개인 정보 단말기의 응용을 위한 내장형 데이터베이스 관리 시스템의 설계 및 구현

제권엽, 최무희, 안병태, 강현석
경상대학교 컴퓨터과학과

Design and Implementation of Embedded Database Management System for Applications of PDA

Gwon-Youb Je, Mu-Hee Choi, Byung-Tae Ahn, Hyun-Syug Kang
Dept. of Computer Sceience, GyoungSang National University
tokai1682@hanmail.net

요 약

최근 개인 정보 관리, 인터넷 접속, 멀티미디어 교육, 게임 등 고급 응용들이 가능한 고성능의 PDA가 일반화되면서, PDA에서도 많은 양의 데이터를 직접 관리하면서 서버와 데이터를 상호교환할 수 있는 내장형 DBMS의 요구가 많아지고 있다.

본 논문에서는 가격 및 안정성이 뛰어난 Linux 운영체제에서 메모리 용량을 최소화하면서 파일시스템에 가까운 버클리 DB를 기반으로 한 PDA용 내장형 데이터베이스 관리 시스템을 설계 및 구현하였다.

1. 서론

정보 통신 기술의 발전으로 사용자가 언제, 어디서든 원하는 정보를 취득하거나 처리하고자 하는 욕구가 커지면서 휴대용 개인 정보 단말 즉, PDA(Personal Digital Assistant)가 크게 보급되고 있다. 이러한 PDA는 개인용이나 업무용으로 정보처리나 정보저장 및 검색 기능을 갖춘 손바닥만한 크기의 소형 컴퓨터로, 스케줄 정보나 주소록 정보 등을 관리하는데 많이 쓰이고 있다[1].

최근에는 PDA에서도 많은 양의 데이터를 직접 관리하면서 서버와 데이터를 상호교환 할 수 있는 내장형 DBMS의 사용에 대한 요구가 많아지고 있다. 그런데, PDA는 아직까지 데스크탑 PC에 비해 용량이 적고 전력소비에 대한 부담이 많아 다양한 응용을 사용하기에는 제약이 따른다. 따라서 PDA에 장착하는 내장형 DBMS도 이동 환경에서 소규모 메모리 등의 제약 조건을 고려해 개발되어야 한다[2].

본 논문에서는 파일 시스템에 가까우면서도 데이터베이스 시스템의 특성을 어느 정도 갖춘 버클리 DB[3]를 기반으로 PDA용 내장형 데이터베이스 관리 시스템인 EDBM(Embeded Database Manager)을 설계 및 구현하였다.

본 논문의 2장에서는 EDBM 개발방법을 소개하고, 3장에서는 EDBM의 설계를, 4장에서는 EDBM 구현 내용을 기술한다. 그리고 5장에서는 결론 및 향후 과제를 제시한다.

2. EDBM 개발방법

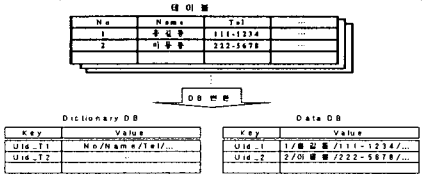
2.1 PDA 운영체제

현재 많이 사용되는 대표적인 PDA의 운영체제는 팜(Palm) OS, 윈도우 CE(Win CE), 리눅스 등이 있다. 그중 팜[4]은 PDA의 초창기 OS로써 현재 Palm OS 4.0까지 개발되어 있으며 여러 가지 종류의 PDA 운영체제로 사용되고 있다. 그리고 윈도우CE[5]는 흑백 및 컬러 디스플레이를 가능하게 하였으며 PDA를 윈도우 환경에서 사용할 수 있도록 하고 있다. 마지막으로 리눅스[6]는 팜 및 윈도우 CE의 기능을 가지면서도 다른 운영체제에 비해 가격이 저렴하고 네트워크 기능 및 다양한 장비지원이 가능하다. 우리는 EDBM 의 개발 플랫폼으로 공개 운영체제인 리눅스를 사용하였다.

2.2 버클리 DB

Sleepycat Software사가 지원하는 버클리 DB는 소스가 공개된 내장형 데이터베이스 시스템이다.[3] 사용이 쉽고, 여러 사용자들이 동시 접근이 가능하며, 시스템과 디스크 파손에도 견디는 트랜잭션 개념을 지원한다. 우리는 EDBM의 개발을 위해 이 버클리 DB를 사용하였다.

