Journal of KIIT. Vol. 12, No. 11, pp. 181-192, Nov. 30, 2014. pISSN 1598-8619, eISSN 2093-7571 181
http://dx.doi.org/10.14801/kitr.2014.12.11.181

논리적 로깅 기반의 새로운 SQLite 복구기법

이준희*, 신민철**, 장용일***, 박상현****

A Novel Recovery Scheme for SQLite Based on Logical Logging

Joonhee Lee*, Mincheol Shin**, Yongil Jang***, and Sanghyun Park****

본 연구는 LG전자의 지원을 받아 수행된 것임.

요 약

SQLite는 로컬 응용프로그램, 임베디드 기기 및 스마트폰 등에 사용되는 대중적인 관계형 데이터베이스 관리 시스템(RDBMS)이다. SQLite는 트랜잭션의 원자성과 지속성을 보존하기 위해 물리적 로깅 기반의 복구 기법을 사용한다. 그런데 물리적 로깅은 한 페이지 내의 일부 데이터가 수정되더라도 전체 페이지를 저장하기 때문에 로그의 크다. 따라서 로그를 관리하는 비용도 크며 SQLite를 사용하는 프로그램의 응답시간 또한 길어진다. 본 논문은 SQLite를 위한 새로운 복구 기법인 Delta-WAL을 제안한다. Delta-WAL은 논리적 로깅 기반의 복구 기법으로 수행되는 작업의 종류 및 입력 값만을 저장하여 생성되는 로그의 크기를 줄인다. 실험결과, Delta-WAL은 기존의 기법에 비해 더 작은 로그를 생성하였으며, 트랜잭션 수행시간 또한 더 짧았다.

Abstract

SQLite is a popular relational database management system(RDBMS) mainly used in local application, embedded device, and smartphone. In order to preserve transactional atomicity and durability, SQLite uses recovery schemes that are based on physical logging. Physical logging generates large log file, because whole page is stored even if only a small portion of page is modified. Therefore, log maintenance cost of physical logging is expensive, and it causes delay in applications that use SQLite. In this paper, we propose a new recovery scheme for SQLite, Delta-WAL. Delta-WAL is a recovery scheme based on logical logging, and reduces log size by storing only operation code and input values. In experiment, Delta-WAL generated smaller log compared to existing recovery schemes, and also showed improved transaction throughput.

Keywords
SQLite, recovery scheme, logical logging, DBMS

* 연세대학교 컴퓨터과학과 석사과정
** 연세대학교 컴퓨터과학과 통합과정
*** LG전자 MC 사업부 Chief Research Engineer
**** 연세대학교 컴퓨터과학과 교수(교신저자)
· 접 수 일: 2014년 10월 28일
· 수정완료일: 2014년 11월 17일
· 게재확정일: 2014년 11월 20일
· Received: Oct. 28, 2014, Revised: Nov. 17, 2014, Accepted: Nov. 20, 2014
· Corresponding Author: Sanghyun Park
Dept. of Computer Science, Yonsei University, 533-1, 3rd Engineering Building Sinchon-dong, Seodaemun-gu, Seoul-si, 690-756, Korea,
Tel.: +82 02 2123-7757, Email: sanghyun@cs.yonsei.ac.kr

Ⅰ. 서 론

우리는 매일 방대한 양의 데이터가 생성되고 디지털 형태로 저장되는 시대에 살고 있으며 이렇게 생성된 데이터를 효율적으로 관리하기 위해 데이터베이스 관리 시스템(DBMS, Database Management System)을 사용한다. DBMS란 상호 연관된 데이터와 그 데이터에 접근/가공하기 위한 프로그램들의 집합이다. DBMS의 주된 목표는 데이터베이스를 저장하고 획득하는데 있어 편리하고 효율적인 방식을 제공하는 것이며 대표적인 DBMS로 SQLite[1], MySQL[2], PostgreSQL[3], Oracle[4] 등이 있다.

