UML Modeling of Embedded Database through C++

Dr. Vipin Saxena[†] and Deepak Arora[††],

Department of Computer Science, B.B. Ambedkar University (A Central University), Vidya Vihar Rae Bareli Road, Lucknow U.P. 226025, India

Summary
Today in the market we find large number of database applications designed according to the need of individual user. A database needs to be fast and reliable, besides that it should be compact and portable too. Nowadays devices require some new kind of databases which consume less amount of memory and can be installed easily on devices such as on mobile phones, PDAs, set-top boxes and automotives, etc. It should be design to use available resources, in an efficient way. These requirements lead the developers towards the concept of Embedded Database. In this paper authors have given the design of Embedded Database through a well known modeling language i.e. Unified Modeling Language (UML). Authors have also given a case study of development of prototype for Embedded Database Management System, implemented using C++ programming language. Authors have tested the performance of developed prototype and also reported the test result in this paper.

Key words:
UML class diagram, UML sequence diagram, Embedded Database, C++

1. Introduction

The Embedded Database is installed as a software component, with any application and that application can invoke all the operation on Embedded Database. The embedding of an Embedded Database can be done within an application either by writing in-line code or by providing linked libraries along with, unlike the traditional general-purpose enterprise relational databases, which normally run as the separate and independent applications such as Oracle, DB2, and SQL Server etc. Another important aspect of Embedded Database is to free users and administrators from time-consuming installations and maintenance work, as the Embedded Databases are easy to install and maintain.

There is lot of research work on Unified Modeling Language (UML) has been done related to the various database technologies, few of them regarding Embedded Database development, developed by Booch [1], is the most suitable visual presentation platform for modeling the real word problems. OMG [2, 3] describe the latest UML specifications related to real world, modeling aspects, the way of representation for XML metadata specification in UML diagrams and the standard storage representations through the Unified Modeling Language. Pllana and Fahringer [4] suggested UML profiles, through which one can model the high performance oriented applications. Chen, R. et al. [5] defines a new UML profile called UML platform, through which one can model the embedded system platform. They have also proposed a set of new building blocks, stereotypes for the representation of specific platform and its services. Martin, G. [6] has defined the nature of the embedded systems and real time embedded system design methodology requirements. Also suggested the important extensions required in UML, so that one can better understand the hidden issues about real time embedded system design. Kukkala, P. et al. [7] have introduced a new UML 2.0 profile called TUT-Profile, having a set of stereotypes, new platform components and design rules for embedded system design. Through TUT-Profile, one can establish the mapping between application and platform description for the system. McUmber, W.E. et al. [8], developed a framework to define the VHDL specifications, which is IEEE adopted language used for describing digital electronic hardware system, with the help of class and state diagrams. They have established the homomorphism between VHDL and UML diagrams. This makes us capable, to understand the behavioural aspect of any embedded systems. Lu, S. et al. [9] has combined the Model Driven Architecture with UML models for developing the software for embedded real time systems. They also performed mapping from platform independent model to platform specific model of an embedded real time systems. Wehrmeister, M. A. et al. [10] introduced the platform-based object oriented design methodology, through which one can easily understand the design aspect of any embedded real time system. One more important reference, regarding mobile Embedded Database is from Zhang, X. et al. [11]. They have proposed a mobile Embedded Database along with its three-layer architecture, in which the top layer is for embedded mobile database, middle one is for synchronization server and the bottom is for background multi-user database system server.

One of the important performance criteria of any Embedded Database Management System is the access time for manipulation of data. Since Embedded Database Management System consists of a very large number of

Manuscript received July 5, 2008.
Manuscript revised July 20, 2008.

complex utilities, it is extremely difficult to build it using assembly language, through which, one can achieve fast response time but it would cause a system to be very much machine dependent. For attaining a good speed and at the same time insuring portability, C++ programming language is selected by the authors for developing such a system. Well-written C++ program tends to match assembly language in terms of speed of obviating the need for going into lower languages and additionally portability is also ensured. In this paper authors have used the concept of linked libraries which provide all features of database. By using it any user can easily embed the coding written for accessing the Embedded Database, in any C++ program. Authors have given the UML Use Case, Class Diagram and Sequence Diagram and discuss a case study of a prototype development of Embedded Database Management System. Authors have tested the query performance of this prototype and test results are mentioned in the paper.

2. Background

The perception of front end and back end is very common to the existing database technologies, is also used in the proposed Embedded Database Management System. In this programmer or the user is provided with a data definition writing tool that helps programmer in performing complete database operations like writing database definitions, table structures, indexes and record insertion. Also the user is provided some linked libraries along with, through which the embedding of code can be done in a normal programming language, for accessing and manipulating the database, created with the help of aforesaid tool.