<그림 1>은 버클리 DB에서 데이터를 저장하는 기본 레코드 구조를 보이고 있다. 그림에서 보는 것과 같이 버클리 DB의 Recno 저장 방식은 UID를 기준으로 논리적 레코드 번호를 각 레코드들에 할당한다. 또한 레코드 번호로 해당 레코드를 검색하고 갱신할 수 있는 저장 구조를 사용한다. 이때 키 값은 0을 허용하지 않으며 키의 범위는 $1 \sim 2^8 - 1$이다.



<그림 1> 버클리 DB의 레코드 구조

이러한 버클리 DB는 그 용량이 회복, 트랜잭션 지원과 함께 모든 접근 방법들을 포함하더라도 175KB로서 적으면서도 매우 효율적인 데이터 관리 시스템이다. 따라서 PDA에서 사용하는데 매우 효과적이다

우리가 버클리 DB를 EDBM의 하부 기반 시스템으로 사용한 이유는 이와같이 공개 소스 제품이면서 적은 시스템 자원을 사용하는 장점을 갖기 때문이었다.

2.3 EDBM의 개발 방법

내장형 데이터베이스를 개발하는 방법은 크게 두 가지로 생각할 수 있다. 즉, (1) ORACLE, MS-SQL

과 같은 상용 DBMS 기능을 내장형 시스템에 적합하게 축약시키는 방법과, (2) 처음부터 새로운 내장형 DBMS를 개발하는 방법이 그것이다. 전자와 같이 상용 DBMS를 이용할 경우는 안전성이 뛰어나고 범용성과 보안성이 좋지만 시스템 자원을 많이 차지하는 단점이 있다. 이에 비해 후자의 경우, 이식성, 작은 족적(Small footing), 쉬운 개발, 인터넷 친화성이 뛰어나지만 부가 기능의 지원에 한계가 있다.

최근의 추세는 후자 접근의 단점을 보완하면서 자신의 응용에 필요한 부가 기능을 추가할 수 있게 하는 움직임이 강하다.

우리는 후자의 방법으로 버클리 DB 시스템 위에 다양한 레코드 형태의 정보 관리가 가능하도록 하는 방법으로 EDBM을 개발하였다.

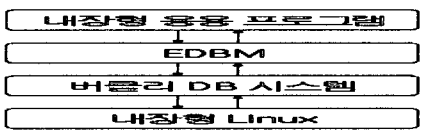
3. EDBM의 설계

이 장에서는 EDBM의 개발환경과 EDBM의 3 Layer 구조의 설계에 대해 알아본다.

3.1 EDBM의 개발환경

내장형 DBMS가 갖는 요구 사항중 가장 중요한 특징은 저장 매체의 제약에 의해 DBMS의 용량이 최소화되어야 한다는 것이다. 즉, DBMS의 엔진 크기 뿐만 아니라 데이터의 저장 공간도 줄여야 한다. 또한, 기본 DBMS의 사용자 접근 기능인 API 수준의 DB 생성 및 데이터의 입력·삭제·갱신 기능을 지원해야 하며, 검색 기능 또한 지원해야 한다. 본 논문에서는 이러한 점을 고려하여 새로운 PDA용 내장형 DBMS이 EDBM을 개발하였다.

<그림 2>는 EDBM의 개발 환경을 나타낸 것이다.



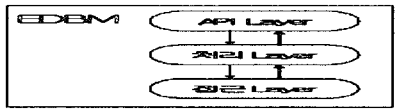
<그림 2> EDBM의 개발 환경

<그림 2>에서 보는 것과 같이 EDBM은 내장형 응용 프로그램(C 또는 C++)을 위한 API를 제공한다. 응용 프로그램에서 사용하게 될 API로는 크게 두 종류로 DB의 생성·삭제, Index의 생성·삭제를 기본으로 하는 DDL 역할의 API와 생성된 DB에 데이터를 입력·수정·삭제·갱신하는 DML 역할의 API가 있다. 여기에 대한 EDBM의 응답으로는, 요청의 올바른 수행여부를 미리 정의된 반환 값으로 돌려주거나, 검색한 데이터를 레코드 단위로 돌려주는 것이다.

3.2 EDBM의 3 Layer 구조

EDBM은 하부 저장소(현재 버클리 DB)의 변화나 응용 프로그램의 요구 변화에 따른 API의 확장에 유연하게 대처하기 위해 3개의 Layer로 구성하였다.

