Journal of the Korea Academia-Industrial cooperation Society
Vol. 12, No. 7 pp. 3227-3235, 2011

DOI : 10.5762/KAIS.2011.12.7.3227

Miracle 임베디드 RDBMS 설계, 구현 및 성능 평가

서남원[1], 김경렬[1], 김수희[1*]
[1]호서대학교 컴퓨터공학과

Design, Implementation, and Performance Evaluation of an Embedded RDBMS Miracle

Nam-Won Seo[1], Keong-Yul Kim[1] and Su-Hee Kim[1*]
[1]**Department of Computer Engineering, Hoseo University**

요 약 이 논문에서는 관계형 임베디드 DBMS를 설계하고 프로토타입 Miracle RDBMS(MDB)를 개발하였다. MDB는 C로 개발되었고, 로컬에서 동작하며 유닉스와 윈도우즈 계열에서 운영 가능하다. SQL 인터페이스와 API 함수를 통해 데이터베이스를 접근하며 B^+ 트리 인덱스를 사용한다. 트랜잭션의 ACID를 보장하고 저수준의 잠금, 단일 테이블에 대한 SQL 문을 처리한다. 데이터의 처리 성능을 평가하기 위해 ARM용 EZ-S3C6410 보드를 이용하여 데이터 적재, 검색, 수정 및 삭제하는데 걸리는 시간을 실험하였다. SQLite와 처리시간을 비교해 보았는데 단위 연산에 대한 평균 처리시간이 SELECT와 INSERT에서 MDB가 각각 38.46%, 22.86% 더 빨랐으며, UPDATE와 DELETE에서 SQLite가 각각 28.33%, 26.00% 더 우수하였다. 이 실험은 데이터베이스에서 데이터를 가져오고 보내는 작업이 MDB에서 더 빠른 반면, B^+ 트리 인덱스는 SQLite에서 더 효율적으로 구축되었음을 보여준다.

Abstract In this paper, a relational embedded DBMS was designed and a prototype 'Miracle' RDBMS (MDB) was developed. MDB is written in C and works on Unix, Linux and Windows platforms locally. It accesses database through SQL interfaces and API functions and uses B^+ tree index. It guarantees ACID in transactions and supports low concurrency control and processes SQL statements on a single table. To evaluate the performance of MDB on an ARM board EZ-S3C6410 and to compare the performance of MDB with that of SQLite, an experiment was carried out to estimate processing times for insertion, selection, update and deletion operations. The result shows that the average times for selections and insertions in MDB were 38.46% and 22.86% faster than those in SQLite, respectively, but the average times for updates and deletions in SQLite were 28.33% and 26.00% faster than MDB, respectively, This experiment shows that fetching data from database and sending data to database in MDB is faster than in SQLite, but B^+ tree index is implemented more effectively in SQlite than in MDB.

Key Words : Embedded Relational DBMS, Design and Implementation, Data Processing Time

1. 서론

임베디드 시스템은 특정 기능을 수행하는 하드웨어와 소프트웨어가 결합된 컴퓨팅시스템이다. 초창기에는 군사・의료용 혹은 반도체 제조 장비를 포함하는 산업용으로 이용되어 왔다.

그러나 정보통신 및 반도체 기술의 발전으로 임베디드 시스템은 더 이상 특수한 분야에 국한되지 않고, 디지털 TV 혹은 휴대폰등과 같은 전자제품까지도 독자적인 프로세서와 운영체제가 탑재될 정도로 임베디드 시스템의 영역이 확대되고 있다[1].

하드웨어의 발전으로 임베디드 시스템은 많은 양의 데이터 처리가 가능해졌다. 최근에는 사용자의 편의를 고려하여 더 많은 데이터를 수용하고 관리를 필요로 하는 임

"이 논문은 2007년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행된 연구임" (20070224)
*교신저자 : 김수희(shkim@hoseo.edu)
접수일 11년 06월 07일 수정일 11년 06월 20일 게재확정일 11년 07월 07일

베디드 기기들이 보편화되고 있다. 임베디드 시스템의 성능이 향상될수록 하부단에서 이루어지는 데이터베이스 처리를 지원하는 임베디드용 데이터베이스 시스템의 확보가 중요시되고 있다[2].

이 논문에서는 파일럿 프로젝트 형태로 수행한 임베디드 관계형 DBMS의 설계와 구현을 토대로 하여 좀 더 체계적이고 확장된 틀에서 임베디드 관계형 DBMS 시스템을 설계하고 이의 프로토타입을 개발하고자 한다[3]. 이를 통해 임베디드 DBMS에 대한 전반적인 아키텍처와 주요 컴포넌트들의 설계와 개발 노하우를 축적하고자 한다.

