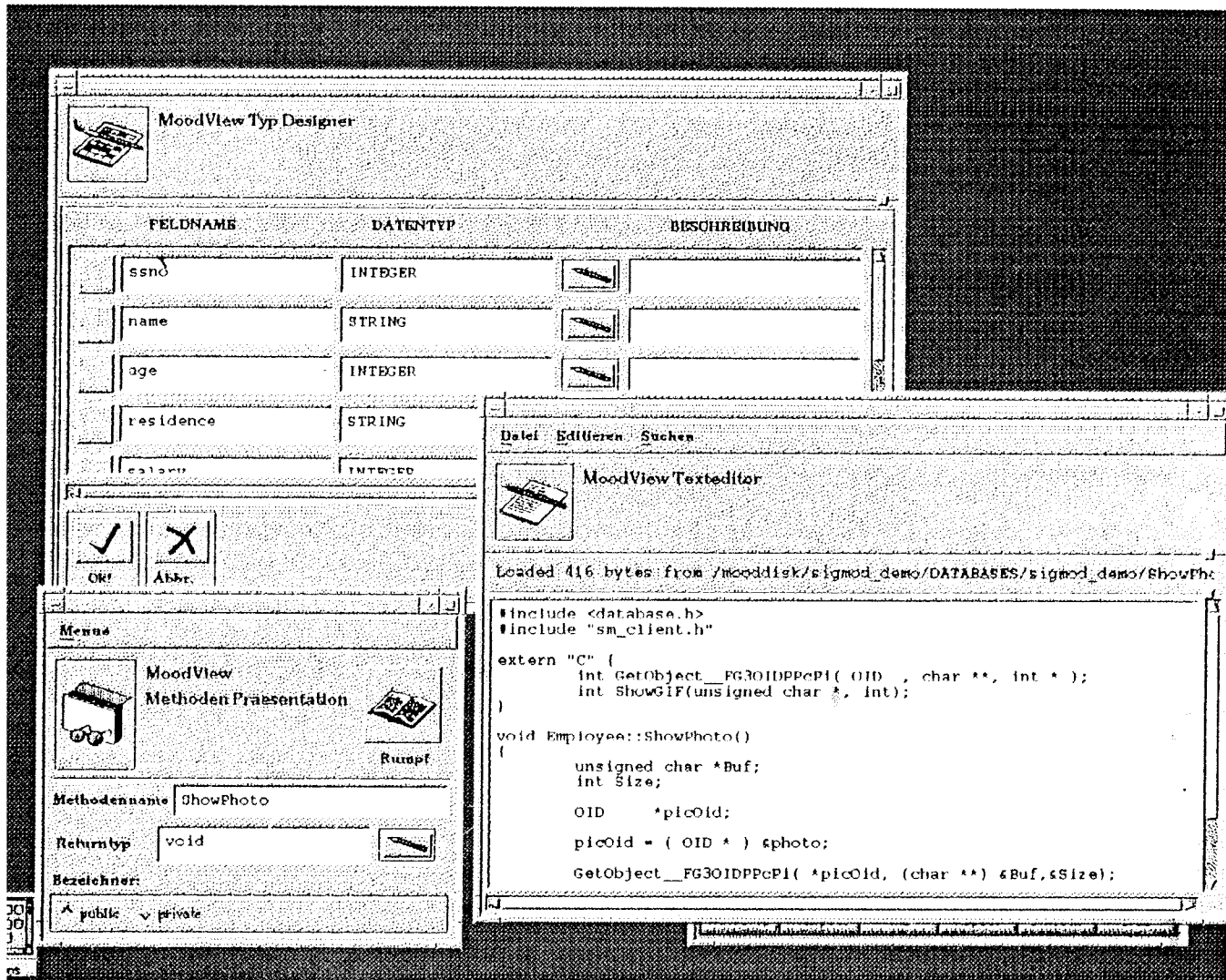



Figure 2. Type Designer, Method Presentation and Method body




MOOD

METU OBJECT ORIENTED DATABASE MANAGEMENT SYSTEM


Copyright 1994 TUBITAK Software Forschungs- und Entwicklungszentrum
All Rights Reserved

modiview



Query Edit

Mood Abfra



Exit Help

c.name c.division.staff c.president.ShowPhoto()

TUBITAK SET(Employee) FUNCTION

```

select c.name, c.division.staff,
       c.president.ShowPhoto()
from Company c
where (c.division.location.name = 'Ankara') and
       (c.name = 'TUBITAK')

```

Exit Help

ssm	name	age	residence	salary	hiredate	drives	manager
2	Arman Dogac	26	Ankara	90000	REF(Date)	LIST(Vehicle)	REF(Employee)
3	Certen Ozkan	25	Ankara	70000	REF(Date)	LIST(Vehicle)	REF(Employee)
4	Mehmet Altinel	25	Ankara	60000	REF(Date)	LIST(Vehicle)	REF(Employee)
5	Ilker Altintas	25	Ankara	60000	REF(Date)	LIST(Vehicle)	REF(Employee)
6	Budak Arpinar	25	Ankara	60000	REF(Date)	LIST(Vehicle)	REF(Employee)
7	Yukseel Saygin	25	Ankara	60000			REF(Employee)

Exit Help

dd	mm	yy
1	1	91

Figure 3. Query Manager and Objects displayed as a result of the Specific Query

METU Object-Oriented DBMS Kernel

Asuman Dogac Mehmet Altinel Cetin Ozkan
Ilker Durusoy

Software Research and Development Center
Scientific and Technical Research Council of Turkiye
Middle East Technical University (METU)
06531 Ankara Turkiye
email: asuman@vm.cc.metu.edu.tr

Abstract

This paper describes the design and implementation of a kernel for an OODBMS, namely the METU Object-Oriented DBMS (MOOD). MOOD is developed on the Exodus Storage Manager (ESM) and therefore some of the kernel functions like storage management, concurrency control, backup and recovery of data were readily available through ESM. Additional kernel functions provided are the optimization and interpretation of SQL statements, dynamic linking of functions, and catalog management. SQL statements are interpreted whereas functions (which have been previously compiled with C++) within SQL statements are dynamically linked and executed. Thus the interpretation of functions are avoided increasing the efficiency of the system. A query optimizer is implemented by using the Volcano Query Optimizer Generator. A graphical user interface, namely MoodView, is developed using Motif. MoodView displays both the schema information and the query results graphically. Additionally it is possible to update the database schema and to traverse the references in query results graphically.

Keywords: Object-oriented database systems, OODBMS kernel implementation, query optimization in OODBMSs, dynamic function linker, late binding

1 Introduction

In this paper we describe our experience in the design and implementation of the METU Object-Oriented DBMS (MOOD) kernel. The system is coded in GNU C++ on Sun Sparc 2 workstations. MOOD has a SQL-like object-oriented query language, namely MOODSQL [Ozk 93b, Dog 94c], and a graphical user interface, called MoodView [Arp 93a, Arp 93b] developed using Motif. MOOD has a type system derived from C++, eliminating the impedance mismatch between MOOD and C++. The users can also access the MOOD Kernel from their application programs written in C++. For this purpose MOOD Kernel defines a class named UserRequest that contains a method for the execution of MOODSQL statements. The MOOD system is available through ftp from ftp.cs.wisc.edu.

MOOD is developed on top of the Exodus Storage Manager (ESM) which has a client-server architecture [ESM 92, Car 86] and each MOOD process is a client application in ESM. ESM provides the MOOD the following kernel functions:

- storage management
- concurrency control
- backup and recovery of data.

Additionally, the MOOD kernel provides the following functions :

- catalog management
- optimization and interpretation of SQL statements. During this interpretation, functions (which have been previously compiled with C++) within SQL statements, are dynamically linked and executed. This late binding facility is essential since database environments enforce run-time modification of schema and objects. With our approach, the interpretation of functions are avoided increasing the efficiency of the system.

An alternative way to handle dynamic linking is to extend a C++ interpreter with DBMS functionality. In this alternative there is a problem of performance decrease due to interpretation. The advantage is to be able to use the full power of C++.

The implementation of a query optimizer for a SQL-like object-oriented query language, namely MOODSQL, is presented. The Volcano Query Optimizer Generator is used in developing the MOOD optimizer.

The paper is organized as follows: An overview of the MOOD Kernel is described in Section 2. In Section 3, the implementation of the Catalog Manager is given. The Query Manager and Dynamic Function Linker are discussed in Sections 4 and 5 respectively. Section 6 describes the MOOD Query optimizer. In Section 7, the implementation of the Database Engine is summarized. Kernel interaction with MoodView is given in Section 8. Finally conclusions are presented in Section 9.

2 MOOD Kernel Design Considerations and an Overview

There are some design choices that we have made and some decisions are enforced by the implementation language, namely C++. These are discussed in the following within the framework presented in [Mat 93].

Object Specification in MOOD: Each object is given a unique Object Identifier (OID) at object creation time by the ESM which is the disk start address of the object returned by the ESM. Object encapsulation is considered in two parts, method encapsulation and attribute encapsulation. These encapsulation properties are similar to the public and private declarations of C++.

Declarative properties of the MOOD : In the MOOD, the aspect concept given in [Mat 93] is limited to type specifications, in other words, class definitions are analogous to type definitions in a programming language. Hence each instance in the system is defined as a member of a class.

Procedural properties of the MOOD: Methods can be defined in C++ by users to manipulate user defined classes. Method binding is performed dynamically at run time. Dynamic linking primitives are implemented by the use of the shared object facility of SunOS [Sun 90]. Overloading is realized by making use of the signature concept of C++.

Object Abstractions in the MOOD: Objects are grouped in the abstraction level of a class, in other words, classes have extensions. Class extensions are implemented as ESM files. A class in the system has an unique

type identifier which is inherited from a meta class named MoodsRoot. This type identifier is used in accessing the catalog to obtain the type information to be used in interpreting the ESM storage objects which are untyped arrays of bytes. The relation between classes and instances is a 1:n relation, i.e., under a class there could be any number of instances associated with it, but an instance can not be associated with more than one class. Class inheritance mechanism of the MOOD Kernel is multiple inheritance. The name resolution is handled as in standard C++. Additionally in our system in case of name conflicts, if the scope resolution operator is not used, the first class in the inheritance order having that attribute is assumed as default.

Aggregates in the MOOD: Aggregate definitions are handled in the MOOD system by introducing type constructors (Set, List, Ref and Tuple). Aggregate classes can be constructed by recursive use of these type constructors.

2.1 MOOD Overview

The general flow of execution in the MOOD system is shown in the Figure 1. MoodView [Arp 93a, Arp 93b] is the graphical user interface of the system. MoodView displays both the schema information and the query results graphically. Additionally it is possible to update the database schema and to traverse the references in query results graphically. In displaying the schema, MoodView either makes direct calls to the Catalog Manager (1) or issues the necessary commands to the Query Manager (3). For primitive operations involving a single function call, MoodView directly communicates with the catalog. Complex operations are passed to the Query Manager. Query Manager parses and executes these commands by obtaining the necessary information from the Catalog Manager (5,6). Results are returned from the Catalog Manager or from the Query Manager depending on which subsystem received the request (2,4). Query Manager handles the method creation through Dynamic Function Linker (8, 9).

MoodView passes the MOODSQL [Dog 94c, Ozk 93a, Ozk 93b] queries to the Query Manager without any modification (3). Query Manager obtains the necessary information from the Catalog Manager (5,6) and then makes syntax and semantic checks on the queries. If no error is detected, a query tree is generated and passed to the query optimizer (7). After the optimization phase, resulting query tree is ready to be executed by the MOOD engine (10).

During execution, class extents are read from the Exodus Storage Manager (ESM) and temporary results are stored in the ESM (16, 17). The query tree is executed in the engine starting from the leaf level. If the class methods are used in the query, they are activated through the Dynamic Function Linker subsystem (14,15). At the end of the execution, results are returned directly to MoodView (13). In Moodview, a user can traverse the links in the database or execute class methods with void return type by using the cursor mechanism provided.

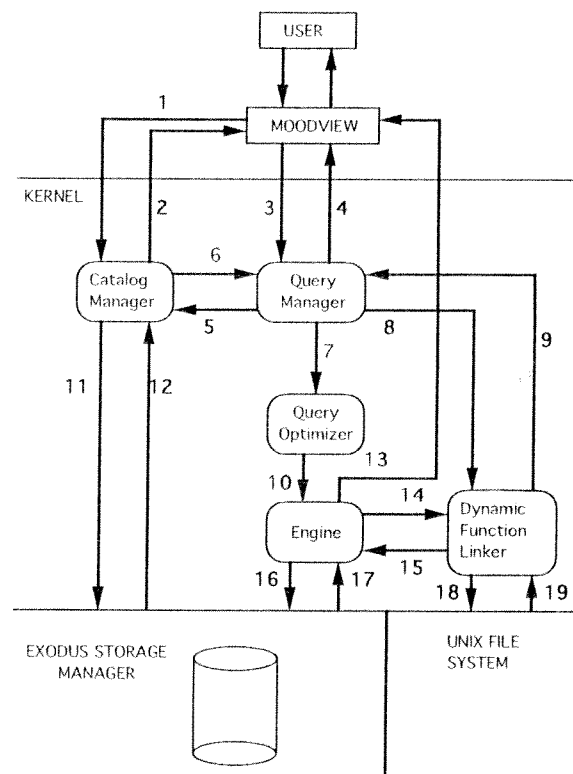


Figure 1. An Overview of the MOOD System

MOOD also has a textual interface to the database. For this interface, MOODSQL provides schema definition and modification commands.

3 Query Manager

Query Manager (QM) is the main subsystem of the kernel which accepts the MOODSQL commands through the MoodView or directly from the textual interface. During the interpretation of these commands, QM communicates with the other kernel subsystems. Functionality of MOODSQL commands can be divided into two parts: schema definition and modification commands, and data manipulation commands. For schema commands, QM interacts with the Catalog Manager and in case of method definitions it also interacts with the Dynamic Function Linker. QM exports the MOOD data types and classes to the C++ compiler of the MOOD, namely MOODCC, by preparing and updating the C++ header files when a new database or a new class is created. Catalog Manager serves as the information storage of the Query Manager. For the method definitions, QM activates the Dynamic Function Linker to construct the shared object of the method and to get information about the method. For the MOODSQL queries, QM makes the syntax and semantic checks on the query using the information obtained from the Catalog Manager. It then constructs an input query tree for the optimizer by filling in the nodes with the information necessary for the optimizer and for the database engine.