2.1 Objective of Proposed Embedded Database

The objective of the proposed Embedded Database is to provide a convenient and effective method of defining and retrieving the information contained in database, by providing libraries of functions, which can be used to manipulate the data in an efficient manner. The objectives of proposed Embedded Database can be more magnify by following points:

- To provide a user interface (Database Management Utility) for creation of databases, tables, indexes and handle the constraints applied on tables.
- To provide a user interface for record insertion in table.
- To provide proper management of indexes.
- To provide a mechanism for deletion of databases, tables.
- To provide a mechanism for viewing records modification of table structures.
- To provide a mechanism for creation of user defined data-types.
- To provide password management.
- To provide database backup & recovery.

2.2 Features of Proposed Embedded Database

- Data independence allows dynamic changes and growth potential.
- Data duplication elimination with controlled redundancy.
- Enhanced data accessing.
- Security enforcement.

2.3 Advantages of Proposed Embedded Database

- The Embedded Database needs no any third party database tool for manipulations regarding data.
- For accessing data there is no need of any driver for establishing compatibility between database (back end) and the program (front end) written for accessing the database.
- Embedding can be done in any normal C++ program, gives the portability aspect with less consumption of memory, facilitates it with the efficient data fetching along with data security.

A programmer can include the libraries provided with Embedded Database in a normal fashion as programmer includes other header file in his program. Inclusion of the header file doesn't mean that user is capable to operate any type of operation on any table which are included in this database, besides that user has to establish the connection to the Embedded Database by calling the connect function along with a valid authentication identity. User can access only those tables, which are contained in the included database, not others. To perform any type of operation on the particular table, the table has to be opened first.

After opening the table the user will get all the rights to take the full advantages of Embedded Database features provide by the database and the libraries. Now user can easily get all the structure variables defined at the time of the creation of the objects like tables in the database using data definition tool.

3. UML Modeling

The considered key design issues related to proposed Embedded Database through UML are as follows,

3.1 Backend Support

The data definition tool or database management utility is a menu-based program, in which there is no need to write any complicated commands and syntaxes. Selection of particular choice is enough to accomplish the specified operation. Also the data definition tool will be protected by the encrypted password security. The tool is capable to open one database at a time but at the same time it can access more than one table, exists in that opened database.

Database can be created after selecting the "New Database" option from the database window. Once the database is created, any one can easily create the tables in this database. The system will also ensure that no user can access the tables or other objects in the database without giving the valid authentication identity, provides a completely secured access to the Embedded Database.

Index can be created for one or more than field for any table and for this purpose, a separate file having ".idx" extension has to be created. Structure of any particular table and available records can be viewed very easily by selecting the table name. Creation of user defined data type can also be possible. Moreover the indexes are self-updated at the time of database start-up.

3.2 Embedding of Code

The embedding of code for accessing the Embedded Database through any program can be done as follows:

- To access the structure variable "name" of table "employee" of "payroll" database programmer can write the command like,

  ```
  cout<<payroll.employee.RecordSet.name;
  or
  strcpy(emp_name,payroll.employee.RecordSet.name);
  cout<<emp_name;
  ```
  Here emp_name can be any local variable in the program.

- To save the record in any particular table user has to place the required value in the available structure variables by using commands like above, and to save the record, user has to call the inbuilt function like,

  ```
  strcpy(payroll.employee.RecordSet.name,emp_name);
  payroll.employee.saveRecord();
  ```

- To traverse all records in any table, programmer can use simple commands, are given below,

  ```
  payroll.employee.movefirst()
  while !payroll.employee.eof()
  {
  cout<<payroll.employee.RecordSet.name;
  payroll.employee.moveNext();
   }
  ```

- If user opens the index file of any table then database provides the whole record set in the indexed order according to specified field in the table.
- If user wants to delete any table from the database one can write the command like,

  ```
  payroll.dropTable("employee");
  ```

3.1 Data Integrity

Constraints can be defined on data in a table to ensure the integrity by means of indexes. Database integrity can be enforced, by specifying the constraints at the time of table creation using data definition tool. These constraints are called declarative Embedded Database integrity constrains, are given below:

I. Unique constraints

Ensures uniqueness that no two rows have the same values in the same specified column(s).

II. Primary Key

Avoids duplication of row and does not allow Null values, when enforced on a table.

III. Null Values

Ensures that Embedded Database table can't accept blank value in the specified field.

IV. Default Value

A default value can be specified for a column in the table.

V. Check Constraints

It specifies the data range that can be inserted into a column of a table.

The class diagram for the Table, Database and Recordset_Base are given below in Fig. 1(a), Fig. 1(b) and Fig. 1 (c) respectively.