이 논문의 구성은 다음과 같다. 2장에서는 기존의 임베디드 DBMS의 현황을 기술하고, 3장에서는 개발하고자 하는 임베디드 DBMS를 설계한다. 4장에서는 프로토타입 임베디드 DBMS 구현에 대해 논의하고, 5장에서는 개발한 DBMS의 처리속도를 평가하기 위해 SQLite와 비교 실험을 수행, 마지막으로 결론 및 앞으로의 연구 방향을 제시한다.

2. 임베디드 DBMS의 현황

임베디드 데이터베이스 시스템은 저장된 데이터의 접근을 필요로 하는 응용 소프트웨어와 매우 밀착하여 통합된 데이터베이스 관리 시스템으로, 애플리케이션에 내장되어 호출되는 방식 등 다양한 응용프로그램 인터페이스를 지원한다.

애플리케이션이 임베디드 데이터베이스를 필요로 할 때 수시로 이를 호출하여 수행하므로 데이터베이스에 대한 관리 활동이 거의 필요하지 않는 시스템에 임베디드 데이터베이스를 활용할 수 있고, 최종 사용자는 자신이 사용하는 기기에 데이터베이스가 내장되어 있다는 사실조차 인식하지 못한 채 편리하게 데이터를 처리 할 수 있다.

지금까지 많은 임베디드 데이터베이스 시스템 제품들이 개발되었다. 대표적인 임베디드 시스템으로 Sybase사의 Advantage Database Server, Oracle사의 Berkeley DB, Microsoft사의 Extensible Storage Engine, Richard Hipp가 개발한 SQLite 등이 있다[4-8]. 국산 임베디드 DBMS로는 알티베이스, ZeroWait, 카이로스 등이 있다[9].

여기에서는 임베디드 DBMS로 널리 이용되고 있는 Berkeley DB와 SQLite에 대해 간단히 살펴본다.

2.1 Berkeley DB

Berkeley DB(이하 BDB)는 빠르고 소스코드가 공개되어 있다. C로 개발된 소프트웨어 라이브러리로 C++, PHP, Java 등 대부분의 프로그래밍 언어로 사용할 수 있는 API를 지원하고, x86상에서 컴파일 되었을 때 약 700KB의 크기이다.

BDB는 관계형 데이터베이스가 아니고, 로컬로 동작하며 네트워크 접근을 지원하지 않는다.

Berkeley DB의 특징은 대체로 다음과 같다[5].

- 고성능의 임베디드 데이터베이스
- SQL이 없는 대신 간단한 API로 데이터를 저장하거나 조회
- key/value 형태로 데이터를 저장
  임의의 key/value 쌍들을 바이트 배열로 저장하고 단일키에 대해 여러 데이터 아이템을 지원
- 다양한 플랫폼 지원
  대부분의 유닉스계열 및 리눅스플랫폼, MS-Windows 플랫폼, Mac OS 등에서 운용이 가능
- 고수준의 데이터베이스
  BDB는 ACID 트랜잭션, 작은 단위의 잠금, 백업과 복제 등과 같은 많은 고수준의 데이터베이스 특징들을 지원

잠금 관리, 로그 관리, 메모리 관리 등과 같은 데이터베이스의 컴포넌트들이 라이브러리 내에서 수행되고, 멀티 프로세스 또는 멀티 쓰레드는 동시에 데이터베이스를 사용할 수 있다.

2.2 SQLite

SQLite는 안드로이드 및 아이폰에 탑재되면서 가장 광범위하게 사용되고 있는 임베디드 관계형 DBMS이다.

SQLite의 특징은 대체로 다음과 같다[7].

- 트랜잭션은 ACID를 보장
- Zero Configuration: 별도의 설치나 관리가 필요하지 않음
- 외부 종속성이 없는 self-contained
- SQL92 표준의 대부분을 구현하였음
- 코드 풋프린트(code footprint): 설정에 따라 실행파일의 크기가 190KB～325KB로 작음
- 대부분의 일반적인 연산에서 범용적인 Client / Server 데이터베이스보다 더 빠름
- 이식성이 좋아 거의 모든 OS에서 작동 가능
- ANSI-C로 개발되었고 소스코드는 공개
- 5개의 데이터타입 지원
  NULL, INTEGER, REAL, TEXT, BLOB
- 테라바이트급의 데이터베이스와 기가바이트급의 문자열과 바이너리 데이터를 지원

3. ‘Miracle’ 관계형 임베디드 DBMS 시스템 설계

이 장에서는 ‘Miracle’ 임베디드 DBMS(이하 MDB)의 아키텍처와 이를 구성하는 주요 컴포넌트들을 설계한다.