이러한 DBMS은 하나의 논리적 일의 단위를 트랜잭션으로 묶어서 처리한다. 트랜잭션은 데이터의 무결성을 유지하기 위해 도입된 개념으로 DBMS가 처리하는 작업의 단위가 된다. DBMS는 트랜잭션이 안전하게 수행된다는 것을 보장하기 위해 표 1과 같은 ACID 속성을 만족시켜야 한다[5]. 그런데 DBMS은 동작 도중 여러가지 실패(디스크 충돌, 소프트웨어 오류, 전원 차단 등)로 인해 데이터베이스에 손실이 발생할 수 있다. 이러한 상황이 발생하면 DBMS는 트랜잭션의 원자성(Atomicity)과 지속성을 보장하기 위해 데이터 복구를 시도한다. 이 때 사용되는 복구 기법은 DBMS의 핵심 부분 중 하나로서, 데이터베이스가 실패 이전의 일관성 있는 상태로 돌아가는 것을 도와준다.

표 1. 트랜잭션의 ACID 속성
Table 1. ACID properties of the transaction

Property	Description
Atomicity	Either all operations of the transaction are reflected properly in the database, or none are
Consistency	Execution of a transaction in isolation preserves the consistency of the database
Isolation	Each transaction is unaware of other transactions executing concurrently in the system.
Durability	After a transaction completes, the changes it has made to the database persist, even if there are system failures.

SQLite는 임베디드 기기 및 스마트폰 뿐 아니라 소규모 데이터베이스를 운용하는 프로그램에서 주로 사용되는 DBMS이며 복구 기법으로 롤백 저널(Rollback Journal) 또는 Write-Ahead Logging(WAL)을 선택해서 사용할 수 있다. 그런데 롤백 저널과 WAL 모두 페이지 전체를 저장하기 때문에 생성되는 로그의 크기가 큰 편이며 이에 따라 로그를 파일을 관리하는 추가적 비용이 크다. 이러한 단점은 SQLite를 사용하는 스마트폰 어플리케이션의 반응속도를 저하시키는 주된 원인으로 지적된 바 있으며[6][7], 다양한 프로그램들이 SQLite를 사용하는 만큼 반드시 해결해야 할 문제 중 하나이다. 따라서 본 논문은 이러한 SQLite 복구 기법의 단점을 개선한 새로운 복구 기법 Delta-WAL(이하 DWAL)을 제안한다. DWAL은 논리적 로깅을 사용하여 저장되는 로그의 크기를 최소화하고, 로그 저장에 필요한 비용을 감소시킴으로서 전체 트랜잭션 처리 성능을 향상시킨다.

본 논문의 구성은 다음과 같다. 서론에 이어 다음 2장에서는 먼저 DBMS에서 사용하는 복구 기법의 종류에 대해서 소개하고 이어 SQLite의 복구 기법 및 파일 구조를 분석한다. 3장에서는 SQLite의 새로운 복구 기법인 DWAL에 대해 소개하고 4장에서 DWAL의 복구 시나리오를 검증한다. 5장에서는 DWAL과 다른 복구 기법을 비교한 실험결과에 대해 분석하고 마지막으로 6장에서 결론을 도출한다.

Ⅱ. 배경 및 관련 연구

2.1 복구 방식의 분류

DBMS의 복구 방식은 로그에 저장되는 내용에 따라서 몇 가지로 분류될 수 있다. 첫 번째는 물리적 로깅(Physical Logging)으로 이 방식은 변경되기 전의 페이지와 변경된 이후의 페이지를 통째로 저장한다. 변경되기 전의 페이지는 언두(Undo) 수행 시 필요하고, 변경된 이후의 페이지는 리두(Redo)에 사용된다. 물리적 로깅은 언두/리두 프로세스가 매우 빠르나 저장하는 로그의 크기가 크다는 단점이 있다.

두 번째 방식은 논리적 로깅(Logical Logging)으로

페이지 자체를 로그로 저장하지 않고 데이터에 수행되는 작업을 작업코드로 저장한다. 이 방식은 작업의 종류와 작업에 사용되는 입력값만 저장하므로 로그 파일의 크기가 물리적 로깅에 비해 매우 작다는 장점이 있다. 그러나 복구 시 각각의 작업을 직접 수행해야 되므로 수행속도가 느린 편이며 삭제, 업데이트 등 역 작업을 정의하기 힘든 작업에 대한 언두를 수행하기 어렵다.

