The MOOD User Manual

Edited by Asuman Dogac
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

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., edtrs., 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 Graphical 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.

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.

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.

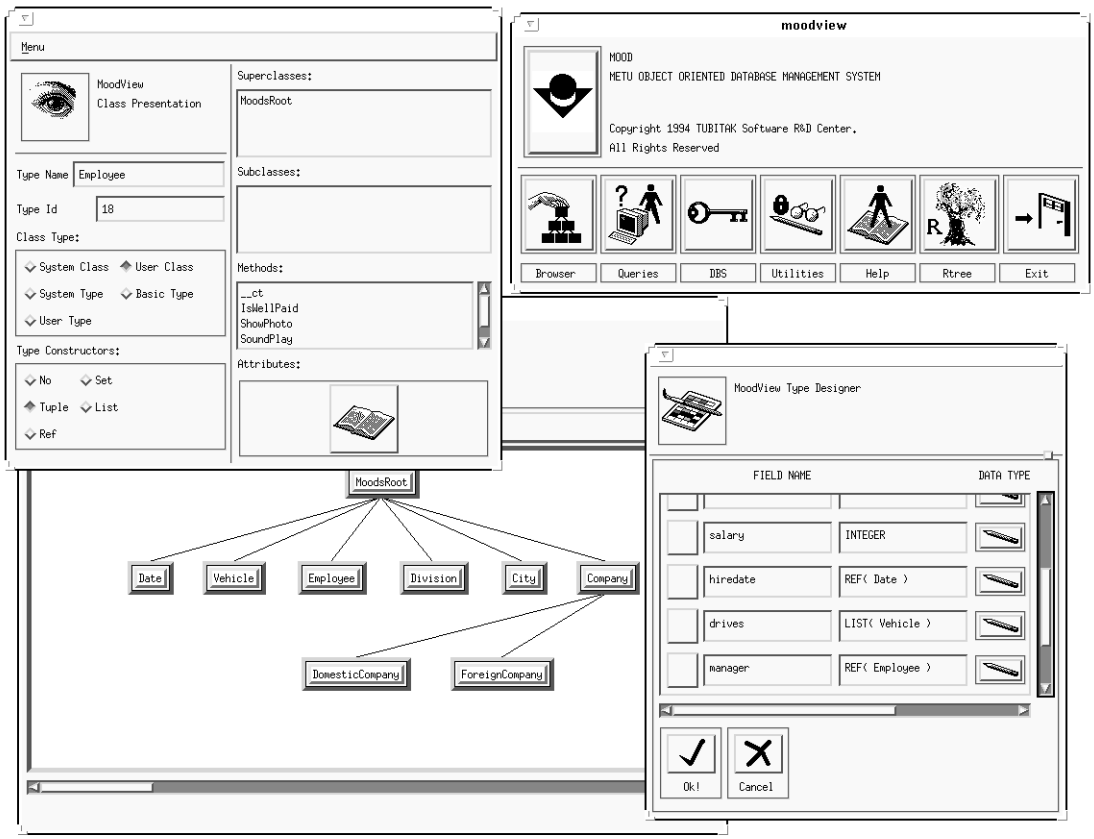

Figure 1. Initial MoodView Window, Class Hierarchy Browser, and Class Presentation

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.

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.

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 1 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.

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 1).

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.

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 a 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. Updates to existing method bodies or creation of the new method bodies can be done through MoodView text editor.

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.

Figure 2. Query Manager and Objects Displayed As A Result of the Specific Query

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

Figure 3. Overall MOOD Structure

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.

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 3. During this interpretation, functions (which have been previously compiled with C++) within SQL statements, are dynamically linked and executed.

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 droping 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 inheret 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 [ASC|DESC], ..., attribute [ASC|DESC]
```

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.

Below two example database schema definitions are provided which will be used in the example queries.

Example Schema Definition 1:

```
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
)
};
```

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

```
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]
)
};
```

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}, se-
lect d from City d where d.name='Ankara');
new Division('development',{select e from Employee e where e.ssno=3}, se-
lect 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','Holywood',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;

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

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

4.3 Example Queries in MOODSQL

The following examples use the example schema 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
==========
Asuman 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 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 Sevuk 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 !

4.5 Deleting User Defined Functions

MoodSQL>ls fnc of Employee;

FUNCTIONS
==========
__ct__8EmployeeFiPcT18REF_Date
Well_Paid_Emp__8EmployeeFi

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

MoodSQL>ls fnc of Employee;

FUNCTIONS
==========
__ct__8EmployeeFiPcT18REF_Date

4.6 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 RESULT
==========
TUBITAK
GM

2 Objects are listed !

4.7 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
certan
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!
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@srdc.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 V.1.0

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 Mood-View 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)

Avaliable 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 necessay 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.
#
[sf]*[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.
#
[sf]*[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.
#
[sf]*[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
```