Table
list: structure tb: structure tb_struct: structure indexlist: structure recordLength: integer recordCount: integer indexfieldno: Integer indexfieldsize: integer
void clear() void addrecord() void create() void displayRecord() void desc() void DeleteRecord() void embeded_Class_Structure() void embeded_Database_Structure() void fieldForIndex() void fetchFreshRecords() int findCode() int findCode1(int checkempty) void fetchRecord() void indexFetchRecord() void indexEntry(char *str) void indexEntry(int str) void indexEntry(float str) void indexEntry(double str) void indexEntry(long str) void listtable() void listdata() void makeIndex() void modify() void newTable() void printheads() int primaryKeyCheck(char *str) int primaryKeyCheck(int str) int primaryKeyCheck(float str) int primaryKeyCheck(long str) int primaryKeyCheck(double str) int rangeCheck(int str) int rangeCheck(float str) int rangeCheck(long str) int rangeCheck(double str) void structurescreen() void sortIndexlist() void tableOpen() void updateIndex() void writeFreshRecords()

Fig. 1(a) Class diagram for Table

Databse
db: structure fp: pointer
int open() void newDatabase() void listdata() void changePasswd() int dbaOpen() int dbapass()

Fig. 1(b) Class diagram for Database

Recordset_Base
file: pointer tab: pointer
int recordLength; int recordCount; int recordNumber; int bof; int eof; int Error; int fileCheck(char *fileName) int open(char *fileName) int saveRecord() int refreshRecord() int moveLast() int movePrevious() int moveNext()

Fig. 1(c) Class diagram for Recordset_Base

Table class is responsible not only for the management of records and indexes but also for generating embedded class definitions, which include all the databases and table definitions defined by the user created with the help of data definition tool. This will help to implement the embedding feature of Embedded Database, in any normal C++ program, through which one can embed the code for accessing the data from Embedded Database, within a normal programming.

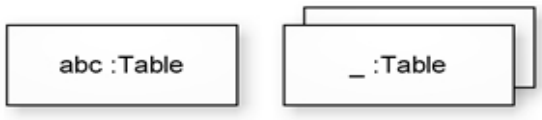

Fig. 2 Single and multiple instance of Table class

Embedded Database can open multiple tables at a time. Single and multiple instance of table class are shown in Fig. 2. Here Recordset_Base class is responsible for providing all the necessary library functions through which a programmer or user can manipulate the Embedded Database records.

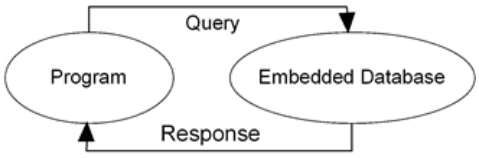

Fig. 3 Communication between program and embedded database

In Fig. 3, one can see that there is one-to-one correspondence between the program and Embedded Database. Code embedded for accessing the records, when

executed, submits the query to the Embedded Database and accordingly Embedded Database responses. Screen class is shown in Fig. 4, which is responsible for managing all the display activities required by the data definition tool like display of window (DOS based) creation, menu creation and all the basic controls like combo box, list box and scroll bars related operations.

Screen
black: Screen_Capture
void startDatabase() void say(char *ms,int rw,int cl) void accept(char *str,int row,int col,int show,int textback) int menu(char tx[][40],int noe,int rw,int cl,int defch=1) int radiobutton(char tx[][40],int noe,int txtcolor,int txtback,int pointcolor,int rw,int cl,int defch=1) int button(int r,int c,char *ch,int windowbkcolor,int buttonbkcolor,int flag) void scrollbar(int rw,int cl,int noe,int pos) int menusp(char tx[][40],int noe,int rw,int cl,int size=-1) int window(int row1,int col1,int row2,int col2,int windowstyle,int shade,int selection,int noe,char array[][40],int bkcolor,int bordercolor,int headbkcolor,int headfrcolor,char *headmsg,int insidemsgcolor,char *insidemsg,int ftbkcolor,int ftfrcolor,char *ftmsg,char *ftp,char *str,int textbkcolor,int windowflag,int buttoncolor=RED) void error(int erno) char *correct(char *str) int combobox(char tx[][40],int noe,int rw,int cl) void acceptpro(char *str,int row,int col) char *rec_accept(int row,int col,int field_type,int field_size,char allownull) void mainMenu(int rw,int col,int noe) void box(char bc,int r1,int c1,int r2,int c2) void coverscreen() void heading() int calender() void ascii()

Fig. 4 Class diagram for Screen

Screen_Capture
buffer: character
void saveScreen() void restoreScreen()

Fig. 5 Class Diagram for Screen_Capture

Screen_Capture class shown in Fig. 5 will incorporate the saving and restoring the graphical sessions to avoid flickering during the change of subsequent screen. Here Screen class is managing all the display activities in text mode not in graphical mode, designed for less memory requirements.