마지막 방식으로 물리적 로깅과 논리적 로깅을 배합한 physiological logging이 있다. 이 기법은 변경되는 데이터 페이지의 이전 상태와 이후 상태를 저장하는 것이 아니라 실제로 변경이 발생하는 데이터에 대해서만 이전/이후의 상태를 저장하고 수행된 작업의 종류를 저장한다. 페이지 자체를 저장하지 않기 때문에 물리적 로깅에 비해 저장되는 로그의 양이 적으며 변경된 데이터의 이전/이후 상태를 보존하기 때문에 언두/리두 모두 손쉽게 수행할 수 있다.

2.2 SQLite

SQLite는 D.Richard Hipp가 개발한 관계형 데이터베이스 관리 시스템으로 2000년 8월에 최초의 버전이 공개되었다. SQLite는 ANSI-C로 짜여졌으며 별도의 서버 프로세스 없이 단독으로 운용된다. 작고 빠르기 때문에 임베디드 기기, 스마트폰 등에 탑재되며 간단한 정보를 저장하는 응용프로그램에서 주로 사용된다. 대표적인 프로그램으로는 사파리, 드롭박스, 크롬, 스카이프 등이 있다[8]. SQLite는 최초 공개 이후 꾸준한 업데이트를 통해 성능이 개선되었고 현재 최신 버전인 3.8.6에 이르렀다.

2.2.1 SQLite의 복구 방식

SQLite의 복구 방식은 저널링 파일 시스템[9][10]으로 구현되어 있으며 크게 롤백 저널과 WAL로 구분된다. 롤백 저널은 데이터베이스의 특정 페이지를 수정해야 할 경우, 원본 페이지를 먼저 백업하여 별도의 파일("<데이터베이스 파일명>-journal")에 따로 보관하고, 이후 페이지를 수정한다. 수정된 페이지는 트랜잭션이 커밋될 때 데이터베이스 파일에 완전히 반영되며 이때 보관하고 있던 저널파일이 삭제된다.

WAL은 롤백 저널과 반대의 특성을 갖는다. WAL은 특정 페이지를 수정해야 하는 경우, 수정한 페이지를 통째로 WAL 프레임 안에 넣고 이를 WAL("<데이터베이스 파일명>-WAL")파일에 저장한다. 이 후 수정된 페이지에 대한 읽기 요청이 들어오면 이를 데이터베이스 파일에서 직접 읽지 않고 WAL 파일에서 가져온다. WAL은 트랜잭션의 원자성을 보존하기 위해 (1) 특정 데이터 페이지가 디스크에 반영되기 전에 해당 로그 레코드(WAL 프레임)를 먼저 디스크에 쓰고 (2) 트랜잭션의 지속성을 만족시키기 위해 트랜잭션이 커밋되기 전에 관련된 모든 로그 레코드를 디스크에 쓴다. WAL은 커밋시 데이터베이스 파일 자체를 수정하지 않고 메모리상의 WAL 프레임을 WAL파일에 쓴다. 이때 쓰여진 WAL 프레임은 체크포인트 시 데이터베이스 파일에 반영된다. 페이지 자체를 통째로 로그에 저장한다는 점에 있어서 롤백 저널과 WAL 모두 물리적 로깅에 속한다고 할 수 있다.

2.2.2 SQLite 파일구조

SQLite는 여타 DBMS와 달리 데이터베이스 전체를 하나의 파일에 저장하여 관리하며 각 테이블을 변형된 B-트리 구조로 관리한다. SQLite의 파일은 그림 1과 같이 구성된다. SQLite의 파일은 여타 DBMS와 마찬가지로 여러 개의 페이지로 이루어지는데 페이지는 SQLite I/O의 최소 단위가 된다. 각 페이지는 앞에서부터 번호가 붙는데 1번 페이지는 데이터베이스의 메타데이터를 저장하고 있으므로 헤더 페이지라고 한다. 헤더 페이지의 첫 번째 100byte는 데이터베이스 헤더로 전반적인 데이터베이스에 대한 정보를 담고 있다[11]. 헤더 페이지의 끝에는 데이터베이스 테이블, 인덱스, 뷰, 트리거 등의 스키마 정보를 갖고 있는 sqlite_master 테이블이 존재하며 새로운 테이블 등이 생성될 때 이에 해당하는 메타데이터가 레코드 형태로 테이블에 삽입된다.