A detailed description of the MOOD data model and MOODSQL is given in [Dog 94c, Ozk 93b]. In this section, brief overviews of the data model and the query language are presented. In the MOOD data model the basic data types are Integer, Float, LongInteger, String, Char, and Boolean. Any complex data type is defined using these types and by recursive application of the Tuple, Set, List and Ref type constructors. The data model also supports multiple inheritance and strongly typed methods.

The syntax of the SELECT-FROM-WHERE block is as follows:

```
SELECT projection-list
FROM   path-name  $r_1$ ,
       path-name  $r_2$ ,
       ...
       path-name  $r_n$ 
WHERE search-expression
```

Projection-list and search-expression may include path expressions and

methods.

MOOD allows dynamic schema modifications and MOODSQL queries return objects. By using these two features a user can store the new objects obtained as the result of a query in the database. Notice that it is user's responsibility to place the new class in the inheritance hierarchy. The following example illustrates a class definition in MOODSQL.

```
CREATE CLASS Company
TUPLE(
    name          STRING[20],
    location       STRING[20],
    AnnualIncome  INTEGER,
    president      REF( Employee ),
    division       SET( Division ) )
```

An example MOODSQL query that finds the president who drives a car manufactured by his company and the company has a division at Los Angeles, is given in the following.

```
SELECT c.president.name
FROM   Company c
WHERE  (c.president.drives.manufacturer = c) AND
      (c.division.location.name = 'Los Angeles')
```

The existing objects returned by MOODSQL queries can be used in populating the REF, SET and LIST attributes in UPDATE and NEW commands as illustrated in the following.

```
UPDATE Company c
SET      c.president= (SELECT e
                        FROM   Employee e
                        WHERE  (e.drives.color='red') and (e.ssno=2))
WHERE   c.name='METU'

NEW Company('GM','New York',9000000,SELECT e
                                           FROM Employee e
                                           WHERE e.ssno=24,
                                           { SELECT d
                                             FROM Division d
                                             WHERE d.location.name='Ankara' } )
```

An example to DELETE command is as follows:

```
DELETE c
FROM   Company c
WHERE  ( c.president.salary<c.division.staff.salary)
```

Notice that an object is deleted physically when its reference count reaches to zero and garbage collection is handled by the ESM.

Using methods in the queries is described in Section 5.

4 Catalog Manager

The MOOD catalog contains the definition of classes, types, and member functions in a structure similar to a compiler symbol table. The catalog is stored on the ESM. In order to achieve late binding at run time, it is necessary to carry compile time information to run time. This information is obtained from the Catalog Manager by the MoodView or Query Manager.

Catalog Manager uses three classes to store data definitions. These are *MoodsTypes*, *MoodsAttributes* and *MoodsFunctions* classes. The *MoodsTypes* class instances keep definitions of classes, indices and data types of the database. Basic data types (integer, float, string, etc.) and type constructors (i.e., set, list, ref, tuple) defined by the system, are instances of *MoodsTypes* class, which are defined externally while creating the database. It is clear that a user can define any type and method to the database. Instances of *MoodsTypes* class may have pointers to *MoodsAttributes* class instances which store the information about the attributes. The *MoodsFunctions* class instances keep track of the member function definitions of classes to support dynamic linking and execution of functions. Needed information to construct instances of *MoodsFunctions* is extracted from the source code of the methods through the MOODCC. Figure 2 shows the structure of the catalog on the ESM.

Catalog classes are not any different from user defined classes. Therefore, MOODSQL can be used in accessing the required information from the catalog.

This implementation approach provides dynamic schema updates which is one of the main design considerations of MOOD. However, since versioning facility has not been implemented yet, the objects in class extents whose

definitions have been modified, are deleted.

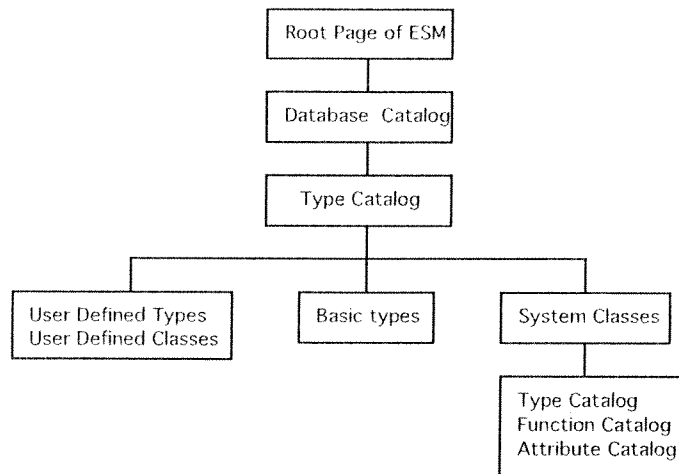


Figure 2. Structure of the catalog on ESM

5 Dynamic Function Linker

Dynamic Function Linker (DFL) provides the late binding of methods to the objects during the query interpretation. DFL is implemented using the shared object facility of SunOS. Methods are stored as shared objects [Sum 90] in a special directory hierarchy in the UNIX file system and they are mapped to the MOOD address space if they are not already there and executed during the interpretation of MOODSQL queries. In this respect, functionality of DFL can be divided into two parts: During the definition of methods, DFL constructs shared objects and provides the necessary information about methods to the Catalog Manager through the Query Manager. During the interpretation of MOODSQL queries, DFL locates and fetches the shared objects of the methods to the memory and finds the address of a method within a shared object.

In the definition phase of a method, DFL first constructs the shared object of the method. This step requires the compilation of the method body with the C++ compiler of MOOD, namely MOODCC. MOODCC is developed by modifying the cfront part of the AT&T C++ compiler (Release 2.0). The

function of cfront is illustrated in the Figure 3.

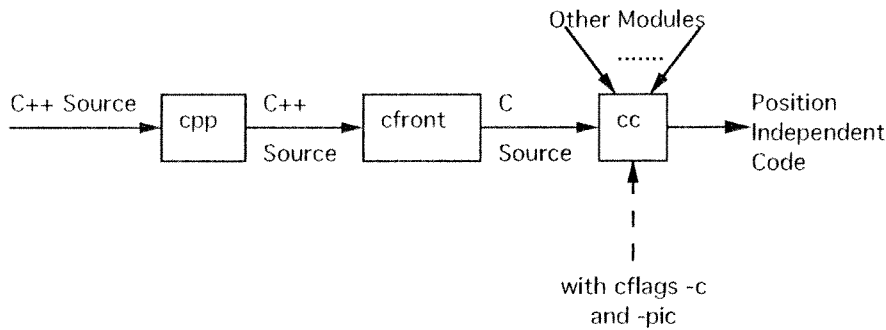


Figure 3. Execution flow of the AT&T C++ (MOODCC) Compiler

The purpose of this modification is to extract the necessary information about the method, to pass the parameter values and to get the result values from the method after its execution. As a result, users can code the method bodies without any restriction and method codes are modified transparently by the MOODCC. The modified cfront makes the following updates in the method:

- All the parameter types are converted to the pointer of the parameter type except for the parameters that are already pointers. This is necessary to eliminate the type dependency in parameter passing.
- Method return types are converted to the pointer of the return type again to eliminate the type dependency as in parameter passing. But there is yet another problem: If a function returns a constant, it is not possible to take the address of it. Therefore the following modification is also necessary. A global variable having the same type as the function return type is created and the function definition is modified so that each function has a new return type as the pointer to this variable. For each return statement, return value is copied to this variable and the address of this variable is returned instead of actual return value.

Since all the database functions are stored in different shared objects, their invocations have to be detected in the method source in order to provide dynamic linking and execution of the called method (This is also required for recursive function calls). Because the methods are not interpreted, to handle

this situation MOODCC pastes the necessary C code block to the output C code of the method.

Each method is stored in a distinct file and the exported MOOD data types and classes are accessed by the method by including a header file named "database.h" at the beginning of the code. In the following example, a method for the class Company is coded which calculates and returns the amount of tax the company will pay depending on a criterion given as a parameter to the method.

```
#include "database.h"
integer Company::Calculate_Tax(integer Criterion) {
float TaxRate;

    TaxRate = 0.1;
    if (AnnualIncome > Criterion)
        TaxRate = 0.2;
    return (integer) (TaxRate * AnnualIncome);
}
```

It is clear from the example that C++ types and MOOD types (e.g. integer) can be mixed in MOOD methods. Assuming that user saves this code in the file Tax.c, this method is declared to the MOOD by the following MOODSQL command in the textual interface:

```
CREATE FUNCTION FOR CLASS Company
WITH PROTOTYPE 'integer Calculate_Tax(integer)'
USING './Tax.c'
```

Upon receiving this MOODSQL command, Query Manager activates the DFL to obtain the shared object code of the method. In DFL Tax.c is compiled with MOODCC and if no errors are detected, object code of the method is prepared and stored in a special directory hierarchy. In addition to the shared object code, MOODCC returns the information about the method including its signature. This signature is prepared by using class name to which method belongs, the method name and the type of the parameters. A method is identified in MOOD by using its signature since signature makes the method name unique within the database. Signature concept also pro-

vides the overloading facility since each overloaded method has a different signature.

During the interpretation of MOODSQL queries that include methods, locating the shared object and finding the binding address of the method in the shared object are handled by the DFL using the signature of the method constructed by the Query Manager. After DFL performs these operations, the Database Engine executes the method for each instance and the return value is used in the expression.

MOOD also allows users to define methods returning no value (i.e. void return type). Such methods can be included in projection expressions. In this case instances are located in Database Engine but activation of the method is left to the user in the MoodView.

Users also have the chance of using the ESM calls in the methods which provides them to perform low level object operations in the methods.

A detailed description of the implementation of DFL is presented in [Alt 94].

6 The Query Optimizer

The MOOD Optimizer is implemented using the Volcano Extensible Optimizer Generator (VOG) [McK 93]. The Volcano Query Optimizer Generator is a data model independent tool that is used to develop a query optimizer for a DBMS. VOG search algorithm uses dynamic programming with branch-and-bound pruning based on cost. The Volcano generated optimizers produce the optimum execution plan when the transformation rules and support functions are provided properly.

The Volcano optimizer generator uses two algebras, called the logical and the physical algebras. The job of a generated optimizer is to map an expression of the logical algebra (a query) into an expression of the physical algebra (a query evaluation plan consisting of algorithms). To do so, it uses transformations within the logical algebra and cost-based mapping of logical operators to physical algebra [McK 93].

Since the paradigm of MOODSQL is the selection from extents of the classes, the traditional set and relation operators are accepted as the basis of the logical algebra. Selection, Join, Intersection, Projection and Union are defined as in the relational systems. An Unnest operator is used to manip-

ulate set-valued components. In addition, a materialize operator (MAT) is used as defined in [Bla 93] to represent each link of path expressions such as employee.department.manager.name. The purpose of MAT operator is to indicate to the optimizer where path expressions are used so that algebraic transformations can be applied. The GET_SET operator appears at the leaves of the query tree. It includes a class into the scope of the query. MOOD Physical Algebra operators are given in Section 7.

An example of the input and output of the optimizer is given in the following to clarify the point. Consider the following query :

```
SELECT e.name, v.type
FROM   Employee e, Vehicle v
WHERE  e.comp.mgr.age>45 AND e.parttime() AND
       v.id=12345 AND v.producer.dept.year< 60 AND
       e.comp.mgr.age = v.producer.dept.year+5
```

Query Manager produces a logical input tree that is given in Figure 4. The generated optimizer optimize this tree and produces the plan given in Figure 5. The produced plan is the optimal according to the given statistics. In this example, as it is observed from the input and output trees, the selections are pushed down the tree according to their selectivities. Materialize operators are converted into explicit joins and are implemented using the PTR_HHJOIN algorithm which is the pointer based version of the hash partition join [She 90]. The traversal of long path expressions are converted into a series of joins, so that the optimal execution plan is obtained according to the given statistics.

7 Database Engine

Database engine is the query execution subsystem of the MOOD. It consists of mainly four parts: expression interpreter generator, query interpreter, expression execution subsystem and object buffer as shown in Figure 6.

The expression interpreter implementation approach of the MOOD differs from the classical ones in that there is no general expression interpreter subsystem. For each operator that has an expression argument, a specific

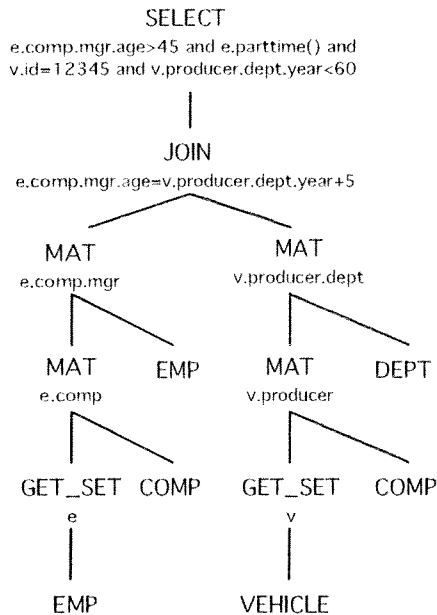


Figure 4. Input Query to the MOOD Optimizer

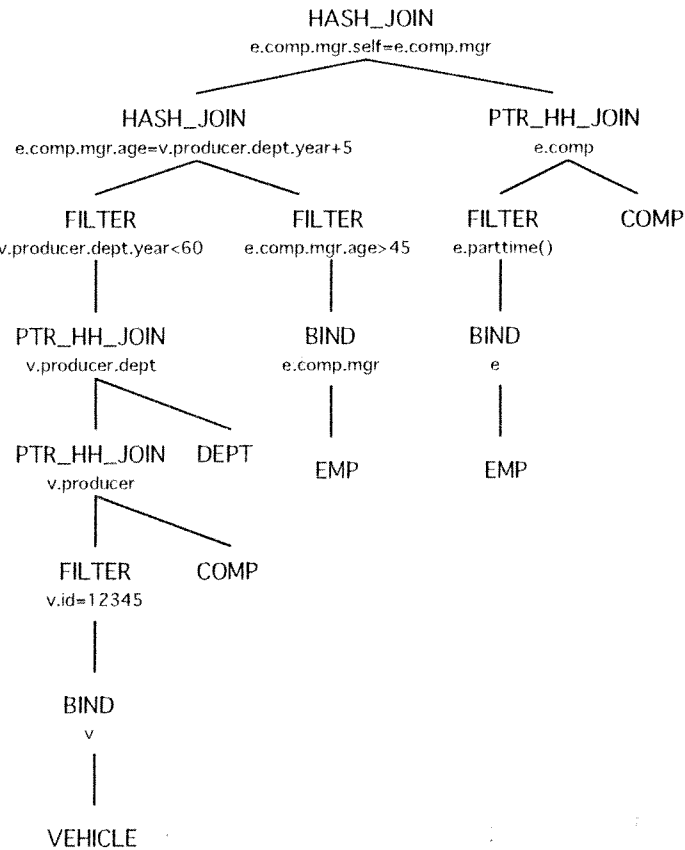


Figure 5. Optimal Access Plan Generated

interpreter is generated which is dedicated only to that input expression. This provides efficient execution of the expressions since dedicated interpreter does not have to deal with any type checking or operator selections. The Query interpreter first sends the input expression tree to the expression interpreter generator and then generated interpreter is executed in the expression execution subsystem for each object in the input extent(s). This implementation approach can be thought of as some kind of compilation of the input query tree and the dedicated interpreter generated can be thought of as the object code.