4. A Case Study: Design of Proposed Embedded Database System

For showing the issues related to the design of proposed Embedded Database, authors have given a case study of development of an Embedded Database prototype using C++ programming language. The UML use case diagram for the developed prototype is given in Fig. 6. There are three actors exist in the system, DBA, USER and C++ Program. One can see the use case diagram that the use case "Create Table" further includes the other use cases like Create Structure, Create Index and Create Embedded Table, Create Embedded Database class definitions.

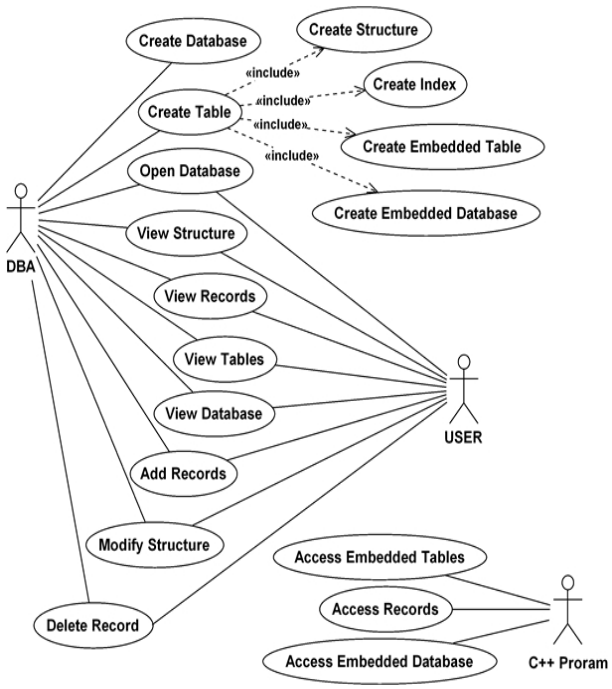

Fig. 6 Use case diagram for embedded database prototype

This will complete the create table use case executed by DBA. DBA has right to perform all the cases of USER, whereas the USER is abstained by some cases, which

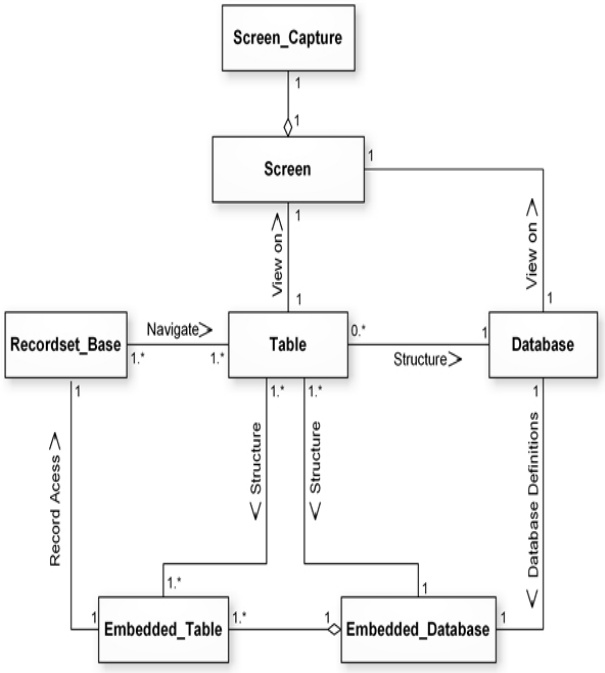

Fig. 7 Class diagram for embedded database

includes data definition like activities like Database and Table creation, can be performed by DBA only. The actor

C++ Program is designed to act on Access Embedded Database, Access Embedded Tables and Access Records, like use cases.

In this, one can say that this system represents two levels for accessing the database and its functionality. DBA is responsible for creating database and tables while user not, except that user is entitle to access and manipulate the records in the tables as needed.

The class diagram for Embedded Database is given in Fig. 7. Here Database class is responsible for the creation of database and giving access to the database created. More than one table can be created in a database and Database class will interact with the Table class for accessing the table's structure information and records contained. Table class is responsible for the creation of indexes, maintaining table structure information, writing embedded class definitions for the databases and tables created.