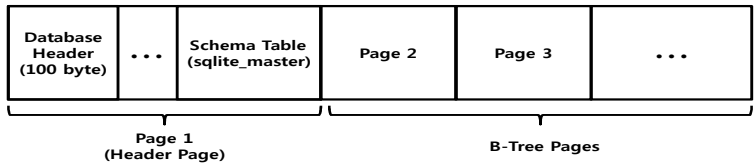
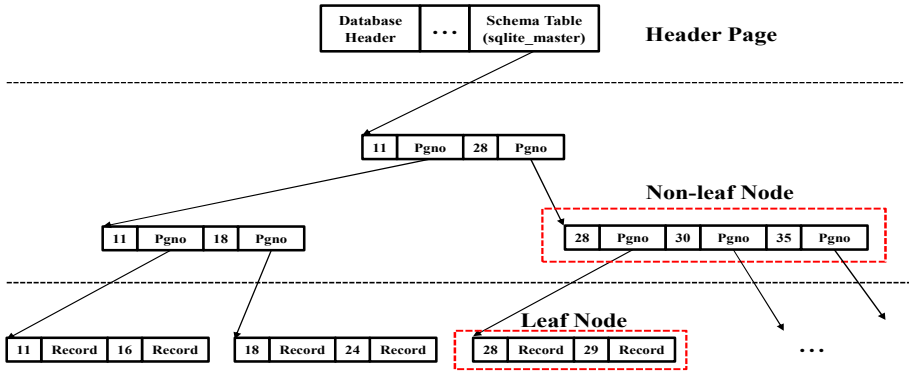
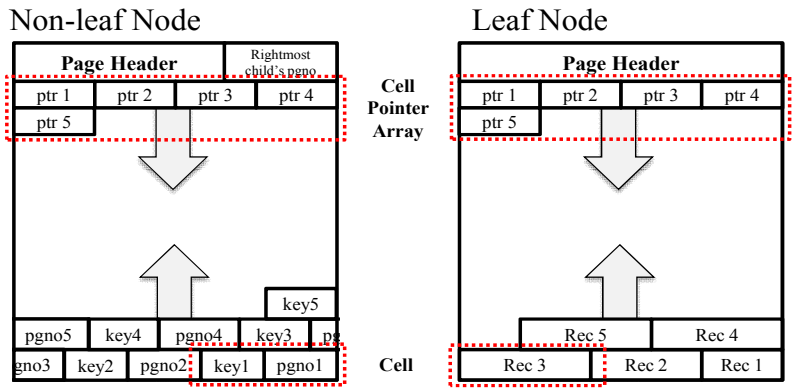


그림 1. SQLite 데이터베이스 파일의 구조
Fig. 1. Structure of SQLite database file



(a) 테이블 B-트리의 구조 (Structure of table B-tree)



(b) 테이블 B-트리 노드의 구조 (Structure of table B-tree node)

그림 2. SQlite의 테이블 B-트리
Fig. 2. Table B-tree of SQLite

SQLite의 테이블은 변형된 B-트리 구조로 설계되어 테이블 B-트리라 불리며 일반적으로 rowid라는 속성(Attribute)을 자동으로 생성하여 B-트리의 키(Key)로 삼는다. 앞서 설명한 헤더 페이지 이외의 다른 페이지는 B-트리의 노드가 되기 때문에 B-트리 페이지라고 한다. 테이블 B-트리의 구조는 그림 2(a)와 같은데 B+트리와 유사하게 non-leaf 노드에는 레코드를 저장하지 않고 leaf 노드에 각 레코드를 저장한다. 그림 2(b)는 테이블 B-트리 노드의 물리적인 구조를 나타낸 것이다. 각 노드를 나타내는 페이지는 slotted-page 구조[12]를 따르며 하나의 슬롯을 셀(Cell)이라고 표현한다. Non-leaf 노드의 각 셀

에는 해당 노드의 자식노드를 가리키는 페이지 번호와 키 값이 쌍으로 저장되고 가장 오른쪽 자식노드를 가리키는 페이지 번호는 페이지 헤더 내에 저장된다. 반면 leaf 노드의 셀에는 테이블의 레코드가 직접 저장된다. SQLite는 인덱스도 이와 같은 형태로 관리하나(인덱스 B-트리), non-leaf 노드에도 데이터(레코드의 row id)가 저장된다는 특징이 있다.