The dedicated interpreter is also generated in the form of a tree and its topology is exactly the same as the input query tree. What makes it different from the input query tree is that each node has an action, an input to that

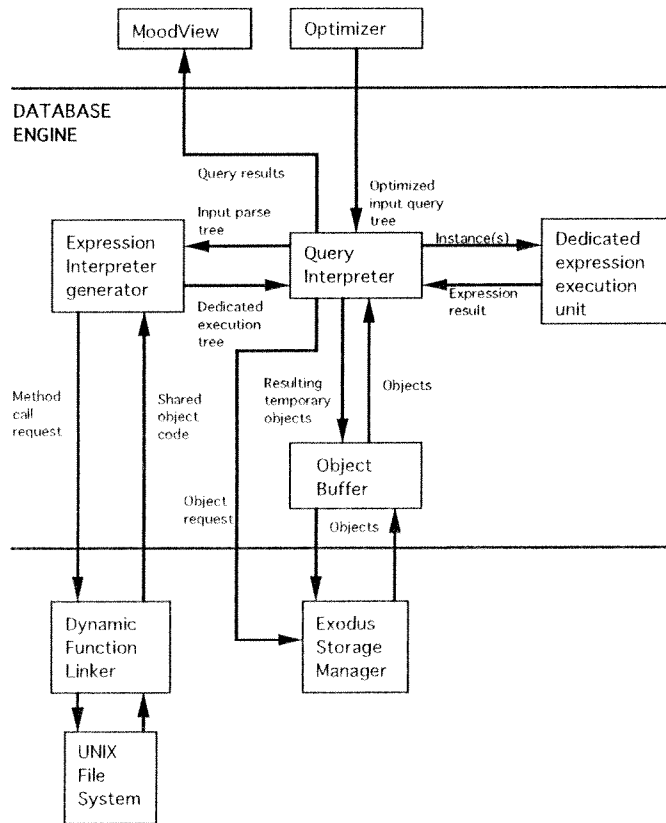


Figure 6. Overview of the Database Engine

action and a local buffer to hold the result of the action. An action is implemented by coding a unique function for it. In general, these functions take the action input as argument and after executing the code store the result in the node's local buffer. During the generation of the dedicated interpreter, depending upon the input elements, necessary actions and action inputs are binded to the nodes and the methods in the expressions are made ready to be executed by the DFL interaction.

Query interpreter is the main part of the database engine since it controls the execution flow. It takes the optimized query tree from the optimizer and interprets it starting at the leaf level. Before the execution has started, an object buffer is opened to store the intermediate results obtained during the execution. If the operator has an expression argument, query interpreter first generates the dedicated interpreter and this interpreter is executed in the operator through the dedicated interpreter execution subsystem. MOOD

physical algebra operators are implemented using ESM calls. Currently, the physical algebra operators implemented are as follows:

- Selection operator (FILTER)
- Indexed Selection (IND_SEL)
- Dereference operator (DEREF)
- Bind operator (BIND)
- Nested Loops Join (NESTED_LOOP_JOIN)
- Pointer Hybrid Hash Join (PRT_HH_JOIN)
- Sort-Merge Join (SM_JOIN)
- Hash Partition Join (HASH_JOIN)
- Sort operator (SORT)
- Union operator (UNION)
- Difference operator (DIF)
- Intersection operator (INT)
- Materialize operator (TRAVERSE)
- Unnest operator (UNNEST)
- Asset operator (ASSET)
- Aslist operator (ASLIST)

At the end of the query execution, the information about the extent in which resulting objects are stored is returned to the MoodView or to the textual interface.

8 Interacting with MoodView

As shown in Figure 1, MoodView interacts with the Catalog Manager, the Query Manager and Database Engine subsystems of the kernel. For schema processing, it communicates with both the Catalog Manager and the Query Manager depending on the complexity of the operation. However for the MOODSQL queries, MoodView sends the query to the Query Manager and results are returned from the Database Engine. These results are displayed using a cursor mechanism. A cursor on the query results is opened by using returned extent information and a cursor handle is obtained. Then with the help of this handle, object browsing on the extent is enabled. The cursor operations include accessing the first object in the extent, getting next or previous object from the current one, and accessing the last object in the extent. Once the object is obtained by the cursor functions, the attribute values can be extracted using the extent information.

The attributes having the types reference, set, list, method or other database types are displayed in a different way so that user can perform operations specific to that attribute. If the attribute type is reference, then referenced object is accessed and shown to the user. For the set and list attributes objects are located and displayed in another window. If the attribute type is method, its activation is deferred to user request. Some example screens showing the schema and the query results are given in Figure 7 and Figure 8 respectively.

Users also have the chance of accessing the MOOD Kernel from their application programs written in C++. For this purpose MOOD Kernel defines a class named UserRequest that contains a method for the execution of SQL statements.

```
class UserRequest {  
    // Local variables of UserRequest  
    .....  
    // Method for Query Execution  
    errorMessage executeQuery( ..... );  
}
```

Whenever a user action requires a database operation at the schema or instance levels, user passes the corresponding SQL statements to the kernel through executeQuery method in the application program. The function

returns a message indicating the success or failure of the operation. Using the cursor mechanism user can access the results of the queries.

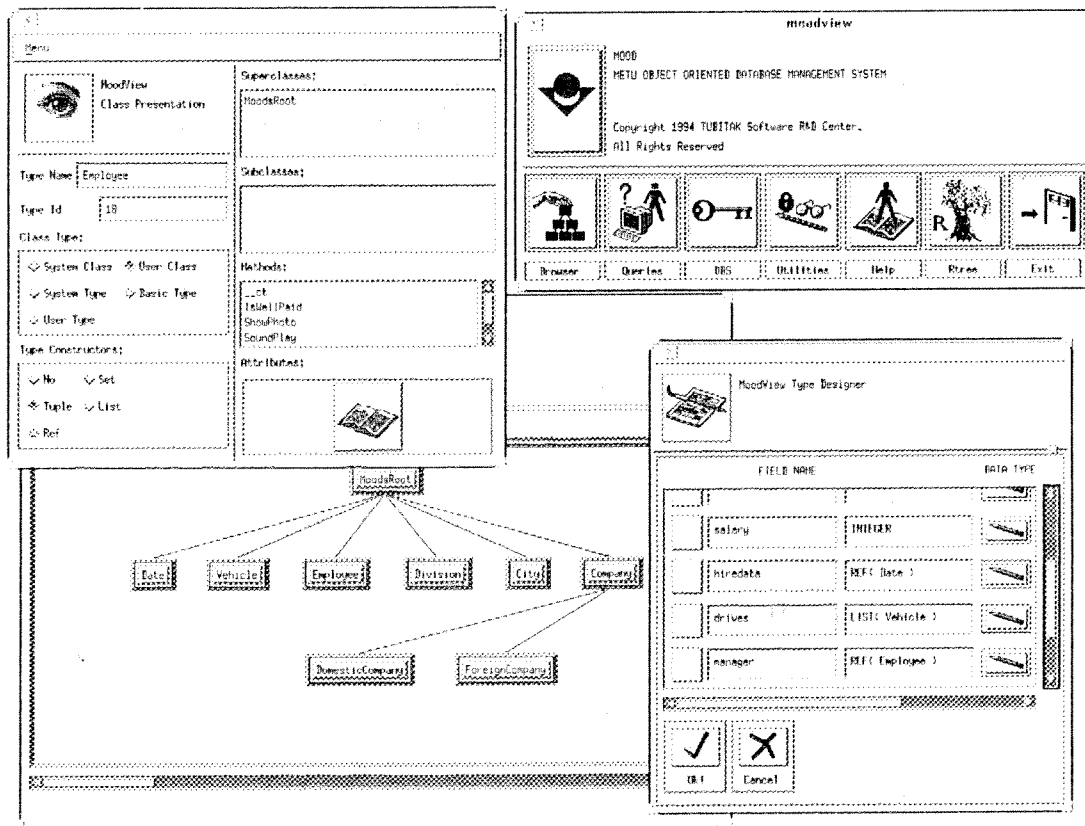


Figure 7. Display of the Example Schema

9 Conclusions

Our design started by deciding to use Exodus Storage Manager after making the observation that it is a powerful, reliable and efficient tool for storage management that also provides for concurrency control and crash recovery. Recognizing the fact that an adhoc query language is essential, our second design decision became to implement an OOSQL whose power is enhanced by using C++ method calls in queries. The persistency needed for catalog management and for the object algebra is implemented through direct

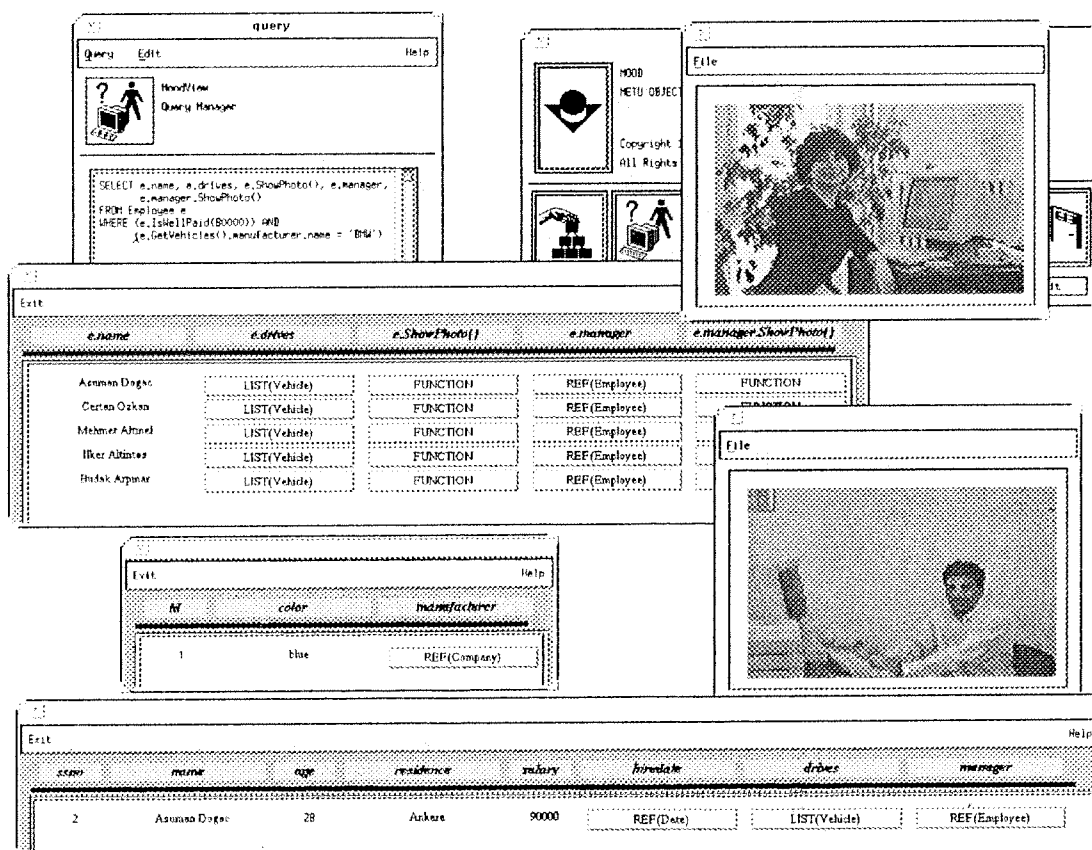


Figure 8. A MOODSQL Query and Resulting Screens

ESM calls. We have left the implementation of a persistent C++ for our system as the final phase. Although persistent C++ is not implemented for the MOOD yet, the following features of the MOOD provide a powerful application development environment:

MOOD kernel can be accessed from C++ application programs and there is no impedance mismatch between MOOD types and C++ types. Furthermore full C++ power is available in the methods.

The paper provides both the design decisions and also the implementation details of the basic components of the MOOD. The implementation details were necessary to provide insight to the techniques used.

The kernel functions implemented are the catalog management, optimization and interpretation of SQL statements and dynamic linking of functions.

Some design decisions are enforced by C++ to prevent the impedance mismatch between the system and C++.

The proposed Dynamic Function Linker approach provides an efficient solution to the late binding problem of object-oriented databases. By dividing the labor between an object-oriented SQL interpreter and a C++ compiler, the interpretation of functions are avoided increasing the overall efficiency of the system. It is clear that previously written C++ methods can be ported to our system without any modification.

The Volcano Query Optimizer Generator is used in developing the MOOD optimizer. VOG provides for fast and easy development of a query optimizer. However VOG's search engine is highly recursive and therefore main memory requirements increase rapidly with increasing number of rules and with complex queries. The next version of VOG will heuristically reclaim memory to overcome this problem [McK 94]. Method optimization is not currently being supported in the MOOD. An object algebra is implemented to realize the operations required by MOODSQL.

The following features are currently missing from the MOOD: View management, security, aggregation operations (SUM, MIN, ...) in MOODSQL queries, NOT/OR in where predicates. Furthermore member function calls with parameters other than constants are not supported because of the limitations of the MOOD optimizer.

Acknowledgements

The authors wish to gratefully acknowledge the MOOD project implementation team: Ilker Altintas, Budak Arpinar, Tolga Gesli, Ismail Tore and Yuksel Saygin.