<그림 3>은 이러한 EDBM의 3 Layer 구조를 나타낸다.



<그림 3> EDBM의 3 Layer 구조

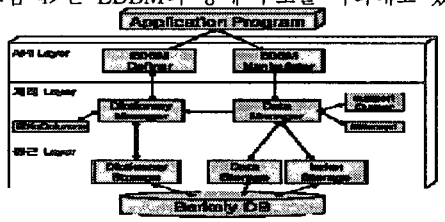
최상위 레벨인 API Layer는 응용 프로그램에서 EDBM을 조작할 수 있는 인터페이스를 제공한다. 처리 Layer는 API Layer로부터 전달되어 오는 응용 프로그램의 요청을 EDBM에서 제어 정보 형태로 유지하고, 데이터 관리가 용이하도록 미리 정의된 형태로 가공된다. 접근 Layer는 주어진 EDBM의 키 값과 EDBM의 데이터 값들을 버클리 DB 하부 저장소의 라이브러리(Library)를 사용하여 물리적으로 데이터를 저장하는 역할과 저장된 데이터를 처리 Layer로 반환하는 역할을 한다.

이러한 EBDM의 3 Layer 구조는 차후 응용 분야 및 저장 시스템의 확장에 대처하는데 매우 효과적이다.

4. EDBM의 구현

4.1 EDBM의 상세구조

<그림 4>는 EDBM의 상세 구조를 나타내고 있다.



<그림 4> EDBM의 상세 구조

<그림 4>는 4장에서 본 바와 같이 사용자 인터페이스를 제공하는 API Layer와 인터페이스를 통해 입력된 데이터를 EDBM 저장 데이터 타입으로 만드는 처리 Layer, EDBM 저장 데이터 타입을 하위 저장소인 버클리 DB에 저장하는 접근 Layer의 3개 Layer로 구성된다. 각 Layer의 구현에 대해 알아본다.

4.1.1 API Layer

API Layer는 EDBM과 응용 프로그램사이의 인터페이스를 담당하는 Layer로, 데이터베이스 정의 모듈(EBDM Definer)과 데이터베이스 조작 모듈(EDBM Manipulator)로 구현되었다. 데이터베이스 정의 모듈은 응용 프로그램으로부터 넘어온 DB 및 Index 정의 정보를 처리 Layer의 Dictionary Manager가 처리할 수 있도록 가공하여 제어 신호와 데이터를 넘기는 것이다. <그림 5>는 EDBM Definer의 주요 API 모듈이다.

API 모듈	기 능
createEDB()	DB 생성
dropEDB()	DB 제거
createEDB_pk()	인덱스 생성 - 기본키
createEDB_idx()	인덱스 생성 - 후보키
dropEDBIndex()	인덱스 제거

<그림 5> EDBM Definer의 주요 API 모듈

데이터 조작 모듈은 응용 프로그램에서 데이터를 입력·삭제·갱신하거나, 원하는 데이터를 검색하는데 사용된다. <그림 6>은 EDBM Manipulator의 주요 API 모듈이다.

API 모듈	기 능
open()	DB open
close()	DB close
insert()	데이터 삽입
update()	데이터(레코드) 수정
updateCol()	데이터(컬럼) 수정
del()	데이터 삭제
select()	특정 데이터 핸들러 선택
fetch()	선택 데이터 핸들러 호출
getData()	데이터값 호출

<그림 6> EDBM Manipulator의 주요 API

4.1.2 처리 Layer

처리 Layer는 크게 DB/Index 관리 모듈(Dictionary Manager)과 데이터 관리 모듈(Data

Manager)로 구현되었다. DB/Index 관리 모듈(Dictionary Manager)은 API Layer의 EDBM Definer 모듈에서 넘어온 DB명과 각 컬럼 정보들을 조합하여, Data Dictionary 정보 데이터 타입인 EDicColumns를 생성하며, 생성한 정보를 접근 Layer의 Dictionary Information Storage Module에 넘겨, 버클리 DB에 저장한다.<그림 7>은 Dictionary Manager의 주요 API 모듈이다.

API 모듈	기 능
ddl_createDb()	DB 생성
ddl_dropDb()	DB 제거
ddl_createDb_addField()	DB에 필드 추가
ddl_createIdx()	인덱스 생성
ddl_dropIndex()	인덱스 제거

<그림 7> Dictionary Manager의 API 모듈

데이터 관리 모듈(Data Manager)은 응용의 데이터와 EDBM의 Record 데이터간의 양방향 변환 및 이 데이터에 대한 인덱스 작성을 담당한다. <그림 8>은 Data Manager의 주요 API 모듈이다.