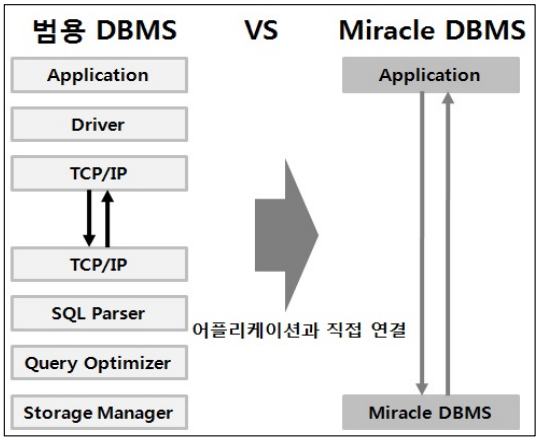
3.1 기존 범용 DBMS와의 비교

개발하고자 하는 MDB는 임베디드 시스템 환경에 특화된 관계형 DBMS로, 로컬에서 동작하며 기본적인 SQL 문을 지원하고 API 함수를 통해 데이터베이스를 관리한다. 이는 범용 DBMS에서 필요한 여러 가지 작업처리 과정을 생략 및 간소화함으로써 제한된 기능으로 트랜잭션 처리를 하지만, 반면 소형 메모리와 저장장치를 필요로 하며 데이터 처리 속도를 극대화한다[10].

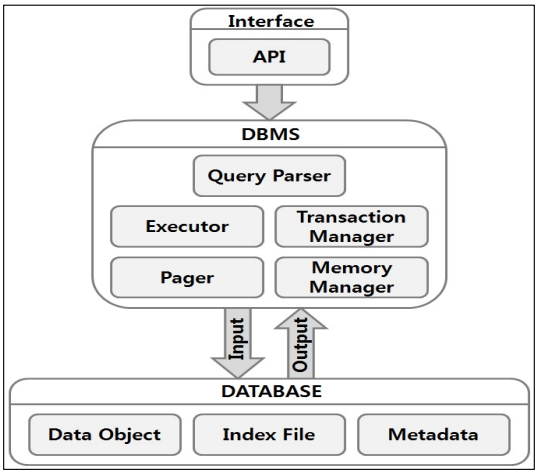
그림 1은 기존의 범용 DBMS와 MDB에서 응용프로그램과의 배치와 접근 형태의 차이를 간단히 나타낸다.

3.2 시스템 아키텍처

MDB는 크게 인터페이스, 관계형 임베디드 DBMS, 데이터베이스로 구성된다. 그림 2는 개발하고자 하는 MDB의 아키텍처를 나타낸다. 그림 2의 아키텍처를 보면 일반적인 범용 DBMS의 아키텍처와 크게 다를 것이 없지만, 임베디드 MDB에서는 모든 기능들이 간소화되고 규모가 축소된다.



[그림 1] 범용 DBMS와 Miracle DBMS
[Fig. 1] General DBMS and Miracle DBMS



[그림 2] Miracle DBMS 아키텍처
[Fig. 2] Miracle DBMS Architecture

3.2.1 인터페이스

MDB는 프로그램 내장형 소프트웨어로 자신의 프로그램 소스코드에 MDB의 소스코드를 포함하고, API 함수로 사용할 수 있는 인터페이스와 라인모드 SQL 인터페이스를 지원한다.

3.2.2 DBMS

MDB는 크게 파서(Parser), 작업처리자(Executor), 트랜잭션 관리자(Transaction Manager), 페이저 (Pager), 메모리 관리자(Memory Manager)로 구성된다.

사용자가 어떤 질의를 입력하면 파서가 구문을 분석하여 작업 처리자에게 넘겨준다. 작업 처리자는 요청된 작업을 수행한다. 각 작업은 입출력, 제어 등으로 나뉘며 작업 수행은 트랜잭션 관리자에 의해 ACID가 보장된 트랜잭션 형태로 수행된다.

페이저는 데이터베이스 파일 내에 페이지를 관리하여 메모리 낭비를 최소화한다. 또한, 테이블의 잠금 설정 및 해제를 담당한다. 메모리 관리자는 데이터베이스가 사용하는 기기 내의 메모리를 관리하여 오버플로우 등의 메모리 오류가 발생하지 않도록 한다. 이들은 MDB의 주요 컴포넌트로 다음 절에서 상세히 서술한다.

3.2.3 데이터베이스

데이터베이스는 임베디드 기기내의 데이터를 데이터베이스화 하여 저장한다.

- 데이터 객체: 저장되고 제어되는 데이터
- 인덱스 파일: 자료의 검색을 위한 인덱스
- 메타데이터: 데이터 객체와 인덱스 등 관리

3.3 MDB의 주요 컴포넌트

이 절에서는 3.2절에서 소개한 MDB의 주요 컴포넌트에 대해 논의한다.

3.3.1 질의 파서

파서는 사용자가 입력한 질의문에서 토큰을 추출하고, 구문분석과 메타데이터를 이용하여 의미분석을 한다. 분석이 끝난 질의문은 작업 처리자에게 전달되어 처리된다.