References

- [Alt 94] Altinel, M, "Design and Implementation of a Dynamic Function Linker and an Object Algebra for the MOOD", MS. Thesis, Dept. of Computer Eng., METU, September 1994.

- [Arp 93a] Arpinar, B., Dogac, A., Evrendilek, C. "MoodView: An Advanced Graphical User Interface for OODBMSs", *SIGMOD Record*, Vol. 22, No. 4., Dec. 1993.
- [Arp 93b] Arpinar, B., "An Advanced Graphical User Interface for Object-Oriented DBMSs: MoodView", M.S. Thesis, Dept. of Computer Eng., METU, September 1993.
- [Bla 93] Blakeley, J., McKenna, W. J., Graefe, G., "Experiences Building the Open OODB Query Optimizer" in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1993.
- [Car 86] Carey, M., DeWitt, D., Richardson, J., Shekita, E., "Object and File Management in EXODUS Extensible Database System", in *Proc. of the 12th Intl. Conf. on VLDB*, 1986.
- [Car 88] Carey M.J., DeWitt D.J., Vandenberg S.L., "A Data Model and Query Language for EXODUS", *Proc. of the ACM SIGMOD Conf.*, 1988.
- [Dar 92] Dar S., Gehani N.H., Jagadish H.V., "CQL++: A SQL for the ODE Object-Oriented DBMS", in *Proc. of Extending Database Technology*, 1992.
- [Deu 91] Deux, O., et al., "The O2 System", *Comm. of the ACM*, Vol. 34, No.10, 1991.
- [Dog 94a] Dogac, A., Ozkan, C., Arpinar, B., Okay, T., Evrendilek, C., "METU Object-Oriented DBMS", *Advances in Object-Oriented Database Systems*, A.Dogac, T. Ozsul., A. Biliris, T. Sellis (Edtrs.) Springer Verlag, 1994.
- [Dog 94b] Dogac, A., et al, "METU Object-Oriented DBMS", Demo description, in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1994.
- [Dog 94c] Dogac, A., MOOD User Manual, 1994.
- [Dur 94] Durusoy, I., " MOOD Query Optimizer", M.S. Thesis, Dept. of Computer Eng., METU, February 1994.

- [ESM 92] Using the Exodus Storage Manager V2.1.1, June 1992.
- [Kim 90] Kim W., *Introduction to Object-Oriented Databases*, The MIT Press, 1990.
- [Mat 93] Mattos, N.M., Meyer-Wegener, K., Mitschang, B., "Grand Tour of Concepts for Object-Oriented Databases from a Database Point of View", *Data and Knowledge Engineering*, No.9, North Holland, 1993.
- [McK 93] McKenna, W. J., "Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer Generator", PhD thesis, Department of Computer Science, University of Colorado, 1993.
- [McK 94] McKenna, W. J., Personal Communication.
- [Ozk 93a] Ozkan, C., Dogac, A., Evrendilek, C., Gesli, T., "Efficient Ordering of Path Traversals in Object-Oriented Query Optimization", In *Proc. of Int. Sym. on Computer and Information Sciences*, Istanbul, Nov. 1993.
- [Ozk 93b] Ozkan, C., "Design and Implementation of an Object-Oriented Query Language, MOODSQL, and its Optimizer", M.S. Thesis, Dept. of Computer Eng., METU, September 1993.
- [Ozk 94] Ozkan, C., Dogac, A., Durusoy, I., "An Efficient Heuristics for Join Reordering", TUBITAK Software R&D Center, Tech. Rep. 94-2, January 1994.
- [She 90] Shekita, E. J., Carey, M. J., "A Performance Evaluation of Pointer Based Joins", in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1990.
- [Sun 90] Sun Microsystems, "Shared Libraries", Programmer's Overview, Utilities and Libraries, 1990.

A Heuristic Approach for Optimization of Path Expressions in Object-Oriented Query Languages

Cetin Ozkan Asuman Dogac Yuksel Saygin
Software Research and Development Center
Scientific and Technical Research Council of Turkiye
Middle East Technical University
06531, Ankara Turkiye

e-mail: asuman@srdc.metu.edu.tr

Abstract

The object-oriented database management systems store references to objects (implicit joins, precomputed joins), and use path expressions in query languages. One way of executing path expressions is pointer chasing of precomputed joins. However it has been previously shown that converting implicit joins to explicit joins during the optimization phase may yield better execution plans. A path expression is a linear query therefore considering all possible join sequences within a path expression is polynomial in the number of classes involved. Yet when the implicit joins are converted to explicit joins, because of path expressions bound to the same bind variable, the query becomes a star query and thus considering all possible joins is exponential in the number of paths involved. This implies that there is a need for improvement by using heuristic in optimizing queries involving path expressions.

A heuristic based approach for optimizing queries involving path expressions is described in this paper. First, given the costs and the selectivities of path expressions by considering a path expression as a unit of processing, we provide an algorithm that gives the optimum execution order of path expressions bound to the same bind variable. For this purpose we derive the formulas for the selectivities of path expressions. Then by using this ordering as a basis we provide a general heuristic approach for optimizing queries involving path expressions.

Two optimizers are developed to compare the performance of the heuristic based approach suggested in this paper with the performance of an optimizer based on an exhaustive search strategy. The exhaustive optimizer is generated through Volcano Optimizer Generator. The results of the experiments can be summarized as follows:

Heuristic optimizer greatly reduces the optimization time. The estimated query execution time of the exhaustive optimizer is slightly better. When it comes to total time, the heuristic optimizer has a superior performance with the increasing number of paths. This result is expected because the time spend in query optimization phase by the exhaustive optimizer is uncomparably larger then the execution time. The heuristic optimizer also performs well for linear queries implying that the heuristic suggested is an effective one.

1. Introduction

The object-oriented database management systems store references to objects (implicit joins, precomputed joins), and use path expressions in query languages. One way of executing path expressions is pointer chasing of precomputed joins. However it has been shown in [Bla 93] that

converting implicit joins to explicit joins during the optimization phase, makes it possible to consider a wider range of join sequences and thus yields better execution plans in most of the cases.

A path expression corresponds to a linear query in relational systems where tables are connected by binary predicates in a straight line. In [Ono 90], it has been shown that the computational complexity (i.e the number of joins that must be considered when using dynamic programming for optimization) of linear queries with composite inners (bushy trees) is $(N^3-N)/6$. If the composite inners are not considered the complexity reduces to $(N-1)^2$. Therefore considering all possible join sequences within a path expression is polynomial in the number of classes involved. Yet when the implicit joins are converted to explicit joins, because of path expressions bound to the same bind variable, the query corresponds to a star query (in fact a hybrid query, i.e., star of linear queries) in relational systems where a table at the center is connected by binary predicates to each of the other surrounding tables. In [Ono 90], it has also been shown that using dynamic programming to optimize a star query with N quantifiers requires evaluating $(N-1)2^{N-2}$ feasible joins. Thus considering all possible joins is exponential in the number of classes involved.

Many existing relational optimizers use heuristics within dynamic programming to limit the join sequences evaluated. One heuristic employed by System R [Sel 79] and R* [Loh 85] constructs only joins in which a single table is joined at each step with the results of previous joins, in a pipelined way.

We have developed a heuristic based approach for optimizing queries involving path expressions. First, given the costs and the selectivities of path expressions by considering a path expression as a unit of processing, we provide an algorithm that gives the optimum execution order of path expressions bound to the same bind variable. For this purpose we derive the formulas for the selectivities of path expressions. Then, by using the results obtained as hint a general heuristic approach is developed.

In order to test the effectiveness of the heuristic proposed, two query optimizers are developed. The first optimizer is based on the heuristic suggested in this paper. The second optimizer is based on an exhaustive search strategy and is generated through Volcano Query Optimizer Generator. Since path optimization mainly involves join enumeration, a subset of the transformation rules given in [Bla 93] are used. The results of the experiments indicate that the heuristic based optimizer has a superior performance with the increasing number of path expressions.

Previous work on optimizing path expressions are given in [Mai 86, Kem 90, Lan 91]. [Mai 86] has concentrated on optimizing path expressions by exploiting path indices. [Kem 90] suggested access support relations, which are separate structures to store object references for processing path expressions. In this work the optimizer may choose between the traversals that start at either end of a path. In [Lan 91] a more general framework has been suggested for processing path expressions where the traversal may start anywhere in a path and other operations like selects and joins can be interleaved with path traversals. Recently the design and implementation of a query optimizer based on complete extensible framework has been reported in [Bla 93] where the query optimizer of the system is generated by Volcano Query Optimizer Generator (VOG) [McK 93, Gra 93]. The Volcano Query Optimizer Generator is a data model independent tool that is used to develop a query optimizer for a DBMS. VOG generates all possible execution plans and its search

algorithm uses dynamic programming with branch-and-bound pruning based on cost. The Volcano generated optimizers produce the optimum execution plan when the transformation rules and support functions are provided properly.

In section 2, the cost model is presented and the formulas for the selectivity of path expressions are derived. In Section 3, an algorithm that gives the optimum execution order of path expressions bound to the same bind variable is provided by considering a path expression as a unit of processing. Section 4 presents the principles of the heuristic based optimizer. In Section 5, the performance results are given. Finally, Section 6 contains the conclusions.

2. Cost Model

In this section the cost model which is used in both of the query optimizer implementations is explained. The cost model parameters are presented and the formulas for the selectivity of path expressions are derived. The cost formulas for implementing the join operations with different join techniques are provided in the Appendix.

2. 1. Cost Model Parameters

The cost model parameters which are used in various selectivity calculations that form the basis of the cost calculation of path traversals, are given in Table 1. Similar cost model parameters have been defined in [Kemp 90], [Ber 92] and [Ber 93]. In defining the cost model, [Kem 90] considers the extensions of classes where in [Ber 92] and [Ber 93], the class inheritance hierarchy is also taken into account. Our cost model considers the class extensions. Furthermore we have defined some more parameters that serve better to our purposes.

In the Table 1, C is a class, A is either an attribute or a parameterless method of class C with an atomic return type which is treated in the same way as an atomic attribute.

Parameter	Definition
C	Total number of instances of C
nbpages(C)	Total number of pages in which class C is stored
size(C)	Size of an instance of class C
nonnull(A,C)	The proportion of the instances in class C with attribute A being not null
fan(A,C,D)	The average number of instances of class D that are referenced by the instances of C through attribute A
totref(A,C,D)	The total number of objects in the class D which are referenced by at least one object in class C through attribute A. (2)
dist(A,C)	Number of distinct values of the atomic attribute A of class C
max(A,C)	The maximum value of the atomic attribute A of class C
min(A,C)	The minimum value of the atomic attribute A of class C

Table 1 . Cost Model Parameters

The number of the total references from class C to class D through the attribute A is denoted by

totlinks(A,C,D) and given by the following equation :

$$\text{totlinks}(A,C,D) = \text{fan}(A,C,D) * |C|$$

The probability that an instance of class D is referenced by the instances of class C through the attribute A is denoted by hitprb(A,C,D) and given by the following equation:

$$\text{hitprb}(A,C,D) = \text{totref}(A,C,D) / |D|$$

2.2. Selectivity

A simple predicate in the system is a triplet of the form $\langle P_1, \Theta, \text{oprnd} \rangle$, where P_1 is a path expression, Θ is a comparison operator ($=, <>, \geq, \leq, >, <$), and oprnd is either a constant or another path expression.

2.2.1. Selectivity for Atomic Attributes

The well-known selectivity calculations assuming the uniform distribution of the atomic values described in [Ozk 90] are used. The selectivity of the expression "s.A = constant", denoted as σ , where s is a bind variable binding to a class C, and A is an atomic attribute, is given by the following formula where dist is distinct values of A in C:

$$\sigma(s.A) = 1 / \text{dist}(A,C)$$

2.2.2. Selectivity of Path Expressions

Assume that there is a path expression involving m classes referenced through attributes, A_1 through A_m , where A_1 through A_{m-1} is constructed using the set or the reference constructors. A_m is an atomic attribute and A_i is an attribute of class C_i . We need to calculate the selectivity, $\sigma_{\text{path}}(p.A_1.A_2...A_m, \Theta)$, for a single path expression "p.A₁.A₂...A_m Θ c", where Θ is a comparison operator and c is a constant. For this calculation we define the shorthand notation for some of the previously mentioned parameters as follows:

$$\begin{aligned} \text{fan}_i &= \text{fan}(A_i, C_i, C_{i+1}) \\ \text{totref}_i &= \text{totref}(A_i, C_i, C_{i+1}) \\ \text{totlinks}_i &= \text{totlinks}(A_i, C_i, C_{i+1}) \quad \text{where } 1 \leq i \leq m-1. \end{aligned}$$

The calculation of the selectivity of " $A_m \Theta c$ ", $\sigma(A_m)$, is clear from the previous section. Therefore the expected number of instances of C_m , denoted by k_m , that satisfies this condition is:

$$k_m = |C_m| * \sigma(A_m)$$

It is clear that when there is no selection on the A_m attribute $k_m = |C_m|$.

In forward traversal, assuming that we start with k objects of class C_1 and traverse the path p.A₁.A₂...A_i in forward direction, the expected number of objects of class C_{i+1} , denoted by fref, is given by the following formula:

$$\text{fref}(p.A_1.A_2...A_i.k) = \begin{cases} k & , i = 0 \\ c(\text{totlinks}_i, \text{totref}_i, \text{fref}(p.A_1.A_2...A_{i-1},k) * \text{fan}_i) & , i > 0 \end{cases}$$

where, $c(n,m,r)$ is an approximation to the following statistical problem: Given n objects uniformly distributed over m colors, how many different colors c are selected if we take just r objects? This statistical problem has been solved by using different mathematical approximations. An approximation assumed in [Cer 85] is as follows:

$$c(n,m,r) = \begin{cases} r & , r < m/2 \\ (r+m)/3 & , m/2 \leq r \leq 2m \\ m & , r > 2m \end{cases}$$

Note that [Car 75] and [Yao 77], in approximating the number of page accesses to a file for a given query, has defined better approximations to this statistical problem. However we have calculated and compared the average error per unit time, and found out that $c(n,m,r)$ well serves our purposes.