2.2.3 SQLite 관련연구

Jeong은 [13]에서 스마트폰의 I/O trace를 분석한 결과 전체 쓰기 요청 중 90%가 SQLite의 데이터베이스와 로그에 대한 것이며 SQLite는 로그를 파일에 쓸 때 많은 양의 임의쓰기를 발생시킨다는 점을 강조하였다. Kang은 플래시 메모리의 FTL인 X-FTL [14] 을 제안하였고 트랜잭션의 원자성과 지속성을 저장장치 레벨에서 직접 보장할 수 있도록 하였다. X-FTL을 사용함으로 SQLite는 따로 로그를 생성하지 않고 이에 따른 I/O 비용을 크게 절감할 수 있다. 단, X-FTL은 copy-on-write 특성을 활용하기 때문에 플래시 메모리 기반에서 유효하며 변경 이전의 페이지를 유지한다는 점에 있어서 DWAL과 접근방식이 다르다. 이외에도 SQLite에서 삭제된 데이터를 복구하는 연구가 수행되었다. [15][16]는 SQLite의 데이터 관리 규칙과 삭제된 데이터의 구조를 분석하여, 삭제된 이후 덮어써지지 않은 레코드를 복구하는 방법을 제시한다. 그러나 이러한 연구의 초점은 트랜잭션 및 데이터의 일관성을 유지하기 위함이 아니고 삭제된 레코드의 획득 그 자체에 있다.

III. Delta-WAL

DWAL은 페이지 전체를 저장하는 롤백 저널이나 WAL과는 달리, 페이지의 변경 이력을 논리적으로 기술하는 방식을 사용한다. 본 장에서는 DWAL의 구체적인 구조 및 알고리즘을 설명한다. 먼저 3.1에서는 DWAL에서 정의하는 논리적 작업들에 대해 설명한다. 3.2와 3.3은 각각 로그 레코드와 로그 페이지의 구조를 설명하며 3.4와 3.5에서는 DWAL의 알고리즘에 대해 설명한다. 마지막으로 3.6에서 DWAL이 기존 SQLite의 복구 방식에 대해 갖는 장점을 기술한다.

3.1 Delta-WAL의 논리적 작업

DWAL은 각 페이지마다 논리적 로깅을 하기 때문에 로그에 저장되는 논리적 작업에 대한 정의가 선행되어야한다. 따라서 본 논문은 아래 2가지 조건을 만족하는 논리적 작업의 집합을 표 2와 같이 정의한다.

표 2. DWAL의 논리적 작업
Table 2. Logical operation unit of DWAL

Opcode	SQLite Function	Location	Operation description
Cell_Insert	insertCell	btree.c	Insert cell into specific page
Cell_Drop	dropCell		Remove cell from specific page
Copy_Cells	copyNodeContent		Related to b-tree balancing
Assemble_Page	balance_nonroot balance_quick		Allocate memory for new page
Page_alloc	allocateBtreePage		Initialize page
Page_empty	zeroPage		add specific page to a free-list
Page_erase	freePage2		
*Page_stressed	pagerStress	pager.c	Write dirty page before transaction commits
Commit	sqlite3PagerCommitPhaseOne		Current transaction commits
Checkpoint	sqlite3PagerCheckpoint		Checkpoint operation
Hdr_change	various functions		Changes to database file header or database page header

1) 하나의 논리적 작업은 데이터베이스를 변경하는 SQLite의 작업 하나에 대응되어야 한다.
2) 데이터베이스를 변경하는 SQLite의 작업은 DWAL의 논리적 작업에 의해 표현될 수 있어야 한다.

DWAL의 논리적 작업은 크게 페이지 안의 셀과 관련된 작업(Cell_Insert, Cell_Drop), 페이지 자체에 관련된 작업(Page_alloc, Page_empty, Page_erase, Page_stressed), B-트리 밸런싱에 관련된 작업(Copy_Cells, Assemble_Page), 데이터베이스 헤더에 관한 작업, 커밋, 그리고 체크포인트로 나눌 수 있다. 이 중 Page_stressed는 다른 작업에서 나타나지 않는 없는 독특한 특징을 갖는데 이는 SQLite의 버퍼 관리 정책(Buffer Management Policy)과 깊은 관련이 있다. SQLite의 버퍼 관리자는 기본적으로 특정 트랜잭션에 의해 수정된 데이터베이스 페이지를 해당 트랜잭션이 커밋되기 전까지 데이터베이스 파일에 반영하지 않는다. 그러나 예외적으로 수정된 페이지의 수가 증가하여 이들이 차지하는 메모리가 SQLite가 정해놓은 한계를 넘어가게 되면 SQLite의 Pager Stress 함수에 의해 수정된 페이지가 트랜잭션이 커밋되기 전에 데이터베이스 파일에 쓰여질 수 있다. 따라서 트랜잭션 롤백 및 데이터 복구 시 이에 대한 처리가 반드시 필요하다. 이와 관련된 내용은 3.2에서 설명한다.