3.3.2 작업처리자

질의문은 SQL 혹은 API 함수를 통해 MDB로 보내지고 파서를 거친 후에 작업 처리자가 이를 처리한다. 작업처리자는 작업 수행을 위해서 필요한 페이지에 대한 lock을 트랜잭션 관리자에게 요청하고 lock을 획득한 후 페이저에게 필요한 페이지에 대해서 요청한다. 그리고 메모리 관리자에게 필요한 메모리를 할당 받아 작업을 처리한다.

작업처리자가 하는 주된 작업은 다음과 같다.

- 데이터베이스 환경 생성(OPEN DB)
- 테이블 생성 및 삭제
  (CREATE/DROP TABLE)
- 데이터 검색/삽입/삭제/수정
  (SELECT/INSERT/DELETE/UPDATE)
- 트랜잭션 제어
  (BEGIN, COMMIT, ROLLBACK)
- SQL 실행(EXEC SQL)

3.3.3 트랜잭션 관리자

트랜잭션은 ACID를 보장하고, 트랜잭션내의 읽기와 쓰기 연산은 필요한 잠금을 획득한 후에 수행된다. 동시성 제어를 위해 테이블 단위의 잠금을 사용하고 읽기 연산을 위해 공용잠금(S-lock)과 쓰기 연산을 위해 전용잠금(X-lock)을 지원한다.

트랜잭션의 완료를 위해 COMMIT을 사용하고 트랜잭션의 취소를 위해 ROLLBACK을 지원한다. 커밋은 자동커밋과 수동커밋이 있다.

3.3.4 동시성 제어

테이블 잠금을 통해 동시성을 제어함으로써 낮은 수준의 병행처리를 지원하다. 데이터의 읽기 연산을 위해 외부 응용프로그램들은 데이터를 공유할 수 있으며 공용잠금을 획득한 후에 데이터를 읽을 수 있다. 그리고 데이터 쓰기 연산을 위해 외부 응용프로그램들은 독점(전용)잠금을 획득한 후에 데이터를 쓸 수 있다. 테이블에 대한 잠금 리스트를 구성하여 잠금을 관리하는 데 다음에 소개하는 페이저가 트랜잭션 관리자와 연계하여 담당한다.

3.3.5 페이저

페이저는 테이블 페이지를 읽거나 쓰는 작업을 수행하는 페이지 관리자 컴포넌트이다. 테이블 페이지를 접근하기 위해 테이블을 잠금하고, 접근이 끝난 후에 잠금을 해제한다.

테이블 캐싱 기법을 이용하여 데이터 처리를 위한 성능향상을 도모한다. 또한 트랜잭션에 대한 임시파일 쓰기 및 읽기, flush 작업, SELECT 문의 결과 페이지 구성 등의 작업을 수행한다.

- 테이블 정보 캐싱

테이블의 데이터를 해당 파일에서 가져오기 전에, 원하는 데이터가 캐시에 있는지 확인한다. 이 작업은 낮은 수준의 캐싱 기법을 요구한다.

- 트랜잭션 임시 파일

트랜잭션 수행하면서 처리한 연산의 결과 값은 임시파일에 저장되는데, 트랜잭션이 COMMIT/ROLLBACK 되기 전까지 트랜잭션의 결과 값을 저장한다.

- 페이지 커서

SELECT, DELETE, UPDATE 작업 시에 해당 조건을 만족하는 데이터가 있는 페이지 주소를 기억해야 할 경우가 있는데, 이때 페이지 커서를 사용하여 해당 주소를 저장한다.

3.3.6 메모리 관리자

데이터 처리를 위해서 데이터베이스의 일부를 메모리에 로드해야 한다. 주 메모리의 용량이 적은 임베디드 시스템의 특성상 작은 크기의 메인 메모리를 사용하여 효율적인 작업수행을 할 수 있어야 한다. 이를 위해 MDB의 메모리 관리는 메모리 풀을 사용하지 않고 낮은 수준으로 메모리를 관리한다. 먼저 MDB 사용자는 현재 사용할 수 있는 최대 메모리 공간을 설정하고, 작업 처리자는 사용자가 설정한 메모리 공간 내에서 작업에 사용할 각 메모리 공간을 분배한다. 설정할 수 있는 메모리 공간의 종류는 다음과 같다.