Starting with one instance of class C_1 , the number of objects of class C_m obtained at the end of forward path traversal is given by $\text{fref}(p.A_1...A_{m-1},1)$. On the other hand, k_m objects have been selected through the predicate $A_m \Theta c$. Then the selectivity of a path expression, $p.A_1.A_2...A_m \Theta c$, which is defined to be the probability of at least one object being in common in two sets with cardinalities $\text{fref}(p.A_1.A_2...A_{m-1},1)$ and $k_m * \text{hitprb}(A_{m-1},C_{m-1},C_m)$ respectively, is given by

$$\sigma_{\text{path}}(p.A_1.A_2...A_m, \Theta) = o(\text{totref}_{m-1}, \text{fref}(p.A_1.A_2...A_{m-1},1), k_m * \text{hitprb}(A_{m-1},C_{m-1},C_m))$$

where $o(t,x,y)$ is the probability that there exists at least one object in common in two sets selected with replacement out of t distinct objects and is defined as follows:

$$o(t,x,y) = 1 - C(t-x,y) / C(t,y)$$

where C stands for combination, and x and y are the cardinalities of the two sets respectively.

3. On the Execution Order of Path Expressions

Consider m path expressions which are bound to the same bind variable, say p , in an AND-term:

$$\begin{aligned} & p.a_{11}.a_{12}...a_{1n1} \Theta_1 c_1 \\ & p.a_{21}.a_{22}...a_{2n2} \Theta_2 c_2 \\ & \cdot \\ & \cdot \\ & p.a_{m1}.a_{m2}...a_{mnm} \Theta_m c_m \end{aligned}$$

Assume F_i denotes the cost of executing the path expression i , $p.a_{i1}.a_{i2}...a_{in_i} \Theta_i c_i$, and let the selectivity of this path expression be $s_i = \sigma_{\text{path}}(p.a_{i1}.a_{i2}...a_{in_i} \Theta_i c_i)$.

Given the cost and the selectivity of each of the path expressions, the problem of finding the least costly execution order of these path expressions can be stated as the following minimization problem:

Find a permutation of the integers 1 through m stored in $i[1]$ through $i[m]$ which minimizes

$$f = F_{i[1]} + s_{i[1]} * F_{i[2]} + s_{i[1]} * s_{i[2]} * F_{i[3]} + \dots + s_{i[1]} * s_{i[2]} * \dots * s_{i[m-1]} * F_{i[m]}$$

where F_j and s_j , $j \in i[1]$ through $i[m]$, are the cost of traversing and the selectivity of the j^{th} path expression respectively. In other words, we are trying to minimize the objective function f , denoting the total cost of executing m path expressions in the order induced by the array i .

Theorem 1 : Assume π denotes a permutation of the integers 1 through m such that path expression indices are sorted in ascending order of $F_i / (1 - s_i)$ values, such that $1 \leq i \leq m$. This π minimizes the objective function f .

Sketch of Proof : By induction on the number of path expressions.

It is true for 2 path expressions. In this case $f = F_1 + s_1 F_2$ or $f = F_2 + s_2 F_1$.

If $F_1 + s_1 F_2 < F_2 + s_2 F_1$ then by simple manipulation,

$F_1 / (1 - s_1) < F_2 / (1 - s_2)$ is obtained.

Assuming that it is true for m path expressions and we will try to show that it is also true for $m+1$ path expressions. Let us assume that $F_i / (1 - s_i) < F_{i+1} / (1 - s_{i+1})$ for $1 \leq i \leq m-1$, and assume also that $F_j / (1 - s_j) < F_{m+1} / (1 - s_{m+1}) < F_{j+1} / (1 - s_{j+1})$ for some j where $1 < j < m$.

We claim that

$f_1 = F_1 + s_1 F_2 + \dots + s_1 s_2 \dots s_{j-1} F_j + s_1 s_2 \dots s_{j-1} s_j F_{m+1} + s_1 s_2 \dots s_{j-1} s_j s_{m+1} F_{j+1} + \dots + s_1 s_2 \dots s_{j-1} s_j s_{m+1} s_{j+1} \dots s_{m-1} F_m$ is minimum. Assume on the contrary that,

$f_2 = F_1 + s_1 F_2 + \dots + s_1 s_2 \dots s_{k-1} F_k + s_1 s_2 \dots s_{k-1} s_k F_{m+1} + s_1 s_2 \dots s_{k-1} s_k s_{m+1} F_{k+1} + \dots + s_1 s_2 \dots s_{k-1} s_k s_{m+1} s_{k+1} \dots s_{m-1} F_m$ is minimum with the assumption that $k < j$ without loss of generality.

First observe that by the induction hypothesis, it can be shown that with the addition of the $m+1^{\text{st}}$ path expression, the relative order of the previous path expression indices do not change.

Therefore, if we parenthesize f_2 by $s_1 s_2 \dots s_{k-1} s_k$ starting from the $k+1^{\text{st}}$ term, we observe that the induction hypothesis stating that aforementioned sort order minimizes the objective function for $m+1-k \leq m$ path expressions, is violated. \square

The strong assumption underlying this approach is that a path expression is an indivisible unit of processing. However as shown in [Bla 93] by converting implicit joins to explicit joins, wider range of join sequences can be obtained and thus better (i.e. less costly) execution plans can be produced. It is clear that when the implicit joins are converted to explicit joins, because of the path expressions bound to the same bind variable, the query becomes a hybrid query, i.e., star of linear queries. Thus when we allow implicit joins to be converted to explicit joins, by using an exhaustive search strategy it is possible to obtain the optimum execution plan. However the number of join sequences to be considered is exponential in the number of classes involved. This observation indicates that heuristic is necessary to improve the performance of object-oriented queries involving path expressions.

4. A Heuristic based Approach for Object-Oriented Queries Involving Path Expressions

In this section we propose a heuristic based method for optimizing object-oriented queries involving path expressions. In this method, we first order the path expressions by using Theorem 1. Procedure 4.1 which implements Theorem 1 decides on the execution order of the path expressions. In Procedure 4.1 the cost of executing the path expression is taken as its forward traversal cost since we are using this cost as a hint to order the path expressions. Then for the chosen path, heuristic is used again as given in Procedure 4.2, to decide on the execution order of the joins within this path expression.

The heuristic we propose in Procedure 4.2 is to favor the less costly and more selective join at each iteration. Notice that the cost and the selectivity of a join operation directly effects the join order but their effect on the order varies depending upon their values. Therefore we have tried a number of evaluation functions that all favor less cost and more selectivity but the effect of cost and selectivity on the evaluation function is different in each of them.

Before proceeding further we will provide some definitions to be used in the Procedures 4.1 and 4.2 :

Definition 4.1 Size Selectivity : The size selectivity of a join operation, $C = A \bowtie B$ where A and B are two classes, is denoted by $\sigma_{size}(A,B)$, and defined as

$$\sigma_{size}(A,B) = nbpages(C) / (nbpages(A) + nbpages(B))$$

where $nbpages(C)$ is the estimated number of pages of the class produced as a result. \square

Definition 4.2 Per-Unit Cost: The per-unit cost of a join operation, $C = A \bowtie B$ where A and B are two classes and J_{cost} is the minimum of the costs of performing this join operation with different join implementation techniques, is denoted by $P_{cost}(A,B)$, and defined as

$$P_{cost}(A,B) = J_{cost}(A,B) / (|A| + |B|). \quad \square$$

Definition 4.3. Evaluation Function: In defining the evaluation function, we make the following observation: the cost and the selectivity of each join operation in a join sequence directly effect the join order. As an example consider $A \bowtie B \bowtie C$ with costs $J_{cost}(A,B)$, $J_{cost}(B,C)$ and selectivities $\sigma_{size}(A,B)$, $\sigma_{size}(B,C)$ and assume $J_{cost}(A,B) > J_{cost}(B,C)$ and $\sigma_{size}(A,B) < \sigma_{size}(B,C)$. Here if we only consider cost we will execute $B \bowtie C$ first, but since $\sigma_{size}(A,B) < \sigma_{size}(B,C)$, executing $A \bowtie$ (The resulting relation) may be more costly depending upon the cost and selectivity values. Therefore less costly and more selective join must be favored at the same time. Again depending upon the cost and selectivity values it may be beneficial to increase the effect of cost or selectivity on the join order. With these observations and with some experimentation we have defined four evaluation functions. In each of these functions low cost and high selectivity are favored however from $\psi^1(A,B)$ to $\psi^4(A,B)$, the effect of cost in the evaluation function is reduced while the effect of the selectivity is amplified.

$$\begin{aligned} \psi^1(A,B) &= J_{cost}(A,B)/(1-\sigma_{size}(A,B)) \\ \psi^2(A,B) &= J_{cost}(A,B)*\sigma_{size}(A,B) \end{aligned}$$

$$\psi^3(A,B) = P_{cost}(A,B) * \sigma_{size}(A,B)$$

$$\psi^4(A,B) = \ln J_{cost}(A,B) * e^{\sigma_{size}(A,B)} . \square$$

In the following we present the algorithms implementing our heuristics.

Procedure 4.1 The Evaluation Order of Path Expressions

```

double orderPathExp( List ListofPathExpressions )
{
    double totalCost = 0;
    int k=|Cp|;
    PathExpression p,p';
    while( ListofPathExpressions is not empty )
    {
        for each p in ListofPathExpressions
        {
            Calculate the forward traversal cost Fp for each path expression;
            Calculate the selectivity of the each path expression , σpathp ;
            Mp=Fp / ( 1- σpathp );
        }
        p' = min(Mp) where p ∈ ListofPathExpressions;
        totalCost=totalCost+OrderImplicitJoins(p', schemaInfo);
        k=Cardinality from the schemaInfo;
        remove p' from ListofPathExpressions ;
    }
    return totalCost;
}

```

Procedure 4.2. Implicit Join Ordering

Let us assume that there is a path expression p.a₁.a₂...a_n where p is bound to C_n and a_i references to the instances of the class C_i (1 ≤ i ≤ n-1). J_{cost}(C_i, C_j) and σ_{size}(C_i, C_j) denote the individual cost and selectivity of the temporary collection obtained by joining class C_i and class C_j.

```

double orderImplicitJoins (List PathExpression, structure schemaInfo)
// the list PathExpression contains the classes { C0, C1,..., Cn-1 }
{
    List tempPE;
    double totalCost;
    for t=1 to 4; // ψt denotes the Evaluation function in use
    {
        tempPE=PathExpression;
        totalCostt=0;
        for each <Ci, Ci+1> in tempPE do
        {
            calculate Jcost(Ci, Ci+1) , σsize(Ci, Ci+1), and ψt(Ci, Ci+1) ;
            // In evaluating ψt(Ci, Ci+1), Jcost(Ci, Ci+1) is the minimum of the costs of applicable
            // join techniques given in the Appendix.
        }
    }
}

```

```

while( tempPE is not empty) do
{
  select  $C_k = \langle C_i, C_{i+1} \rangle$  which gives the minimum value for  $\psi^l(C_i, C_{i+1})$ ;
  Generate schemaInfo for  $C_k$ ;
   $totalCost^l = totalCost^l + \psi^l(C_i, C_{i+1})$ ;
  Delete  $i^{th}$  and  $i+1^{st}$  items from tempPE;
  Compute  $J_{cost}(C_{i-1}, C_k)$ ,  $\sigma_{size}(C_{i-1}, C_k)$ ,  $J_{cost}(C_k, C_{i+2})$ ,  $\sigma_{size}(C_k, C_{i+2})$ ,  $\psi^l(C_{i-1}, C_k)$ ,
  and  $\psi^l(C_k, C_{i+2})$ ;
  Insert  $C_k$  after  $C_{i-1}$  to the list tempPE;
}
}
totalCost =  $\min_l (totalCost^l)$  ;
schemaInfo = schemaInfo for  $C_k$ 
return totalCost;
}

```

It should be noted that when a temporary collection C_{ij} is obtained by joining class C_i and class C_j , the references from class C_{i-1} can not be used to reach the objects in C_{ij} . For such cases, only explicit join techniques can be used.

The time complexity of this algorithm is $O(n^2)$.

5. Performance Evaluation

Two optimizers are developed for optimizing the queries involving path expressions. The first optimizer uses the heuristics described in Section 4. The second optimizer is generated through Volcano Query Optimizer Generator [McK 93]. The Volcano generated optimizers produce the optimum execution plan when the transformation rules and support functions are provided properly because of its exhaustive search strategy. In this implementation the transformation and implementation rules given in [Bla 93] are used. However since we are considering only the join and path expression optimizations, in other words, join enumeration, some of the transformation rules given in [Bla 93] are not necessary and therefore they are disabled. The transformation rules used are:

1. The rule implementing the join commutativity.

$JOIN \ ?o1 \ (\ ?1 \ ?2 \) \ ->! \ JOIN \ ?o2 \ (\ ?2 \ ?1 \)$

The $->!$ sign in the rule denotes that this transformation is performed exactly once. This prevents the infinite recursion.

2. The rule implementing join associativity.

$JOIN \ ?o1 \ (\ JOIN \ ?o2 \ (\ ?1 \ ?2 \) \ ?3 \) \ -> \ JOIN \ ?o3 \ (\ ?1 \ JOIN \ ?o4 \ (\ ?2 \ ?3 \) \)$

Join associativity together with the join commutativity, provides for all possible join sequences.

3. The rule that converts a materialize node into joins.

$\text{MAT } ?o1 (?1) \rightarrow \text{JOIN } ?o2 (?1 \text{ GET_SET } ?o3 ())$

Notice that materialize operator indicates a path expression of length one. By converting a materialize operator into join it becomes possible to apply the transformation rules on join associativity and on join commutativity.

4. The rule that interchanges two successive materialize nodes.

$\text{MAT } ?o1 (\text{MAT } ?o2 (?1)) \rightarrow ! \text{MAT } ?o3 (\text{MAT } ?o4 (?1))$

The application of this transformation rule may results in other transformations.

With these rules the VOG generated optimizer finds the optimum join ordering for the queries involving path expressions.

5.1. Testbed

Both of the optimizers are run on a Sun Sparc 2 station which has Sun 4/40 CPU, 12 MB of memory and 32 MB swap space. Each of the optimizers were the only active process during the experiments.

A random query generator is used to generate the queries with m path expressions of length n where both m and n ranges between 1 and 9. With our hardware, 12 MB memory, the optimizer generated through Volcano can not run queries when m exceeds 9, or when $m*n$ exceeded 12. Note that, McKenna, was able to go up to 12 joins for star queries with 32 MB of main memory [McK 93]. The reason for this behavior is that the Volcano Optimizer Generator's search engine is highly recursive, and therefore as the number of equivalent classes in the query increases, optimizer rapidly exhausts the memory. The next version of VOG will heuristically reclaim memory to overcome this problem [McK 94].