API 모듈	기 능
ddl_insert()	데이터 삽입
ddl_update()	데이터(레코드) 수정
ddl_updateCol()	데이터(컬럼) 수정
ddl_select()	특정 데이터 핸들러 선택
ddl_fetch()	선택 데이터 핸들러 호출
ddl_del()	데이터(레코드) 삭제

<그림 8> Data Manager의 주요 API 모듈

4.1.3 접근 Layer

접근 Layer는 데이터를 버클리 DB에 저장하는 모듈로 Dictionary Storage 모듈, Data Storage 모듈, Index Storage 모듈로 구성되어 있다. 각각의 모듈은 버클리 DB의 API를 이용하여 버클리 DB에 Dictionary, Data, Index 정보를 저장한다. <그림 9>는 Berkeley DB 조작의 주요 API이다.

API 모듈	기 능
open()	DB open
close()	DB close
put()	레코드 삽입
get()	레코드값 호출
del()	레코드 삭제
remove()	DB파일 삭제

<그림 9>버클리 DB 조작의 API 모듈

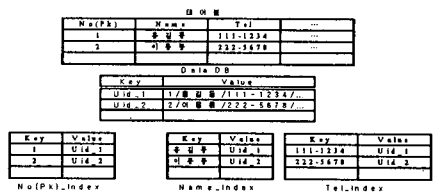
4.2 시스템 DB의 구조

EDBM은 데이터베이스 관리 기능을 원활하게 수행하기 위해 버클리 DB에서 제공하는 라이브러리를 통해 Dictionary DB, 데이터 DB, 인덱스 DB를 유지한다. <그림 10>은 Dictionary DB의 구조를 나타낸 것이다.

Key Value	Data Value
Uid_T1	1/홍 길 동 /111-1234/...
Uid_T2	2/이 몽 룡 /222-5678/...

<그림 10> Dictionary DB의 구조

이 DB는 EDBM에서 관리되는 모든 메타 정보를 관리하기 위해 사용된다. 이러한 구조를 갖게 된 것은 하부 저장시스템인 버클리 DB의 레코드 저장구조를 그대로 활용하기 위해서이다. Data/Index DB는 각 사용자 DB별로 1개의 Data DB 파일과 0개 이상의 Index DB 파일이 유지된다. <그림 11>은 Data/Index DB 파일 구조를 나타낸 것이다.



<그림 11> 사용자 DB의 Data/Index DB 파일

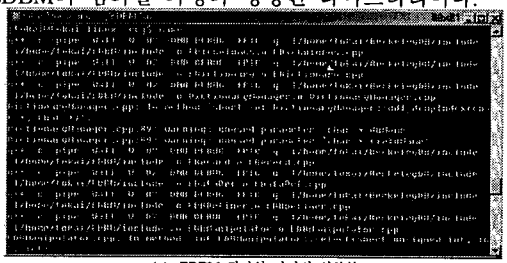
데이터 저장소는 데이터 정보를 저장하는 장소로써 하나의 이름에 하나의 Data DB가 만들어진다. 인덱스 저장소는 인덱스 정보를 저장하는 장소로써 하나의 Data DB에 여러개의 인덱스가 만들어 질 수 있다. <그림 12>는 데이터 저장소를 위한 버클리 DB의 파일 구조를 나타낸 것이다.

Key Value	Data Value
Uid_1	1/홍 길 동 /111-1234/...
Uid_2	2/이 몽 룡 /222-5678/...

<그림 12> Data DB 파일 구조

4.3 EDBM 구현 결과

EDBM의 구현은 C++로 작성되었으며 gcc Version 2.95.4로 컴파일 되었다. <그림 13>은 EDBM의 컴파일 과정과 생성된 라이브러리이다.



<a> EDBM 컴파일 과정의 일부분

<b> 동적으로 생성된 EDBM 라이브러리
<그림 13> EDBM의 컴파일 과정과 생성된 라이브러리

<그림 13 a>는 gcc 컴파일러를 이용하여 EDBM 라이브러리 소스를 컴파일 하는 과정의 일부분이며, <그림 13 b>는 컴파일되어 생성된 EDBM 동적 라이브러리를 보여주고 있다.