3.2 Delta-WAL의 로그 레코드

DWAL은 데이터베이스를 변경하는 논리적 작업이 시스템에 의해 수행되면 로그 레코드를 생성하고 이를 DWAL 페이지에 삽입한다. 로그 레코드는 하나의 논리적 작업을 리두하는데 필요한 데이터로 그림 3과 같이 구성되어 있다.

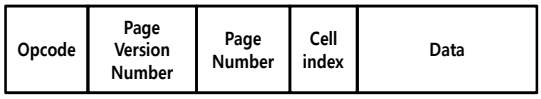


그림 3. DWAL의 로그 레코드 구조
Fig. 3. Structure of DWAL log record

Opcode는 표 2에서 정의한 작업 중 어떤 작업이 수행되었는지를 나타낸다. Page Number는 이 작업으로 인해 수정된 데이터베이스의 페이지 번호를, Cell index는 페이지 내의 몇 번째 셀에 관한 작업인지를 의미한다. Cell index 뒤의 Data는 Opcode의 특성에 따라 필요한 데이터를 저장한다. 대부분의 경우 표 2에 명시된 함수를 호출하는데 필요한 파라미터를 저장하지만 앞서 언급한 Page_stressed 작업에 대해서는 트랜잭션 시작 전에 존재하는 원본 페이지를 통째로 저장한다. SQLite는 앞서 언급한대로 커밋되지 않은 트랜잭션의 작업을 데이터베이스 파일에 반영하지 않기 때문에 논리적 로그 기반 복구를 할 때 로그를 언두할 필요가 없다. 단, Pager Stress에 한해 수정된 페이지가 커밋 전 데이터베이스 파일에 반영될 수 있고 이후 해당 트랜잭션이 롤백되면 미리 쓰여진 페이지 또한 트랜잭션 시작 전으로 돌아가야 한다. 해당 페이지에 연관된 모든 작업에 대해 언두를 수행하는 것은 비효율적이므로 이것을 방지하기 위해서 DWAL은 PagerStress 함수 호출 시 수정되기 전 원본 페이지를 로그 레코드의 Data 필드에 저장한다.

마지막으로 Page Version Number(PVN)는 복구 과정에서 해당 로그 엔트리가 데이터베이스 상에 이미 반영되어 있는지를 확인하기 위해 사용된다. PVN은 1부터 시작하여 트랜잭션이 커밋 될 때 수정된 페이지의 PVN이 1씩 증가한다. 단, 이를 위해서는 데이터베이스의 각 페이지도 PVN값을 저장해야 하는데 DWAL은 각 페이지마다 PVN을 저장할 수 있도록 4바이트 예약 공간을 할당한다.

3.3 Delta-WAL의 로그 페이지

DWAL 파일은 SQLite의 데이터베이스 파일과 마찬가지로 여러 개의 페이지가 모여서 이루어진다. 각각의 페이지는 slotted-page 구조로 구현되었는데 이는 그림 4와 같다. 페이지의 가장 앞부분은 페이지 헤더가 존재한다. 페이지 헤더에는 페이지 번호, 페이지 안의 로그 엔트리 개수, 사용 가능한 공간(lower, upper), 기타 플래그 등이 저장된다. 페이지 헤더 뒤의 포인터들은 페이지 끝에서부터 저장되어 있는 로그 엔트리의 시작 주소와 크기(offset, length)를 저장하고 있다. 따라서 Pointer n이 가리키고 있는 주소에서 해당 byte만큼을 가져옴으로써 n번째 로그 엔트리에 접근할 수 있다.