For each n, m pair, 50 random queries are generated and the average values for optimization time and execution costs are obtained. The size of the classes involved ranged between 1000 and 100,000 objects where object sizes ranges from 100 to 2000 bytes. Exactly the same queries are run on both of the optimizers.

In the cost calculations, the available buffer space for executing the queries is assumed to be 4MB. Furthermore we have assumed that the results of the join operations are written back to disk. The queries generated do not contain select operator.

The results of the some of the test runs are tabulated in Table 5.1.

5.2. Query Optimization Time

The Figures 5.1 and 5.2 depict the query optimization times of the optimizers for linear and star queries respectively. From these figures it is clear that the heuristic based optimizer greatly reduces the optimization time. The explanation for this behavior is two fold:

1. The number of joins enumerated by the heuristic based optimizer for a path expression of length N is $3*(N-1)$. In procedure 4.2 we first generate $N-1$ joins, choose one and for the remaining $N-1$ joins, generate 2 more joins. However exhaustive optimizer generates $(N^3-N)/6$ joins for linear queries. Heuristic based optimizer order the path expressions by using Procedure 4.1 and for m path expressions, forward traversal cost is calculated m^2 times. Yet exhaustive optimizer, by converting implicit joins into explicit joins creates star queries and generates $(N-1)2^{N-2}$ feasible joins.

2. In the heuristic based optimizer, the data structures and the algorithm itself are very simple. Therefore it spends less time in optimization. This fact is clear from the comparison of the optimization times of the two approach for 1 join as depicted in Table 5.1.

In Figure 5.3 the optimization time of the exhaustive optimizer for star and linear queries are plotted and Figure 5.4 shows the optimization time of the heuristic based optimizer which verify the analytical formulas.

Table 5.1 Results of some of the test runs

		Exhaustive	Exhaustive	Heuristics	Heuristics
Path Length (n)	No of Paths (m)	Optimiztion time(secs.)	Execution time(secs)	Optimization time(secs.)	Execution time(secs.)
1	1	0.1854	78.9585	0.053	78.9585
1	3	1.203	197.705	0.138	200.806
1	5	11.261	175.962	0.298	176.65
1	9	1881.438	222.342	0.739	226.257
3	1	0.911	168.517	0.098	168.565
3	3	74.124	204.05	0.349	218.255
4	3	385.958	154.577	0.475	171.262
5	1	2.748	420.894	0.149	421.29
6	2	98.233	178.341	0.439	185.088
7	1	7.198	277.837	0.217	277.995
8	1	11.71	435.722	0.261	436.019
9	1	17.082	260.921	0.293	263.913

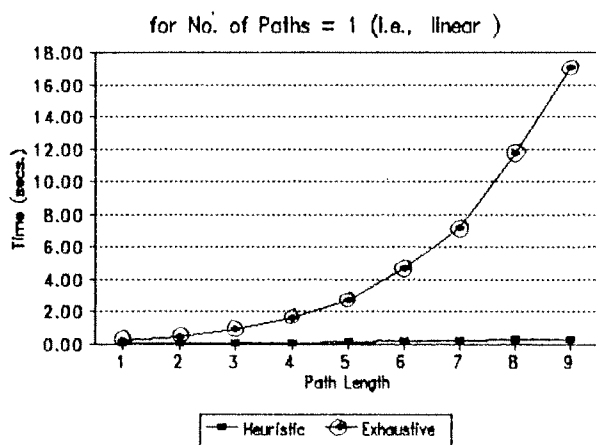


Figure 5.1 Query optimization time of linear queries

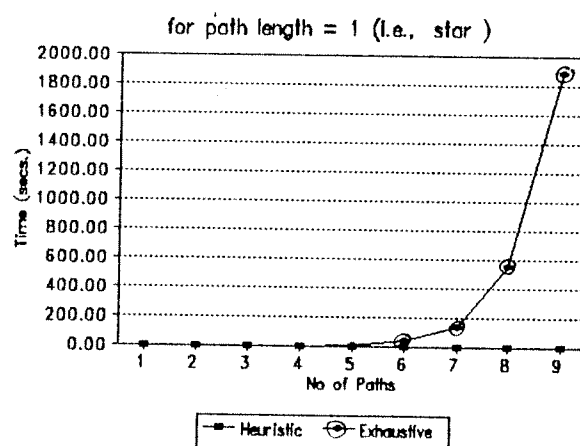


Figure 5.2 Query optimization time of star queries

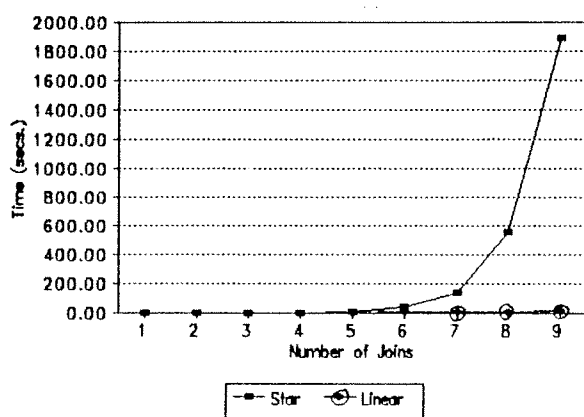


Figure 5.3 Query optimization time of the exhaustive optimizer

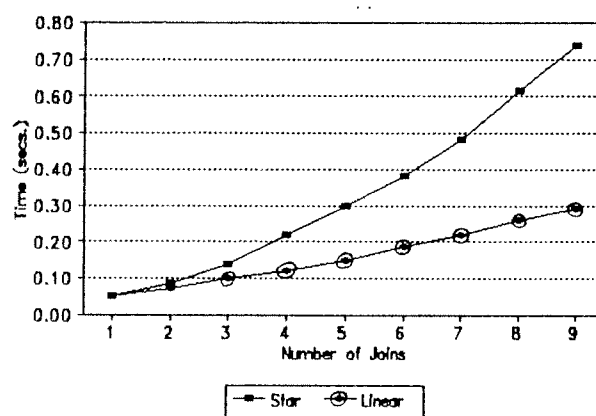


Figure 5.4 Query optimization time of the heuristic based optimizer

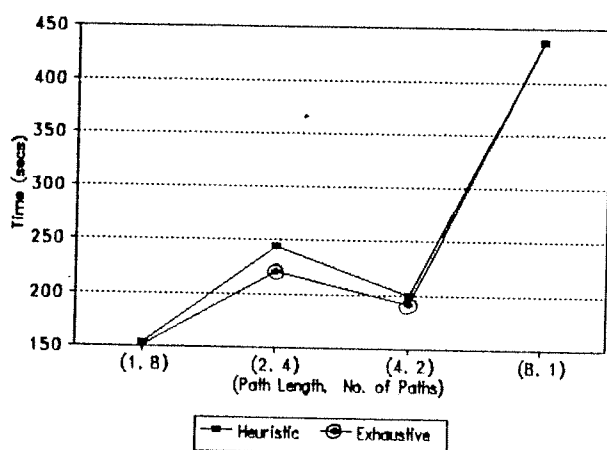


Figure 5.5 Estimated query execution time for a mixture of queries

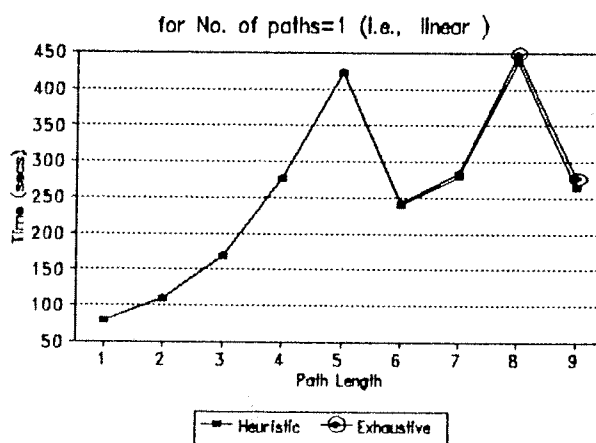


Figure 5.6 Total time for linear queries

for path length = 1 (i.e., star)

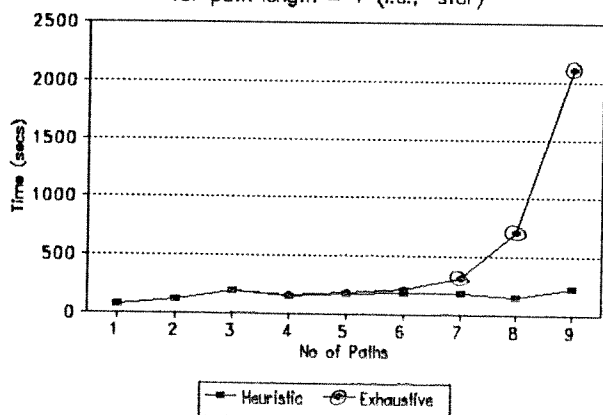


Figure 5.7 Total time for star queries

for no of paths = 1 (i.e., linear)

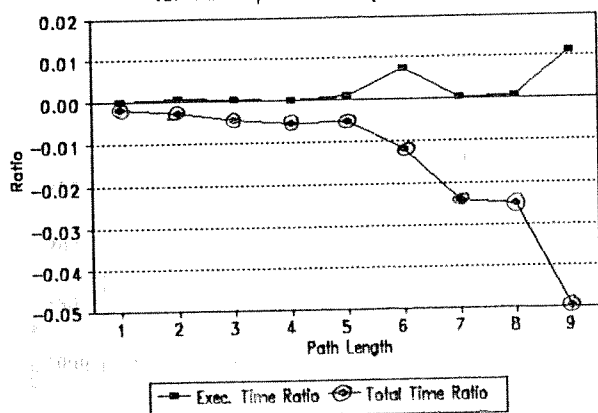


Figure 5.9 Time error for linear queries

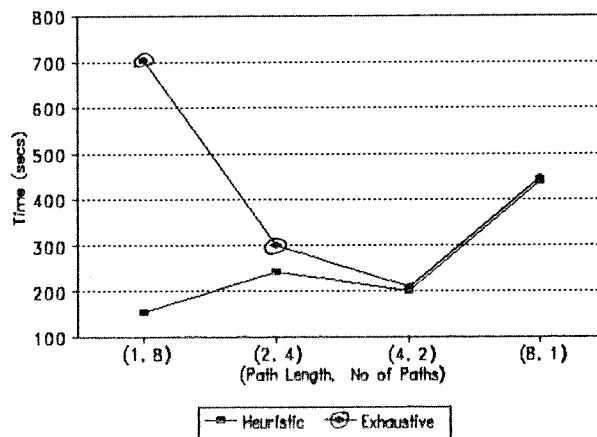


Figure 5.8 Total time for a mixture of queries

for path length = 1 (i.e., star)

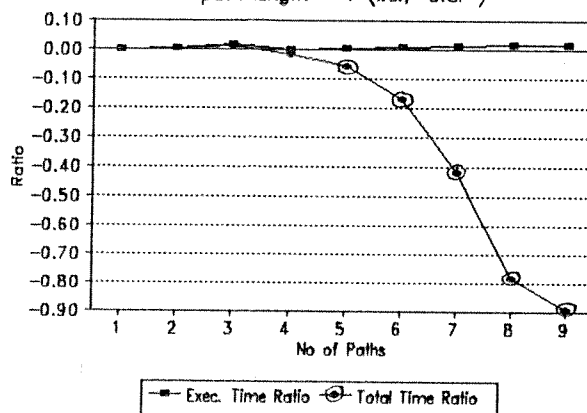


Figure 5.10 Time error for star queries

5.3. Execution Time and Total Time

The costs are estimated by using the cost model presented in Section 2 after obtaining the query execution plans from both of the optimizers. Figure 5.5 indicates that there is no drastic difference between the plans generated by the two optimizers; the plans generated by the exhaustive optimizer are slightly better than the plans generated by the heuristic optimizer. When we consider the total time, that is, the query optimization time plus the query execution time, heuristic based optimizer outperforms the exhaustive optimizer when number of paths exceed 7 as shown in Figure 5.7. For linear queries, heuristic based optimizer is slightly better than the exhaustive optimizer as shown in Figure 5.6. These results indicate that the heuristic used is an effective one.

Figure 5.8 indicates that the total query execution time is greatly reduced for star queries. In Figure 5.9 and Figure 5.10 we have plotted the execution time error and total time error for linear and star queries respectively, which are defined as follows:

execution time error = execution time of heuristic based optimizer - execution time of exhaustive optimizer / execution time of exhaustive optimizer

total time error = total time of heuristic based optimizer- total time of exhaustive optimizer/ total time of exhaustive optimizer

From Figure 5.9, it is clear that for linear queries, the plans produced by the heuristic based optimizer deviates from the optimal plans by 1 percent for path length 9, but when the total time is considered for the same path length, the heuristic based optimizer performs 5% better. When star queries are considered the performance gain in total time is a drastic 90% as shown in Figure 5.10 although slightly worse plans are produced by the heuristic based optimizer.

6. Conclusions and Future Work

Because of the exponential nature of the query optimization for star queries, many existing relational optimizers use heuristics within dynamic programming to limit the join sequences evaluated. A path expression in an object-oriented query language is a linear query but because of the path expressions bound to the same bind variable, the query becomes a hybrid query when the implicit joins are converted to explicit joins.

Two optimizers are developed to compare the performance of the heuristic based approach suggested in this paper with the performance of an optimizer based on exhaustive optimization. The exhaustive optimizer is generated through Volcano Optimizer Generator. The results of the experiments indicate that the heuristic based optimizer performs as good as exhaustive optimizers for queries involving a single path expression (linear queries) and has a superior performance with the increasing number of path expressions.

As a future work we plan to generalize the heuristics suggested in this paper to relational systems to involve explicit joins and different bind variables and also to compare its performance with 2 Phase-Optimization technique given in [Ioa 90].