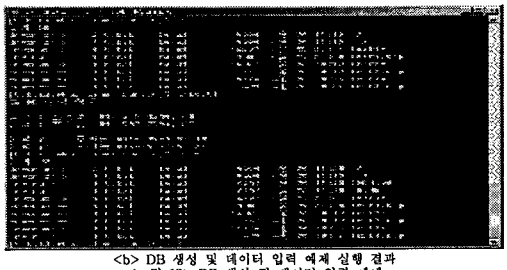
여기서는 생성된 동적 라이브러리 사용 예를 살펴본다.

4.3.1 데이터 생성 예

다음 <그림 14>는 EDBM을 사용하여 DB를 생성하고 데이터를 입력하는 예이다.

```
#include <EDBM.h>
int main() {  /* 동문주소록 DB 생성구문 */
    if(createEDB("EX",3,STRING_t,"c_name",
                       STRING_t,"c_phon", INT_t, "age") == 0) {
        createEDB_idx("EX", "c_name");
        createEDB_idx("EX", "c_phon");
        createEDB_idx("EX", "age");
        cout << "EX DB 생성 성공₩n" << endl;
    } else  cout << "DB 생성 실패₩n" << endl; /* 데이터 입력부 */
    EDBManipulator ptr;
    ptr.open("EX");
    edbm_id_t id;
    if ((id = (ptr.insert("조용필", "01X-215-1248", 40))) != 0) {
        cout << "입력 : 조용필, 01X-215-1248, 40" << endl;
    } else  cout << "데이터 입력 실패" << endl;
    if ((id = (ptr.insert("홍길동", "01X-547-7544", 25))) != 0) {
        cout << "입력 : 홍길동, 01X-547-7544, 25₩n" << endl;
    } else     cout << "데이터 입력 실패₩n" << endl;
                /* 데이터 출력부 */          ptr.select(0);
    while((id = ptr.fetch())) {
        cout << "입력 값 : " << " " << (char *)ptr.getData(0)
        << " " << (char *)ptr.getData(1) << " " << *(int
        *)ptr.getData(2) << endl;     }}
```

<a> DB 생성 및 데이터 입력 예제 구문



<b> DB 생성 및 데이터 입력 예제 실행 결과
<그림 13> DB 생성 및 데이터 입력 예제

위의 <그림 14 a>에서, 먼저 "EX"라는 DB와 인덱스를 생성하고 있다. 그리고 "조용필"과 "홍길동" 두 사람의 데이터를 입력하고 있다. 그리고 입력된 결과를 화면에 출력하고 있다. 이에 대한 실행 결과로 <그림 14 b>와 같이 EDBM.edb라는 Dictionary DB와 EX.db라는 데이터 DB를 생성되어, 데이터가 입력 및 출력된다.

4.3.2 데이터 변경 예

<그림 15>는 위의 <그림 14>의 예제에서 생성된 DB의 입력 데이터를 변경하는 예제이다.

```
#include <EDBM.h>
int main() {  /* 데이터 내용 변경 */
    EDBManipulator ptr;
    ptr.open("EX");
    edbm_id_t id;
    ptr.select(0, "조용필");
    while((id = ptr.fetch())) {
        if ((ptr.updateCol(id, 0, "나훈아")) != 0) {
            cout << "내용 변경 성공 : 조용필 -> 나훈아" << endl;
        } else  cout << "내용 변경 실패" << endl;    }
    ptr.select(0, "홍길동");
    while((id = ptr.fetch())) {
        if ((ptr.updateCol(id, 0, "아무개")) != 0) {
            cout << "내용 변경 성공 : 홍길동 -> 아무개₩n" << endl;
        } else  cout << "내용 변경 실패₩n" << endl;    }
                /* 데이터 출력부 */
    ptr.select(0);
    while((id = ptr.fetch())) {
        cout << "변경 결과 : " << " " << (char *)ptr.getData(0)
        << " "<< (char *)ptr.getData(1) << " " <<*(int *)ptr.
getData(2) << endl;    }}
```

<a> 데이터 변경 예제 구문

<b> 데이터 변경 예제 실행 결과
<그림 15> 데이터 내용 변경 예제

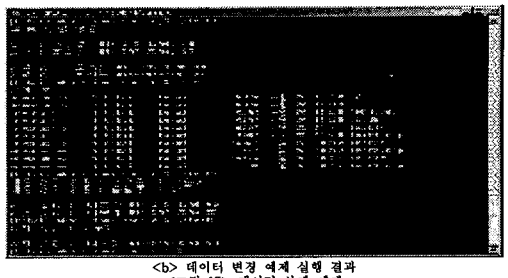
위의 <그림 15 a>에서 기 생성된 DB에 저장되어 있는 "조용필"과 "홍길동" 두 사람의 데이터에서 이름을 각각 "나훈아"와 "아무개"로 변경하고 있다. 그리고 <그림 15 b>는 변경된 결과를 보여주고 있다.