- 데이터베이스를 사용하기 위한 필수 정보
  - 데이터베이스 환경 정보
  - 테이블 캐시
  - 메모리 버퍼 정보
  - 테이블 잠금 정보
- 데이터베이스 제어 시 사용되는 정보

- 임시 버퍼: 각종 작업 수행 시에 임시 저장 공간
- 트랜잭션 임시 데이터
- 커서 정보 저장 공간 : 검색될 결과 페이지의 주소 저장
- SELECT 문 수행 시에 결과값을 저장하는 공간

3.3.7 B^+ 트리 인덱스

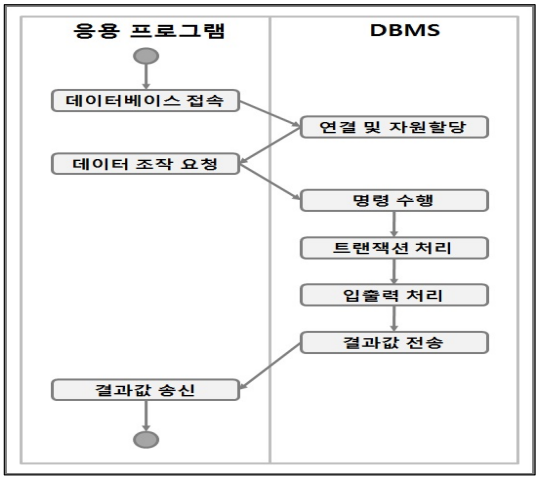
데이터의 검색 성능을 높이고 삽입, 삭제, 수정 연산을 빠르게 처리하기 위해 B^+ 트리 인덱스를 사용한다. B^+ 트리는 다양한 검색 형태에 우수한 성능을 가지며 단말노드들의 레벨이 모두 같은 트리이다[11,12].

인덱스 페이지의 길이는 모두 같고 인덱스 페이지는 헤더정보를 포함하는 첫 페이지를 제외하고 모두 같은 형식의 페이지로 구성된다. 이를 통해 인덱스 파일내의 특정 페이지를 검색할 때 페이지번호를 이용하여 검색할 수 있다.

기본키에 대한 인덱스는 테이블이 생성될 때에 자동으로 생성된다.

3.4 MDB의 동작방식

응용 프로그램에 내장되어 실행될 때에, MDB는 응용 프로그램과 함께 메모리에 적재되어 수행된다. 그러므로 응용프로그램의 요청에 따라 작업을 수행하며, 기본적인 작업 순서는 그림 3과 같다.



[그림 3] Miracle DBMS 작업 처리 순서
[Fig. 3] Miracle DBMS Operation Sequence

1) 응용프로그램은 데이터베이스에 접속
2) MDB는 응용프로그램에게 자원할당
3) 응용프로그램은 MDB에게 데이터조작 요청
4) MDB는 요청한 작업을 수행
5) 각 작업은 트랜잭션 관리자에게 전송되어 트랜잭션 단위로 수행
6) 트랜잭션 수행 중에 입출력 작업이 이루어짐
7) 결과값을 응용프로그램에게 보냄

4. MDB 구현

3 장에서 설계한 MDB를 기반으로 프로토타입 프로세스 내장형 프로그램 MDB를 인텔 호환 CPU, 1G 메모리의 윈도우즈 환경에서 개발하였다.

4.1 MDB 사양과 특징

MDB의 사양과 특징을 다음과 같다.

- 종류: 프로세스 내장형 관계형 DBMS
- 사용 가능 운영체제: 유닉스와 리눅스 플랫폼, MS-Windows 플랫폼, MAC OS X 환경
- 개발 언어 : ANSI-C
- 인터페이스
  - 응용프로그램에서 접근: API 함수 제공
  - SQL 인터페이스 제공
- 지원 데이터 형
  CHAR, SHORT, INTEGER, LONG, FLOAT
- 크기
  - LOC : 2만 5천
  - 소스 코드 용량 : 750KB
  - 실행 파일 용량 : 132KB
- 기본키에 대한 인덱스 자동 생성
- 트랜잭션 ACID 보장
- 낮은 수준의 병행처리: 테이블 단위 잠금/해제
- 테이블 생성 및 삭제 (DDL)
  - 테이블 CREATE/DROP문 (ALTER 미지원)
- 데이터 조작 (DML)
  - SELECT, INSERT, UPDATE, DELETE문
- 안정적인 동작
  - 메모리 유수 등의 오류를 사전에 방지
  - 치명적인 오류에서 복구 가능하도록 개발
- 데이터 독립성
  - 데이터베이스와 응용프로그램이 독립적으로 운용됨

4.2 MDB 실행

4.1절에서 언급하였듯이 개발한 MDB는 유닉스와 리

눅스 플랫폼, MS-Windows 플랫폼, MAC OS X 환경에서 실행가능하다.

이 절에서는 임베디드 보드에서 실행하기 위해 실행화일 생성하는 방법과 참조용으로 올려놓은 실행화일의 실행에 대해 간단히 설명한다.

- 임베디드 보드에서 실행
  - ARM용 보드에 포팅하기 위해 MDB를 크로스 컴파일하여 실행파일 생성함
  - NFS를 구축하여 실행파일을 타겟보드에 전송함
- 참조용 실행화일
  참조용으로 올려놓은 실행화일 Miracle_DBMS.exe [13]를 다운로드 받아 윈도우 OS 상에서 실행할 수 있다.