References

- [Ber 93] Bertino, E., Martino, L., Object-Oriented Database Systems: Concepts and Architectures, Addison-Wesley, 1993.
- [Ber 92] Bertino, E., Foscoli, P., "A Model of Cost Functions for Object-Oriented Queries", In Proc. of 5th International Workshop on Persistent Object Systems, Italy, September 1992.
- [Bla 93] Blakeley, J. A., McKenna, W. J., Graefe, G., "Experiences Building the Open OODB Query Optimizer", Proc. of the ACM SIGMOD Conf., 1993.
- [Car 75] Cardenas A.F., "Analysis and Performance of Inverted Data Base Structures", Comm. ACM, May 1975.
- [Cer 85] Ceri, S., Pelagatti, G., Distributed Database systems, McGraw Hill, 1985
- [Gra 93] Graefe, G., McKenna, J. W., "The Volcano Optimizer Generator: Extensibility and Efficient Search", Proc. IEEE Conf. on Data Eng., Vienna Austria, 1993.
- [Ioa 90] Ioannidis, Y., Kang, Y., "Randomized Algorithms for Optimizing Large Join Queries", Proc. of the ACM SIGMOD Conf., 1990.
- [Kem 90] Kemper A., Moerkotte G., " Access Support in Object Bases", Proc. of the ACM SIGMOD Conf., 1990.

- [Lan 91] Lancelotte, R. S. G., Valduriez, P., Ziane, M., Cheiney, J-P., "Optimization of Nonrecursive Queries in OODBs", In Proc. of the Second Intl. Conf. on Deductive and Object-Oriented Databases, 1991.
- [Loh 85] Lohman, G.M. et. al., "Query Processing in R*", Query Processing in Database Systems, Kim, Batory, Reiner, eds. Springer-Verlag, 1985.
- [Mai 86] Maier, D, Stein, J., "Indexing in an Object-Oriented DBMS", in Proc. Intl. Workshop on OODBMSs, September 1986.
- [McK 93] McKenna, W.J., "Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer Generator", Ph. D. Thesis, Univ. of Colorado, 1993.
- [McK 94] McKenna, W.J., Personal communication, 1994.
- [Ono 90] Ono, K., Lohman, G. M., "Measuring the Complexity of Join Enumeration in Query Optimization", Proc. of Intl. Conf. on Very Large Databases, 1990.
- [Ozk 90] Ozkaran E., "Database Management Concepts, Design and Practice", Prentice-Hall, 1990.
- [Sal 88] Salzberg B., "File Structures, an Analytic Approach", Prentice-Hall International, Inc, 1988.
- [Sel 79] Sellinger, P.G., "Access Path Selection in a Relational Database Management System", Proc. of the ACM SIGMOD Conf., 1979.
- [She 90] Shekita, E. J., Carey, M. J., "A Performance Evaluation of Pointer-Based Joins", Proc. of the ACM SIGMOD Conf., 1990.
- [Yao 77] Yao S.B., "Approximating Block Access in Database Organizations", Comm. of the ACM, Vol. 20, No. 4, April 1977.

Appendix

Cost Analysis of Join Operator in a Paged Environment

Physical parameters of the disk, which are used in the cost formulas are as shown in Table A1 [Sal88].

Parameter	Definition
B	page size
btt	page transfer time
ebt	effective page transfer time
r	average rotational latency
s	average seek time

Table A1. Physical Parameters for hard disk

The cost of sequential accesses to b pages is denoted by $SEQCOST(b)$ and is calculated as

$$SEQCOST(b) = s + r + b * ebt .$$

The cost of random access to b pages, denoted by $RNDCOST(b)$ is

$$RNDCOST(b) = b * (s + r + btt).$$

Cost of Explicit Join Operation

Nested-Loop With Hashing: In this method the first relation with b pages is divided into partitions such that half of the available buffer space is enough to construct a hash table for each partition [Sal 88]. Thus in total there are $\text{ceiling}(b/((\text{no. of available buffer pages}/2)*.7))$ segments of the first relation where .7 is the load factor. For each partition a hash table is constructed in the memory on the join attribute. After the construction of hash table, second relation with b' pages is scanned using the same hash function and matches are found. The cost of this method is :

$$J_{cost} = SEQCOST(b) + nsg * SEQCOST(b') + SEQCOST(b'')$$

where b'' is the number of pages in the output relation and $SEQCOST(b'')$ denotes the cost of writing the result of the join operation back to disk.

Cost of Hash Partition Join: The cost of Hash-Partition Join in relational databases for two relations is given as [Sal 88]:

$$J_{cost} = 3 * SEQCOST(b + b') + [1 + (b'/b)] * nsg^2 * (r + s) + SEQCOST(b'')$$

where b and b' are the number of pages in outer and inner relations respectively, and nsg denotes the number of segments of the smaller relation;

$nsg = \text{ceiling}(b/\text{no. of available buffer pages})$.

Cost of Sort Merge Join: The cost of Sort-Merge Join in relational databases for two relations is given as [Sal 88]:

$$J_{cost} = 3 * SEQCOST(b + b') + 2 * (nsg1^2 + nsg2^2) * (r + s) + SEQCOST(b'')$$

where b and b' are the number of pages of the involved relations, and $nsg1, nsg2$ denote the number of segments of relations as defined above.

Cost of Implicit Join Operation

Throughout the following cost analysis, it is assumed that the objects of Class C are to be joined (implicitly) through the attribute A of the class C with the objects of Class D, which can be explicitly shown as $C.A = D.\text{self}$.

Cost of Forward Traversal: The cost of forward traversing the objects of C into D is given by the following formula:

$$J_{cost} = SEQCOST(nbpages(C)) + RNDCOST(|C| * fan(A, C, D)) + SEQCOST(b'')$$

This is worst case formula where there are no page hits in the buffer allocated for the objects of Class D.

Cost of Using Pointer-based Hash Partition Join: In case of pointer-based Hash-Partition Join [She 90], the referencing class, i.e., class C is hashed on the pointer field A and partitions are created. Then for each object of class C, the pointer, C.A, is chased to retrieve the object from class D. So the cost of joining the objects of class C with the objects of class D by using pointer-based hybrid hash join can be given as follows:

$$J_{cost} = 3 * SEQCOST(nbpages(C)) + RND COST(nbpg) + SEQCOST(b'')$$

where $nbpg = nbpages(D) * (1 - (1 - 1/nbpages(D))^{\alpha})$ and $\alpha = totref(A,C,D)$.

Note that this join technique can only be applied when constructor of attribute A is Reference.

Nesne-yönelimli veri tabanları konusunda dünyada yapılan çalışmaların bir özeti

Nesne yönelimli veri tabanı sistemlerinin şu andaki pazarı 70 milyon Dolardır. İlişkisel sistemlerin 3 milyar dolarlık pazarıyla karşılaştırıldığında ihmal edilebilir bir pazar gibi görülmesine karşın, veri tabanı camiasında yaygın görüş bu sistemlerin geleceğin sistemleri olduğudur. Bu meyanda hemen hemen bütün büyük ticari ilişkisel veri tabanı şirketleri, ürünlerinin yeni versiyonlarının nesne-yönelimli olacağını açıklamışlardır. Bununla beraber nesne yönelimli sistemlerin henüz tam olarak olgunlaştığını söyleyemeyiz. Bu nedenle nesne-yönelimli veri tabanları (Object-Oriented DBMSs) konusunda gerek araştırma gerek geliştirme çalışmaları çok yoğun bir şekilde devam etmektedir.

Bu pazara hakim belli başlı şirketlerin oluşturduğu bir grup, ürünlerinde bundan böyle bu standartta uyacaklarını taahhüt ederek, bir standart, ODMG 93, geliştirmişlerdir. ODMG 93 standardına, bu standarda uyacaklarını taahhüt ederek katılan şirketler şöyledir: SunSoft, ObjectDesign, Ontos, O2 Technology, Versant, Objectivity

Bu standarda katılan diğer şirketler şöyledir: Hewlett-Packard, Poet Software, Itasca, Intellitic, Digital Equipment Cooperation, Sevio, Texas Instruments.

Bu standardın belli başlı özellikleri şöyledir:

1. Veri modeli karmaşık nesnelere ilave olarak geri yönde ilişkileri de desteklemektedir. Nesneler için anahtar alan tanımlanabilmektedir.
2. OQL (Object Query Language) adlı SQL benzeri bir sorgu dilini desteklenmekte, ayrıca veri tabanına C++ ile erişim sağlanmaktadır.

Bu standarda rakip bir standartta SQL3 standardıdır. 1200 sayfayı aşan SQL3 standardı nesne yönelimli sistemlerden beklenen bütün özellikleri sağlamaktadır. SQL3 standardını "gatekeeper.dec.com" internet adresinden, /pub/standards/sql dizininden almak mümkündür.

Nesne-yönelimli veri tabanları konusunda yapılmış araştırmalar ile ilgili bir referans listesi ilişikte sunulmuştur. Bu liste daha ziyade akademik yayınları ve ürünler pazara çıkmadan önce yapılmış olan yayınları içermektedir. Ürünler bir kere ticari hale geldikten sonra bu ürünlerin gerçekleştirimini anlatan yayınlar, ticari sır olmaları nedeniyle açıklanmamaktadır.

Pazarda bulunan nesne-yönelimli veri tabanları sistemleri şöyledir:

- Gemstone yazılımı Serviologic (1987) tarafından geliştirilmiş olup, sisteme C, Smalltalk, ve C++ ile ulaşımı sağlamaktadır.
- Ontos yazılımı Ontologic(1989) tarafından geliştirilmiş olup, sisteme C++ ile ulaşımı sağlamaktadır.
- ObjectStore yazılımı ObjectDesign (1990) tarafından geliştirilmiş olup, sisteme C, ve C++ ile ulaşımı sağlamaktadır.
- ObjectivityDB yazılımı Objectivity (1990) tarafından geliştirilmiş olup, sisteme C++ ile ulaşımı sağlamaktadır.
- Versant yazılımı Versant(1990) tarafından geliştirilmiş olup, sisteme C, Smalltalk, ve C++ ile ulaşımı sağlamaktadır.
- O2 yazılımı O2 Technology(1991) tarafından geliştirilmiş olup, sisteme C ve C++ ile ulaşımı sağlamaktadır.
- Open ODB yazılımı HP(1992) tarafından geliştirilmiş olup, sisteme OSQL ile ulaşımı sağlamaktadır.
- Poet yazılımı BKS Software(1992) tarafından geliştirilmiş olup, sisteme C++ ile ulaşımı sağlamaktadır.
- Matisse yazılımı Intellitic(1993) tarafından geliştirilmiş olup, sisteme C++ ile ulaşımı sağlamaktadır.

Araştırma laboratuvarlarında geliştirilen sistemlerden bazıları ise şöyledir: ODE (AT&T Bell), IRIS (HP), Open OODBMS (Texas Inst.), Starburst (IBM), Postgres (Berkeley), Zeitgeist (Texas Inst.), MOODS (TÜBİTAK)

Referanslar:

Books

- [Ber 93] Bertino, E., Martino, L., Object-Oriented Database Systems: Concepts and Architectures, Addison-Wesley, 1993.
- [BDK 92] F. Bancilhon, C. Delobel, and P. Kanellakis, Edtrs., Building an Object-Oriented Database System, Morgan-Kaufmann, 1992.
- [C 94] The Object Database Standard: ODMG-93, Edited by R. Catell, Morgan Kaufmann, 1994.
- [DOBS 94] Doğaç, A., Özsu, T., Biliris, A., Sellis, T., Object-Oriented Database Systems, Springer-Verlag, 1994.
- [K 90] Kim, W., Introduction to Object-Oriented Databases, MIT Press, 1990.
- [K 95] Kim, W., Modern Databases, Addison-Wesley, 1995.

The Data Model and the Query Language

- [AG 89a] Agrawal R. and Gehani N. H., "ODE (Object Database Environment): The Language and the Data Model," Proc. ACM SIGMOD Intl. Conf. on Management of Data, 1989.
- [ALS 89] Alashqur A. M., Su S.Y.W. and Lam H., "OQL: A Query Language for Manipulating Object-Oriented Databases," Proc. of 15th Intl. Conf. Very Large Data Bases, 1989.
- [BZ 87] Bloom, T. and Zdonik, S. B., "Issues in the design of Object-Oriented Database Programming Languages", OOPSLA '87, Oct. 1987.
- [CDV 88] Carey M. J., DeWitt D. J. and Valdenberg S. L., "A Data Model and Query Language for EXODUS," Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1988.
- [K 89] Kim, W. "A Model of Queries for Object-Oriented Databases," Proc. of 15th Intl. Conf. Very Large Data Bases, 1989.
- [KM 90] Kemper, A., and Moerkotte, G., "Access Support in Object Bases", Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1990.
- [LRV 88] Lecluse C., Richard P., and Velez F., "O2, an Object- Oriented Data Model," Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1988.
- [LR 89] Lecluse C., and Richard P., "The O2 Database Programming Language" Proc. of 15th Intl. Conf. Very Large Data Bases, 1989.
- [S 84] Stonebraker M., Anderson A., Hanson E. and Rubenstein B., "QUEL as a Data Type," Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1984.
- [RS 87] Rowe L. and Stonebraker M., "The POSTGRES Data Model," Proc. of 13th Intl. Conf. Very Large Data Bases, 1987.
- [Ber 92] Bertino, E., Foscoli, P., "A Model of Cost Functions for Object-Oriented Queries", In Proc. of 5th International Workshop on Persistent Object Systems, Italy, September 1992.
- [Bla 93] Blakeley, J. A., McKenna, W. J., Graefe, G., "Experiences Building the Open OODB Query Optimizer", Proc. of the ACM SIGMOD Conf., 1993.
- [Gra 93] Graefe, G., McKenna, J. W., "The Volcano Optimizer Generator: Extensibility and Efficient Search", Proc. IEEE Conf. on Data Eng., Vienna Austria, 1993.
- [Ioa 90] Ioannidis, Y., Kang, Y., "Randomized Algorithms for Optimizing Large Join Queries", Proc. of the ACM SIGMOD Conf., 1990.
- [Lan 91] Lanzelotte, R. S. G., Valduriez, P., Ziane, M., Cheiney, J-P., "Optimization of Nonrecursive Queries in OODBs", In Proc. of the Second Intl. Conf. on Deductive and Object-Oriented Databases, 1991.
- [Loh 85] Lohman, G.M. et. al., "Query Processing in R*", Query Processing in Database Systems, Kim, Batory, Reiner, eds. Springer-Verlag, 1985.
- [McK 93] McKenna, W.J., "Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer Generator", Ph. D. Thesis, Univ. of Colorado, 1993.
- [Ono 90] Ono, K., Lohman, G. M., "Measuring the Complexity of Join Enumeration in Query Optimization", Proc. of Intl. Conf. on Very Large Databases, 1990.
- [She 90] Shekita, E. J., Carey, M. J., "A Performance Evaluation of Pointer-Based Joins", Proc. of the ACM SIGMOD Conf., 1990.