4.3.3 데이터 삭제의 예

다음 <그림 16>은 앞의 <그림 15>에서 변경했던 데이터 중 일부를 삭제하는 예제이다.

```
#include <EDBM.h>
int main() {   /* 데이터 삭제 */
    EDBManipulator ptr;
    ptr.open("EX");
    edbm_id_t id;
    ptr.select(0, "아무개");
    while((id = ptr.fetch())) {
        if ((ptr.del(id)) == 0)
            cout << "삭제 성공 : 아무개₩n" << endl;
    else  cout << "삭제 실패₩n" << endl; }/* 데이터 출력부 */
    ptr.select(0);
    while((id = ptr.fetch())) {
        cout << "삭제 결과 : " << " " << (char *)ptr.getData(0)
        << " " << (char *)ptr.getData(1) << " " << *(int *)ptr.
getData(2) << endl;    }}
```

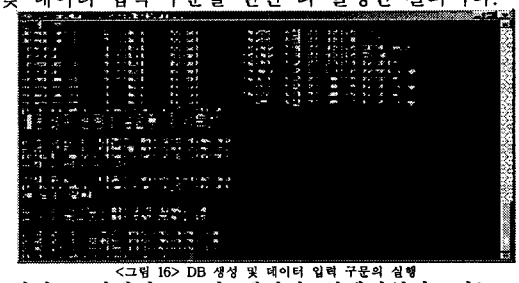
<a> 데이터 삭제 예제 구문



<b> 데이터 변경 예제 실행 결과
<그림 15> 데이터 삭제 예제

위의 <그림 16 a>의 예제는 <그림 15>의 데이터 변경 예제에서 "아무개"로 변경했던 데이터를 삭제하고 있다. <그림 16 b>의 실행 결과에서는 "나훈아"와 "아무개"의 데이터 중 "아무개"의 데이터가 삭제되어 "나훈아"의 데이터만 보여주고 있다.

다음 <그림 17>은 위의 <그림 14>의 DB 생성 및 데이터 입력 구문을 한번 더 실행한 결과이다.



<그림 16> DB 생성 및 데이터 입력 구문의 실행

위의 그림에서 DB의 생성이 실패되었다. 이는 기 실행에서 EX.db라는 DB 파일이 생성되어 있어 같은 이름의 DB 파일이 생성되지 않기 때문이다. 하지만 "조용필"과 "홍길동"의 두 데이터는 EX.db 파일이 기 생성되어 있기 때문에 그대로 입력되고 있다.

5. 결론 및 향후과제

본 논문에서는 개인 정보 단말 시스템 즉 PDA를 위한 내장형 DB가 가져야 될 기능에 초점을 맞춰 최소의 크기와 단순한 입출력을 정확하게 수행할 수 있도록 하는 새로운 내장형 DBMS인 EDBM을 개발하였다. 이는 파일 시스템 수준에 병행성 제어 등의 DB 기능을 추가한 버클리 DB 시스템을 기반으로 최소한의 필수 기능만을 갖는 시스템으로 개발 되었다.

향후 과제로는 현재 EDBM은 DBMS의 엔진과 데이터 저장 공간의 공간적 제약을 염두에 두고 개발되어 일반적인 상용 DBMS들에 비해 보안적 측면과 트랜잭션 개념의 지원이 많이 부족한 편이다. 개발된 EDBM이 좀 더 광범위한 분야에 적용되기 위해 범용 내장형 DBMS가 되기 위해서 서버측 DBMS와 동기화 등을 효과적으로 지원되어야 한다.

[참고 문헌]

[1] http://www.miraesys.com
[2] http://www.itel.co.kr
[3] M. Olson, K. Bostic, and M Seltzer, "Berkeley DB", Proceeding of the 1999 Summer Usenix Technical Conference, Monterey, California, June 1999.
[4] http://www.cesscom.com
[5] http://www.orafag.org/frqpol.htm
[6] http://www.linux.org