4.3 SQL 인터페이스을 이용한 MDB 구동

그림 4는 MDB를 구동한 화면이다. 간단한SQL 명령들을 예시로 실행하였다.

```
root@ubuntu:/# ./MiracleDBMS
Starting Miracle Database System...

MSQL> CREATE TABLE Person(
        id integer PRIMARY KEY,
        name char(30),
        address char(50));

Table Created.

MSQL> insert into Person
      values(1000, 'Hong Kil Dong', 'Seoul');

Insert Completed.

MSQL> insert into Person
      values(1001, 'Chul Su', 'Dae Jun');

Insert Completed.

MSQL> delete from Person where id = 1001;

Delete Completed.

MSQL> commit;
MSQL> update Person set id = 1005 where id = 1000;

Update Completed.

MSQL> select * from Person;
1005 | Hong Kil Dong | Seoul

1 rows selected.
```

[그림 4] MDB 질의 처리의 예
[Fig. 4] Example of Query Processing in MDB

5. MDB와 SQLite의 처리속도 평가

이 장에서는 개발한 MDB의 성능이 어떤지를 테스트하기 위해 벤치마킹 대상으로 가장 널리 사용되고 있는 SQLite를 선택하였다. 메인 메모리의 크기가 적고 데이터 저장 용량이 적은 임베디드 보드에서는 데이터 처리 속도가 매우 중요하다. 이미 널리 사용되고 있으며 몇 차례의 대규모 업그레이드를 수행한 SQLite와 MDB의 프로토타입 버전과는 여러 측면에서 비교할 수준이 못 된다.

그럼에도 불구하고, 2장에서 언급한 대로 SQLite는 속도가 매우 빠른 것으로 평가되고 있으므로 MDB가 데이터를 처리하는 성능이 SQLite와 비교하여 어느 정도 위치에 있는가를 파악하는 것은 매우 의미가 있다고 하겠다.

5.1 실험방법

이 실험에서는 ARM용 EZ-S3C6410 보드를 이용하여 기본적인 4개의 연산 SELECT, INSERT, UPDATE, DELETE에 대해 처리시간을 비교해보기 위해 간단한 실험용 테이블을 생성하고 각 연산에 대한 수행 시간을 측정하는 프로그램을 작성하였다. 실험용 샘플 테이블 Test의 스키마는 표 1과 같다. Test 테이블 생성 시에, MDB와 SQLite 모두 기본키인 ID에 대해 인덱스를 자동으로 생성한다.

[표 1] 실험용 샘플 테이블 스키마
[Table 1] Experimental Sample Table Schema

Test

속성명	ID	NAME	AGE
도메인	integer	char(20)	integer
비고	기본키		

먼저, Test 테이블에 2000개의 행을 INSERT하는데 걸리는 시간, 그 다음으로 SELECT, UPDATE, DELETE 연산들에 대해 각각 2000번씩 수행하는데 걸리는 시간을 차례대로 실험해 보았다. 그 다음으로 5000개의 행을 INSERT한 후에 SELECT, UPDATE, DELETE 연산들에 대해 각각 5000번씩 차례대로 실험해 보았다.

표 2는 사용한 SQL문과 이를 수행하는 데 소요된 처리시간을 나타낸다.

[표 2] 실험에 사용한 질의문 및 처리 시간
[Table 2] Queries used in Experiment and Processing Times

질의	수행횟수	DBMS	총 수행시간 (sec)	평균속도 (sec)
INSERT INTO TEST VALUES (rand#, 'XXXX', 25)	2000	MDB	4.94	0.0025
		SQLite	7.07	0.0035
	5000	MDB	13.48	0.0027
		SQLite	17.58	0.0035
SELECT * FROM TEST WHERE ID = rand#	2000	MDB	1.54	0.0008
		SQLite	2.62	0.0013
	5000	MDB	4.27	0.0008
		SQLite	6.59	0.0013
UPDATE TEST SET ID = ID + 6000 WHERE ID = rand#	2000	MDB	10.74	0.0054
		SQLite	8.79	0.0044
	5000	MDB	30.17	0.0060
		SQLite	21.60	0.0043
DELETE FROM TEST WHERE ID = rand#	2000	MDB	9.15	0.0046
		SQLite	7.40	0.0037
	5000	MDB	24.95	0.0050
		SQLite	18.32	0.0037