The generated embedded class definitions, Embedded_Table and Embedded_database will facilitate the creation of record set and accessing the records from the needed table within any C++ program. Also the Recordset_Base class will interact with the Table class, so that the records can be accessed and navigated in an efficient way. The Screen class will interact with table and database class directly for providing graphical user interface to the data definition tool, through which user or DBA can perform all the necessary operation on the

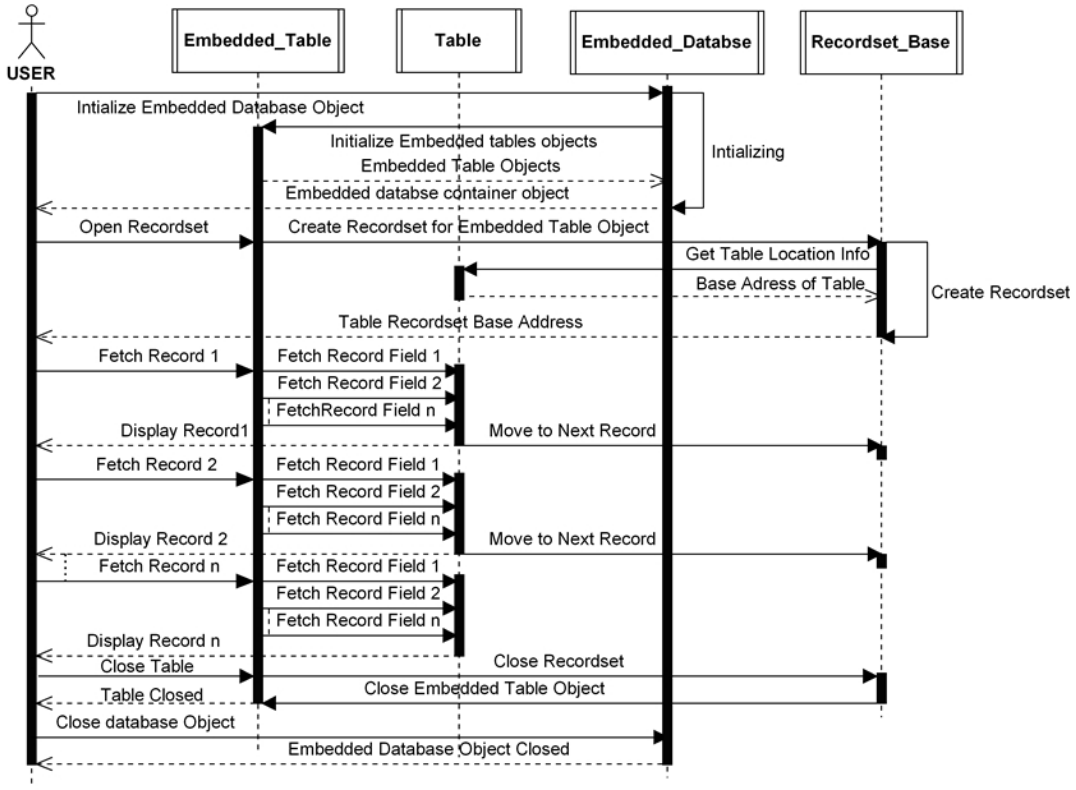

Fig. 8 Sequence diagram of embedded database

Embedded Database.

The sequence diagram represents the scenario for accessing the Embedded Database through C++ program, which is shown below in Fig. 8. In this diagram one can see that how the objects are getting communicated and what are the messages are passed during the execution of C++ program for accessing the Embedded Database, for which the embedding was done. One can initialize the Embedded_Database object created by the system, acts as container object for the table objects, contained in it. The initialization of Embedded_Database object will cause the container Embedded_Table objects initialization too. When initialization of Embedded_Database completes, then record set creation will start.

The Recordset_Base object will retrieve the records from Table object in to the structure specified by the Embedded_Table definition. The created record set base object will be assigned to the caller C++ program along with end of file (EOF) and beginning of file (BOF) marker. Now fetching of records from Embedded_Database will start. From calling move next operation, one can easily navigate the complete record set and access all the records. Embedded_Table can be closed, by passing close message to Embedded_Table object, and now a close record set message will also send to the record set base by Embedded_Table object. This will close the Embedded_Table object. When C++ program finishes, it will send the close message to the Embedded_Database.

5. Queries and Performance Evaluation

The above design of Embedded Database system is easy to use and portable. The developed prototype supports the conventional method of accessing the records from any database management system. It consumes less efforts of writing code and gives higher degree of easiness to the programmers and its users. The code written for accessing the records is more understandable and consumes less record fetching cycles.