Systems in General

- [ABD 94] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, D., Maier, D., Zdonik, S., "The Object-Oriented Database Systems Manifesto" in the book entitled "Building an Object-Oriented Database System" (F. Bancilhon, C. Delobel, and P. Kanellakis, Edtrs.), Morgan-Kaufmann, 1992.
- [M 94] Wesley P. Melling, Gartner Group, "Enterprise Information Architectures - They're finally Changing", ACM Intl. Conf. on Management of Data, Minneapolis, May 1994.
- [V 94] David Vaskevitch, Director of Enterprise Computing, "Microsoft Corporation Database in Crisis and Transition: A Technical Agenda for the Year 2001", ACM Intl. Conf. on Management of Data, Minneapolis, May 1994.
- [B 89] Bretl R., Maier D., Otis A., Penney J., Schuchardt B., Stein J., Williams E. H., and Williams M., "The GemStone Data Management System," Object-Oriented Concepts, Databases, and Applications, W. Kim and F. H. Lochovsky Edtrs., ACM Press, 1989.
- [BK 90] Bancilhon, F. and Kim, W., "Object-Oriented Database Systems: In Transition", ACM SIGMOD Record, Vol.19, No.4, Dec. 1990.
- [BK 90] Bancilhon, F. and Kim, W., "Object-Oriented Database Systems: In Transition", IEEE Data Engineering, Vol.13, No.4, Dec. 1990.
- [BM 91] Bertino, E and Martino, L., "Object-Oriented Database Management Systems: Concepts and Issues", IEEE Computer, April, 1991.
- [C 90] Cattell R. G., "Object Data Management," Tutorial 2, OOPSLA Conference, 1990.
- [CM 84] Copeland G. and Maier D., "Making Smalltalk a Database System," Proc. ACM SIGMOD Intl. Conf. on Management of Data, 1984.
- [F 87] Fishman D.H., Beech D., Cate H.P., Chow E.C., Connors T., Davis J.W., Derrett N., Hoch C.G., Kent W., Lyngbaek P., Mahbod B., Neimat M.A., Ryan T.A., and Shan M.C., "Iris: An Object-Oriented Database Management System," ACM Trans. on Office Information Syst., vol.5 no.1, Jan. 1987.
- [HK 87] Hudson, S.E. and King, R., "Object-Oriented Support for Software Environments", Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1987.
- [K 90] Kim, W., "Object-Oriented Databases: Definition and Research Directions", IEEE Trans. on Knowledge and Data Engineering, Vol.2, No.3, Sept. 1990.
- [KB 89] Kim, W., Ballou, N., Chou, H., and Garza, J. F., "Features of the ORION Object-Oriented Database", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989.
- [KW 87] Kemper, A., Lockemann and Wallrath, M., "An Object-Oriented Database System for Engineering Applications", Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1987.
- [M 89] Maier, D., "Making Database Systems Fast Enough for CAD Applications", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989.
- [P 90] Premerlani W. J., Blaha M.R., Rumbaugh, J. E. and Varwig T. A., "An Object-Oriented Relational Database," Commun. ACM, Vol.33, No.11, Nov. 1990.
- [S 90] Stonebraker M., Rowe L., Lindsay B., Gray J., Carey M. and Beech D., "Third Generation Data Base System Manifesto," Proc. ACM SIGMOD Intl. Conf. on Management of Data, 1990.
- [SR 86] Stonebraker M. and Rowe L. A., "The Design of POSTGRES," Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1986.
- [V 91] Vural, S., "A Survey on Features of Object-Oriented Database Systems", Tech. Rep.

Query Processing

- [AG 89b] Agrawal R. and Gehani N. H., "Design of the Persistence and Query Processing Facilities in O++: The Rationale" IEEE Data Engineering, vol.12, no.3, Sept. 1989.
- [B 86] Batory, D. S., "Extensible Cost Models and Query Optimization in GENESIS", IEEE Data Engineering, Vol.13, No.4, Dec. 1986.
- [BR 86] Bertino, E., Rabitti, F., "Query Processing Based on Complex Object Types", IEEE Data Engineering, Vol.13, No.4, Dec. 1986.

- [GD 87] Graefe, G. and DeWitt, D. J., "The EXODUS Optimizer Generator", Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1987.
- [KM 90] Kemper, A. and Moerkotte, G., "Advanced Query Processing in Object Bases Using Access Support Relations", Proc. of 16th Intl. Conf. Very Large Data Bases, 1990.
- [SZ 89] Shaw, G. and Zdonik, S., "An Object-Oriented Query Algebra" IEEE Data Engineering, vol.12, no.3, Sept. 1989.

Storage Structures and Indexing

- [BK 89] Bertino, E. and Kim, W., "Indexing Techniques for Queries on Nested Objects", IEEE Trans. on Knowledge and Data Engineering, Oct. 1989.
- [CDRS 89] Carey, M.J., DeWitt, J. D., Richardson, J. E., Shekita, E. J., "Storage Management for Objects in EXODUS", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989
- [KKD 89] Kim, W. Kim, K., and Dale, A. "Indexing Techniques for Object-Oriented Databases", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989
- [HO 89] Hafez A. and Ozsoyoglu G., "Storage Structures for Nested Relations" IEEE Data Engineering, vol.12, no.3, Sept. 1989.
- [VBD 89] Velez, F. Bernard, G., and Darnis, V., "The O2 Object Manager: A overview" Proc. of 15th Intl. Conf. Very Large Data Bases, 1989.

Object-Oriented Deductive Databases

- [] Prolog Interface in Smalltalk V.
- [B 88] Ballou, N., et al. "Coupling an Expert System Shell with Object-Oriented Database Systems", Journal of Object-Oriented Programming, Vol.1, No.2, July 1988.
- [CCCTZ] Cacace, F., Ceri, S., Crepsi-Reghizzi, Tanca, L. and Zicari, R., "Integrating Object-Oriented Data Modelling with a Rule-Based Programming Paradigm", Proc. of 1990 ACM SIGMOD Intl. Conf. on Management of Data.
- [DM 89] Diederich, J. and Milton, J., "Objects, Messages, and Rules in Database Design", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989.
- [KW 89] Kifer, M. and Wu, J., "A Logic for Object Oriented Programming", Proc. of ACM PODS, 1989.
- [UZ 90] Ullman, J. D. and Zaniolo, C., "Deductive Databases: Achievements and Future Directions", ACM SIGMOD Record, Vol.19, No.4, Dec. 1990.

Architectural Issues

- [B 87] Bennet, J., "The Design and Implementation of Distributed Smalltalk", Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications, Oct. 1987.
- [D 89] Decouchant D., "A Distributed Object Manager for the Smalltalk-80 System", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989
- [DMFV 90] DeWitt, J. D., Maier, D., and Fattersack, P. and Velez, F., "A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems" Proc. of 16th Intl. Conf. Very Large Data Bases, 1990.
- [MRS 87] Mellender, F. Riegel, S. and Straw, A., "Optimizing Smalltalk Message Performance", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989

Concurrency Control

- [CF 90] Cart, M. and Ferrie, J., "Integrating Concurrency Control into an Object-Oriented Database System", Proc. of EDBT'90, Venice, March 1990.
- [CFR 90] Cart, M., Ferrie, J. and Richy, H., "An Optimistic Concurrency Control for Nested Typed Objects",

Proc. of EDBT'90, Venice, March 1990.

[GK 88] Garza, J.F., and Kim, W. "Transaction Management in an Object-Oriented Database System", ACM SIGMOD Record, Sept. 1988.

[H 90] Herlihy, M., "Apologizing Versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types", ACM Trans. on Database Syst., Vol.15, No.1, March 1990.

[HK 89] Hudson, S. E., and King, R., "Cactis: A Self-Adaptive Concurrent Implementation of an Object-Oriented Database Management System", ACM Trans. on Database Syst., Vol.13, No.3, Sept. 1989.

[W 91] Wiczerzycki, W., "Concurrency Control for Multiversion Composite Objects", Proc. of ISCIS VI, 1991.

[SZ 90] Skarra, A. H. and Zdonik, S.B., "Concurrency Control and Object-Oriented Databases", Object-Oriented Concepts, Databases, and Applications, Edited by Kim, W. and Lochovsky, F. H., ACM Press, 1989

Pictorial Databases

[C 90] Cardenas, A. F., "Pictorial/Visual access to Multimedia/heterogenous Databases" IEEE Data Engineering, Vol.13, No.2, June 1990.

[CIK 88] Cheng, Y. Iyengar, S.S. and Kashyap, R. L., "A New Method of Image Compression Using Irreducible Covers of Maximal Rectangles", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[CYDA 88] Chang, S. K., Yan, C. W., Dimitrof, D. C. and Arndt, T., "An Intelligent Image Database System", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[GHS 89] Goodman A. M., Haralick, R.M., and Shapiro, L. G., "Knowledge-Based Computer Vision", IEEE Computer, Dec. 1989.

[JC 88] Joseph, T. and Cardenas, A. F., "PICQUERY: A High Level Query Language for Pictorial Database Management", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[JO 89] Jagadish, H.V. and O'Gorman, L., "An Object Model for Image Recognition", IEEE Computer Dec. 1989.

[KA 88] Kasturi, R. and Alemany, J., "Information Extraction from Images of Paper-Based Maps", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[MK 88] Mohan, L. and Kashyap, R. L., "An Object-Oriented Knowledge Representation for Spatial Information", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[OM 88] Orenstein, J. A and Manola, F. A., "PROBE Spatial Data Modeling and Query Processing in an Image Database Application", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[RFS 88] Roussopoulos, N. Faloutsos, C. and Sellis, T., "An Efficient Pictorial Database System for PSQL", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[TI 88] Tanaka, M. and Ichikawa, T., "A Visual User Interface for Map Information Retrieval Based on Semantic Significance", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

[USV 88] Unnikrishnan, A. Shankar, P. and Venkatesh, Y.V., "Threaded Linear Hierarchical Quadrees for Computation of Geometric Properties of Binary Images", IEEE Trans. on Software Eng., Vol. 14, No.5, May 1988.

CAD Databases

[BKK 85] Bancilhon, F., Kim, W. and Korth, H., "A Model of CAD Transactions" Proc. of 11th Intl. Conf. Very Large Data Bases, 1985.

[BK 85] Batory, D., and Kim, W., "Modeling Concepts VLSI CAD Objects", ACM Trans. on Database Syst., Vol.10, No.3, Sept. 1985.

[KIM 84] Kim, W., McNabb, D., Lorie, R. and Plouffe, W., "A Transaction Mechanism for Engineering Design databases", Proc. of 10th Intl. Conf. Very Large Data Bases, 1984.

Proje No: Yazılım 1	2- Rapor Tarihi: 30 Eylül 1994
Projenin Başlangıç ve Bitiş Tarihleri: 1 Ekim 1991 - 30 Eylül 1994	
Projenin Adı: Nesnesel Veri Tabanı Yönetim Sistemi Prototipi	
Proje Yürütücüsü ve Yardımcı Araştırmacılar: Prof.Dr. Asuman Doğan, Çetin Ozkan, Cem Evrendilek, Mehmet Altınal, Budak Arpınar ve MS öğrencileri.	
Projenin Yürütüldüğü Kuruluş ve Adresi: TUBİTAK Yazılım Araştırma ve Geliştirme Enstitüsü, Bilgisayar Müh.Bölümü, ODTÜ, 06531, Ankara	
Destekleyen Kuruluş(ların) Adı ve Adresi: ----	
Öz (Abstract): Nesne-yönelimli yaklaşımı kullanan MOOD sisteminin sağladığı başlıca teknik olanaklar şöyledir: - Grafik, fotoğraf, ses gibi çoklu ortam verisi de içerebilen karmaşık veri yapılarının temsil ve etkin bir şekilde kullanma olanağı - Bu verilere grafik ortamda erişimi sağlayan ve Motif yazılımı kullanılarak geliştirilmiş bir grafik kullanıcı arayüzü (Graphical User Interface), MoodView - Yine veri erişimi ve güncelleme amacıyla kullanılan bir nesne yönelimli SQL dili, MoodSQL - C++ dili ile tanımlanmış olan fonksiyonların, SQL dili içerisinde dinamik olarak çağırılmasını sağlayan bir dinamik fonksiyon bağlayıcısı (Dynamic Function Linker) - SQL sorgularının sistemde kullandığı kaynakları en aza indirmek amacıyla geliştirilmiş bir SQL sorgu en iyileştirici (Query Optimizer). Bu sistem Colorado Üniversitesinde geliştirilmiş bulunan Volcano Query Optimizer generatör yazılımı kullanılarak geliştirilmiştir. - Sistem kataloglarını yöneten birim, MOOD Catalog Manager - Veri erişim komutlarını disk yönetim sistemi komutlarına dönüştüren MOOD Algebra - Çok boyutlu verilere etkin erişimi sağlamak amacıyla, R ve R* tree ve ikonik indeksleme yöntemleri - Üstün bu sistemler Wisconsin-Madison Üniversitesinde geliştirilmiş bulunan Exodus Storage Manager isimli disk yönetimi yapan sistem üzerine geliştirilmiştir. Projenin ilk safhalarında ayrıca Borland C++ "persistent" hale getirilmiştir Ahtar Kelimeler: Nesne yönelimli veri tabanı yönetim sistemleri, sorgu eniyileme.	
Proje ile ilgili Yayın/Tebliğlerle ilgili Bilgiler: Proje ile ilgili 1 kitap, 1 makale, 11 yayın ve 13 Yüksek Lisans tezi gerçekleştirilmiştir.	
Bilim Dalı: Bilgisayar	ISIC Kodu:
Doçentlik B. Dalı Kodu:	
Uzmanlık Alanı Kodu: 619.02.02	
Dağıtım (*): <input type="checkbox"/> Sınırlı <input checked="" type="checkbox"/> Sınırsız	
Raporun Gizlilik Durumu: <input type="checkbox"/> Gizli <input checked="" type="checkbox"/> Gizli Değil	

Projenin Sonuç Raporunun ulaştırılmasını istediğiniz kurum ve kuruluşları ayrıca belirtiniz