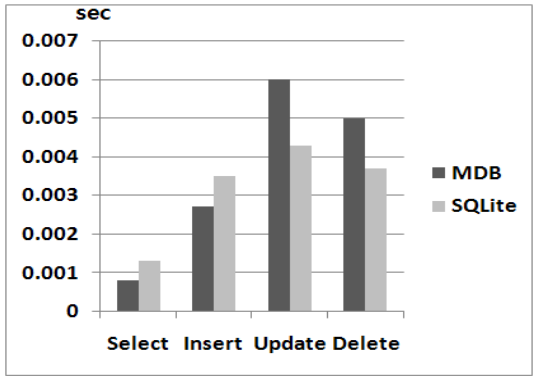
표 2에서 각 SQL 문에 있는 rand#는 1과 2000(5000) 사이의 정수이며, 각 정수는 한번만 랜덤하게 생성되도록 하였다. 실질적으로 임베디드 시스템에서 동일한 형태의 질의가 2000번 혹은 5000번 연속적으로 수행될 가능성은 전혀 없지만, MDB와 SQLite를 대상으로 처리 속도를 측정하여 MDB가 상대적으로 어떤가를 파악해보기 위한 것이다.

5.2 실험 결과 및 분석

표 2에 있는 5000번 반복한 실험 데이터에서, 한번의 INSERT에 걸린 평균 시간은 MDB와 SQLite에서 각각 0.0027sec과 0.0035sec이다. SELECT에 걸린 평균 시간은 MDB와 SQLite에서 각각 0.0008sec과 0.0013sec이다. UPDATE에 걸린 평균 시간은 MDB 0.0060sec, SQLite 0.0043sec 이다. DELETE에 걸린 평균 시간은 MDB 0.0050sec, SQLite 0.0037sec이다. MDB에서 INSERT와 SELECT가 각각 22.86%, 38.46% 빨랐으며, UPDATE와 DELETE에서는 SQLite가 각각 28.33%, 26.00% 만큼 더 빠름을 알 수 있다[표 2, 표3, 그림5 참조].

[표 3] MDB와 SQLite의 성능 요약
[Table 3] Summary of Performance of MDB and SQLite

질의 타입	단위연산당 평균속도(sec)		비교
	MDB	SQLite	
SELECT	0.0008	0.0013	MDB: 38.46% 더 빠름
INSERT	0.0027	0.0035	MDB: 22.86% 더 빠름
UPDATE	0.0060	0.0043	SQLite: 28.33% 더 빠름
DELETE	0.0050	0.0037	SQLite: 26.00% 더 빠름



[그림 5] MDB와 SQLite의 평균 처리속도 비교
[Fig. 5] Comparison of Average Processing Speeds of MDB with SQLite

쓰기 연산 중 INSERT는 MDB가 더 빠르고 UPDATE와 DELETE 연산에서 SQLite가 더 우수하므로 두 시스템의 여러 컴포넌트들을 분석해 본 결과 인덱스에서 SQLite가 더 효율적으로 구현되었음을 발견하였다. MDB와 SQLite 둘 다 B^+ 트리 인덱스를 사용하는데 SQLite에서는 인덱스의 각 노드의 크기를 페이지 크기와 동일하게 설정하여 가능하면 많은 인덱스 셀(키 값과 포인터)들이 저장되지만, MDB에서는 인덱스의 각 노드의 크기를 페이지 크기와 동일하게 설정하지만 인덱스 페이지 내의 일부 공간이 허비되고 있음을 파악하였다. 한 노드에 많은 셀(키 값과 포인터)이 저장되면 B^+ 트리의 높이가 작아지므로 검색속도가 빨라지며, 또한 인덱스의 갱신도 빠르게 이루어질 수 있다. MDB에서는 SELECT와 INSERT가 더 우수한 성능을 보여준다. SELECT 연산에서는 비록 인덱스가 최적으로 구축되지는 않아 데이터 검색이 효율적이지 않지만, 검색된 데이터를 파일에서 가져오는 작업은 매우 효율적으로 처리되고 있음을 알 수 있다. INSERT에서는 삽입되는 행이 파일의 마지막에 쓰여지

고 인덱스의 갱신이 수반된다. MDB의 인덱스 구현이 효율적이지 못함에도 불구하고, MDB의 INSERT 연산이 더 우수한 것은 MDB에서 파일에 쓰는 작업이 매우 효율적으로 이루어지고 있음을 알 수 있다.

그리고 MDB는 2000번 수행할 때와 5000번 수행할 때 단위 연산 당 평균 처리시간이 다소 다른 반면, SQLite는 반복 횟수에 관계없이 거의 일정함을 알 수 있다.

2장에서 언급한 것처럼 처리 속도가 매우 빠른 것으로 평가 받고 있는 SQLite와 비교할 만한 수준으로 실험결과가 나온 것은 매우 고무적이라 할 수 있다.