In the Fig. 9, a sample code segment shows that how one can embed the system generated class definitions for accessing the desired Embedded Database, through a normal C++ program. Here authors have taken a sample database named “demo”, created with data definition tool by the DBA. This “demo” database contains a table “employee”, which has some sample records entered by the user. The creation of table “employee” and sample records are inserted with the help of data definition tool performed by the DBA and USER respectively. This code is written for printing of all the records exist in “employee” table one by one. To access and navigate the records contained in the table, one must have inclusion of a library of record set related functions, called “dbutil.edc” in the C++ program, which contains the functions like open Table(), moveNext(), movePrev(), closeTable() etc. The code segment depicted in Fig. 9 also has an inclusion of the file “demo.db”, which is a system generated file, contains all the embedded class definitions related to the “employee” table and an object of the container database “demo”, shown in Fig. 10. The system generated

```
#include"dbutil.edc"
#include"demo.db"
#include<conio.h>
void main()
{
demo.openTable("employee");
while(demo.employee.eof==FALSE)
{
  clrscr();
  cout<<"Detail of Employees \n";
  cout<<"--------------------\n";
  cout<<"\n\n\n";
  cout<<"Total Records : "<<demo.employee.recordCount<<endl;
  cout<<"================================================\n";
  cout<<"Record Number       : "<<demo.employee.recordNumber<<endl;
  cout<<"Employee Code       : "<<demo.employee.RecordSet.emp_code<<endl;
  cout<<"Employee Name       : "<<demo.employee.RecordSet.emp_name<<endl;
  cout<<"Employee Address  : "<<demo.employee.RecordSet.emp_address<<endl;
  cout<<"Employee Salary     : "<<demo.employee.RecordSet.salary<<endl;
  cout<<"PF Deduction          : "<<demo.employee.RecordSet.pf<<endl;
  cout<<"\n\n\n";
  cout<<"Press any key to resume .....";
  getch();
  demo.employee.moveNext();
}
demo.closeTable("employee");
getch();
}
```

Fig. 9 Code segment shows the implementation of embedding feature

embedded class definitions written in "demo.db", is responsible for giving access to the database and tables contained in it, as desired by the user. The output generated from the execution of code segment given in Fig. 9, is shown in Fig. 11.

```
struct s001 { int isDeleted;char emp_code [6];char
emp_name [30];char emp_address [40];float salary;
int pf;  };
class t001 :public RecordSetBase
{
  public:
      s001 RecordSet;
 void open(char *fileName)
   {
   RecordSetBase::tab=&RecordSet;
   RecordSetBase::recordLength=sizeof(RecordSet);
   RecordSetBase::open(fileName);
   }
};
class DB001
{
  public:
      t001 employee;
 void openTable(char *file)
 {
   if(strcmp(file,"employee")==0)
      employee.open("employee");
 }
 void closeTable(char *file)
  {
   if(strcmp(file,"employee")==0)
   employee.close();
  }

};
DB001 demo;
```

Fig. 10 System generated embedded class definitions

```
Detail of Employees
-------------------


Total Records : 5
=========================================
Record Number      : 1
Employee Code      : E001
Employee Name      : Deepak Arora
Employee Address   : L.D.A., Lucknow
Employee Salary    : 25000
PF Deduction       : 12


Press any key to resume .............
```

Fig. 11 Output of code segment given in Fig. 9

The prototype is developed, using C++ programming language and is based on binary file system. There is no graphical initialization used in the system, in spite of simple text based graphical user interface is provided to the users, result into faster response by the side of database. The Databases, Tables, Indexes, Table Structures and embedded class definitions, all uses binary file system, which provides the reliability and security and fast record accessibility like features to the system.

The Authors have also evaluated the performance of Embedded Database prototype and found it, satisfactory. Fig. 12 shows the test results and authors have measured the performance by assigning various tasks and calculate the time interval between the submission and competition of that task to the system. Testing is done on a low configuration node, and the performance will be higher on a high configuration node.

Also the security is a very important aspect for any Embedded Database Management System. System is equipped with encrypted password securities for the DBA as well as its users. In case of improper shut down, the Embedded Database Management System auto recovers and updates the required items.

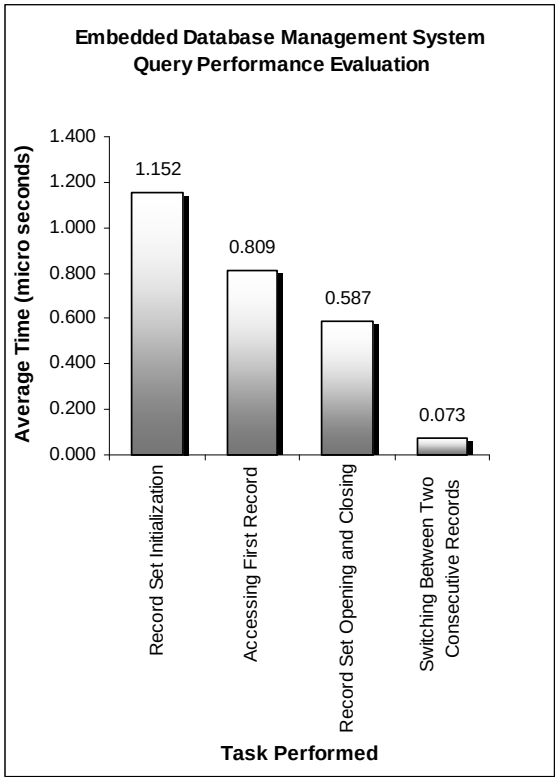

Fig. 12 Query performance evaluation of embedded database prototype

Following are few output screens of Database Management Utility, of developed prototype of Embedded Database prototype, shown from Fig. 13 to Fig. 20:

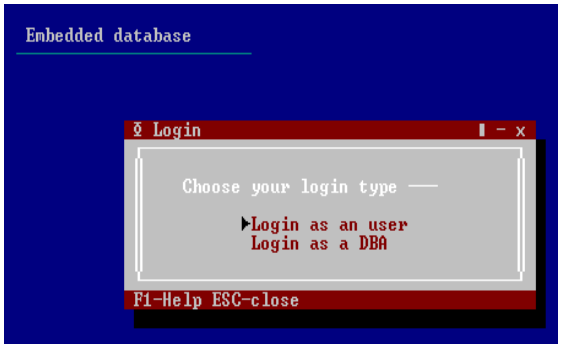

Fig. 13 Login window

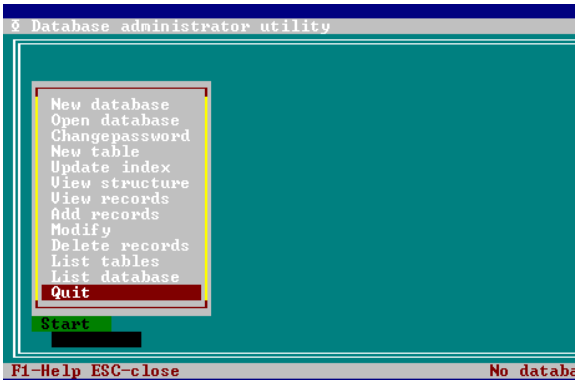

Fig. 14 Task menu

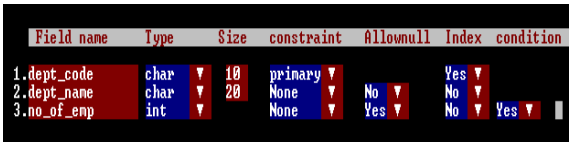

Fig. 15 Table structure creation utility

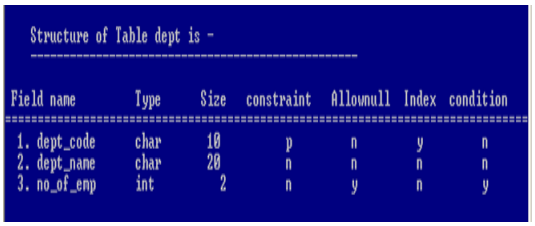

Fig. 16 Table structure view utility

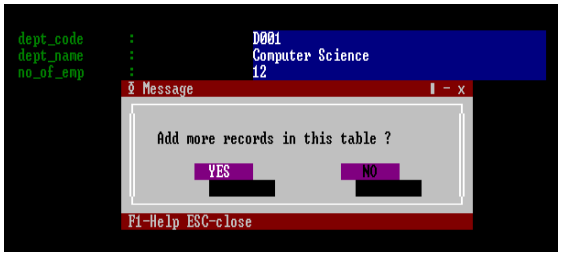

Fig. 17 Record insertion utility

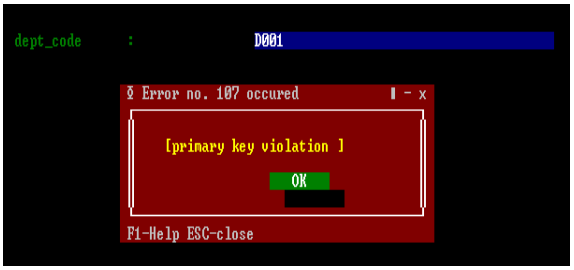

Fig. 18 Constraint violation message

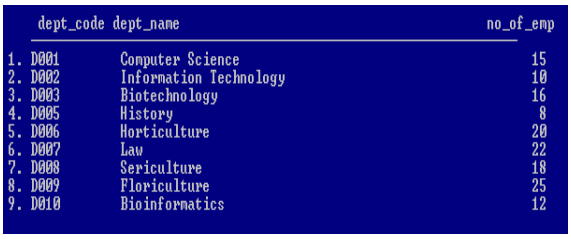

Fig. 19 Record-viewing utility

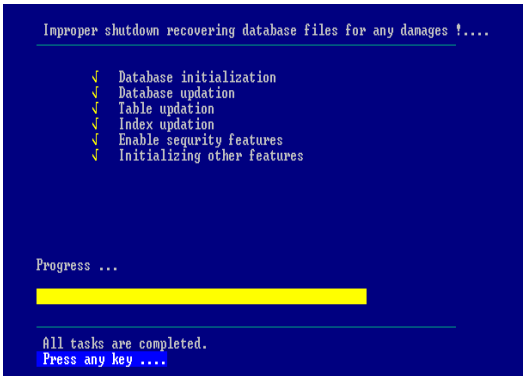

Fig. 20 Database recovery

The above system is designed with the help of well adopted visual modeling language used for modeling real word problems i.e. Unified Modeling Language. As the UML has adopted bottom up approach, supports faster and reliable application development as comparer to the earlier approaches based on top down approach. The system is developed by using C++ programming language, gives it high memory utilization, less space complexity and greater efficiency over other approaches.

6. Concluding Remarks

From the above it is concluded that the designed embedded prototype database system is a very convenient and effective system for receiving the information contained in the database. The importance of Embedded Database system is defined in the paper. The designed model by the use of UML and then implemented the model through C++ is a security protected system and

performance of the system is evaluated by considering the numerous queries. This work can be further extended for the development of Embedded Database for the real time system through the UML modeling.

References

[1] Booch, G., Rumbaugh, J., Jacobson, I., 1999, The Unified Modeling Language User Guide, Addison Wesley, Reading, MA 1999.

[2] OMG Unified Modeling Language Specification, 2001. Available online via http://www.omg.org.

[3] OMG XML Metadata Interchange (XMI) Specification, 2002. Available online via http://www.omg.org.

[4] Pllana, S. and T. Fahringer, 2002, On Customizing the UML for Modeling Performance Oriented Applications. In <<UML>>, Model Engineering Concepts and Tools, Springer-Verlag., Dresden, Germany, 2002.

[5] Chen, R., Sgroi, M., Edmund Martin, G., Lavagno, L., Sangiovanni-Vincentelli, A. and Rabaey, J., 2002, Embedded System Design Using UML and Platforms, Proceedings of Forum on Specification and Design Languages (FDL'02), 2002.