6. 결론 및 향후 계획

이 논문에서는 관계형 임베디드 DBMS를 설계하고 프로토타입을 개발하였다. 임베디드 시스템 환경에 특화된 관계형 DBMS로 로컬에서 동작하며 기본적인 SQL문을 지원하고 API 함수를 통해 데이터베이스를 관리하며 B^+ 트리 인덱스를 지원한다. 구현에 있어서 여러 가지 한계로 인하여 저수준의 잠금, 저수준의 메모리 관리를 하고 있고 단일 테이블에 대한 SQL 문만을 처리하고 있다. 기본키에 대한 인덱스를 테이블 생성 시에 자동으로 생성하지만, 수동으로 인덱스를 생성하는 SQL문을 지원하지 않고 있다.

데이터의 처리 성능을 평가해 보기 위해 ARM용 EZ-S3C6410 보드를 이용하여 간단한 테이블을 생성하고 데이터를 적재하는 데 걸리는 시간, 검색, 수정 및 삭제하는데 걸리는 시간을 SQLite와 비교해 보았다. 이 실험을 통하여 INSERT와 SELECT 연산에서는 MDB가 각각 22.86%, 38.46% 빨랐으며, UPDATE와 DELETE에서는 SQLite가 각각 28.33%, 26.00% 만큼 더 우수하였다. SQLite는 2000번의 반복이나 5000번의 반복에 관계없이 각 연산 당 걸리는 평균 시간이 거의 일정하였지만, 반면 MDB는 다소 편차가 있음을 볼 수 있었다. 이 실험을 통해 SQLite와 성능을 비교분석하면서 MDB의 B^+ 트리 인덱스 구현에서 필수적으로 보완할 점을 파악하게 된 것이 큰 수확이라고 볼 수 있다.

MDB는 초기버전으로 각 컴포넌트별로 보완하고 개선할 부분들이 아직 많이 있다. 향후 연구에서 추가하고 확장하고자 하는 큰 방향은 다음과 같다.

- B^+ 트리 인덱스 수정 보완
- 질의문에서 논리적 연산 지원 (and, or 등)
- 다중 테이블을 대상으로 하는 질의문 지원
- 고수준의 잠금 관리
- 기본키가 아닌 다른 속성에 대한 인덱스 생성문 지원
- 그 외, SQL92 표준에 의거한 데이터타입 및 기능 추가

References

[1] JeongJeong Woo, GongJun Sa, "Embedded Linux Basics and Applications", Hanbit Media, inc. 2007

[2] YongJin Sin, “Maximizing Through-put with Embedded DBMS,” Embedded World, May, 2008.

[3] Dong-SeoK Lee, Nam-Won Seo, Dong-Hyun Nam, Georgy Ni, Su-Hee Kim, “Design and Implementation of an Embedded DBMS”, Korea Information Science Society, Vol.36 No.2C, pp. 82-86, 2009, 11.

[4] http://www.sybase.com/products/databasemanagement/advantagedatabaseserver

[5] http://www.oracle.com/us/products/database/berkeley-db/index.html

[6] http://en.wikipedia.org/wiki/Extensible_Storage_Engine

[7] http://www.sqlite.org

[8] http://en.wikipedia.org/wiki/Embedded_database

[9] http://blog.paran.com/blog/detail/postBoard.kth?blogDataId=552438&pmcId=mtkim&page=0&totalCount=0&pageStyle=null&myCateId=0&yearMonth=null&rDay=null&style=Board

[10] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widsom. Database System Implementation DepARMent of Computer Science Stanford University, 2000.

[11] http://en.wikipedia.org/wiki/B%2B_tree

[12] Raghu Ramakrishnan, Johannes Gehrke. Database Management Systems 3rd Edition. McGraw-Hill, 2007.

[13] http://blog.naver.com/runter1

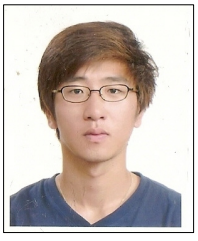
김 수 희(Su-Hee Kim) **[정회원]**

- 1979년 : 부산대학교 졸업 (학사)
- 1986년 : University of Georgia (전산학 석사)
- 1988년 : University of Georgia (수학 석사)
- 1993년 : University of South Carolina (전산학 박사)

- 1993년 ~ 1994년 : Benedict College 조교수
- 1994년 ~ 현재 : 호서대학교 컴퓨터공학부 교수

<관심분야>
XML 정보검색, 데이터베이스, 유비쿼터 데이터 처리

서 남 원(Nam-Won Seo) **[정회원]**



- 2010년 2월 : 호서대학교 컴퓨터공학과 졸업 (학사)
- 2010년 3월 : 호서대학교 대학원 컴퓨터공학과 (석사과정)

<관심분야>
데이터베이스, 유비쿼터스 데이터 처리

김 경 열(Kyung-Ryul Kim) **[정회원]**

- 2011년 2월 : 호서대학교 컴퓨터공학과 졸업 (학사)
- 2011년 3월 : 호서대학교 대학원 컴퓨터공학과 (석사과정)

<관심분야>
데이터베이스, 모바일응용, 무선통신