[6] Martin, G., 2002, UML for embedded systems specification and design: motivation and overview, IEEE Proceedings, Design, Automation and Test in Europe Conference and Exhibition, 2002, pp. 773-775.

[7] Kukkala, P., Riihimaki, J., Hannikainen, M., Hamalainen, T.D., Kronlof, K., 2005, UML 2.0 profile for embedded system design, IEEE Proceedings, Design, Automation and Test in Europe, Vol. 2, 2005, pp. 710-715.

[8] McUmber, W.E., and Cheng, B.H.C., 1999, UML-based analysis of embedded systems using a mapping to VHDL, IEEE Proceedings, High-Assurance Systems Engineering, 1999, pp. 56-63.

[9] Shourong Lu, Halang, W.A., Lichen Zhang, 2005, A component-based UML profile to model embedded real-time systems designed by the MDA approach, IEEE, Embedded and Real-Time Computing Systems and Applications, 2005, pp. 563-566.

[10] Wehrmeister, M.A., Becker, L.B., Wagner, F.R. and Pereira C.E, 2005, An object-oriented platform-based design process for embedded real-time systems, Eighth IEEE Proceedings, Object-Oriented Real-Time Distributed Computing, 2005. Volume, Issue, 18-20, May 2005, pp. 125–128.

[11] Zhang, X., Meng, X. and Wang, S., 2000, KingBase Lite: a smart mobile embedded database system, IEEE Proceedings, High Performance Computing in the Asia-Pacific Region Volume 2, 2000, pp. 806–811.

Dr. Vipin Saxena: He is a Reader & Head, Dept. of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India. He got his M.Phil. Degree in Computer Application in 1991 & Ph.D. Degree work on Scientific Computing from University of Roorkee (renamed as Indian Institute of Technology, India) in 1997. He has more than 12 years teaching experience and 16 years research experience in the field of Scientific Computing & Software Engineering. Currently he is proposing software designs by the use of Unified Modeling Language for the various research problems related to the Software Domains & Advanced Computer Architecture. He has published more than 55 International and National publications.

Deepak Arora: He is a Research Scholar, Dept. of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India. He got his Master Degree of Computer Applications in 2003 and M.Phil. Degree in Computer Science in 2006. Currently he is actively engaged in the research work on Distributed Computing Systems through the Unified Modeling Language. Ha has produced several outstanding publications on Distributed Computing Systems.