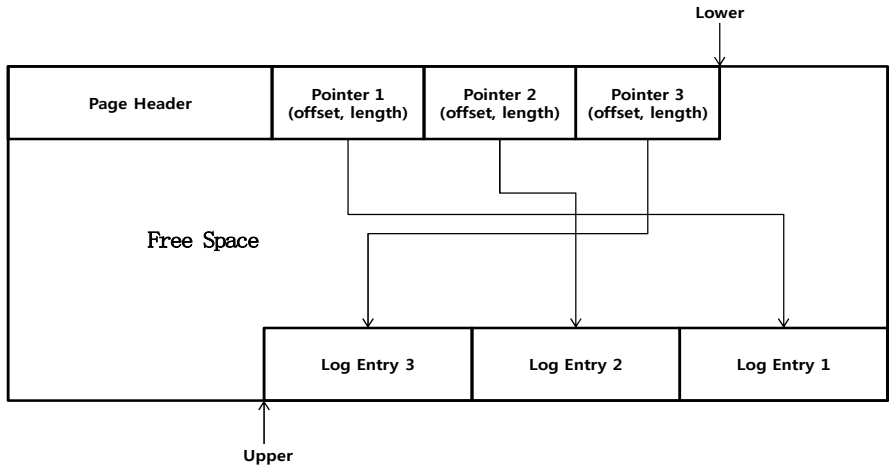


그림 4. DWAL의 로그 페이지 구조
Fig. 4. Structure of DWAL log page

마찬가지로 로그가 삽입될 때는 pointer의 끝에 새로운 포인터가 만들어지고 마지막 로그 엔트리가 저장된 주소 앞에 삽입될 로그 엔트리가 저장된다. 또한 DWAL은 로그 페이지의 내부 단편화를 최소화하기 위해 오버플로우 레코드를 지원한다. 삽입되는 로그 레코드가 로그 페이지의 빈 공간보다 클 경우 2개 이상의 로그 페이지에 걸쳐서 저장될 수 있으며 페이지 헤더의 오버플로우 플래그를 통해 이를 관리한다.

3.4 Delta-WAL의 커밋/체크포인트

DBMS에서 커밋과 체크포인트 연산은 트랜잭션에 의해 변경된 데이터베이스 페이지가 데이터베이스 파일에 완전히 반영되도록 한다. 복구 기법에 따라 커밋과 체크포인트에 수행되는 작업이 조금씩 다르므로 각 작업에 대한 정의가 필요하다.

Algorithm 1 : Commit algorithm of Delta-WAL

```
1 : Insert commit log record into log page
2 : Update log file header
3 : For each log page lp in memory
4 :     Write lp into log file
5:      Free lp
6 : Fsync log file
7 : For each dirty database page dp in memory
8 :     Write dp into database file
```

DWAL의 커밋 알고리즘은 알고리즘 1과 같다. DWAL은 먼저 커밋 로그 레코드를 생성하여 로그 페이지에 삽입한다(알고리즘 1의 줄1). 이후 로그 파일 헤더를 업데이트 하고(줄2) 메모리에 존재하는 모든 로그 페이지를 로그 파일("<데이터베이스 파일명>-DWAL")에 쓴뒤 이를 메모리에서 제거한다(줄3-5). 이 때 로그 페이지들이 실제로 로그 파일에 완전히 써지지 않고 운영체제의 os캐시에 남아있을 수 있으므로 fsync 명령을 호출하여 모든 로그 페이지가 확실하게 로그 파일에 반영되도록 한다(줄6). 로그파일에 대한 처리가 끝나면 커밋된 트랜잭션에 의해 수정된 모든 데이터베이스 페이지를 데이터베이스 파일에 쓴다(줄 7-8). DWAL은 체크포인트 알고리즘은 다음과 같다. 우선 체크포인트 로그 레코드를 생성하여 로그 페이지에 삽입한 후, 로그페이지를 로그파일에 쓴다. 이후 레코드가 로그파일에 확실하게 써졌음을 보장하기 위해 fsync 명령을 호출하고, 이 후 데이터베이스 파일에 대한 fsync 명령을 호출한다.

DWAL은 트랜잭션 커밋 시점에 데이터베이스 파일에 대한 fsync 명령을 호출하지 않는다. Fsync 호출이 되지 않으므로 트랜잭션에 의해 변경된 사항이 데이터베이스 파일에 완전히 반영되지 않았을 가능성이 존재하나 트랜잭션에 의해 수행된 작업들은 모두 확실하게 로그 파일에 기록된다. 따라서 데

이터베이스 파일에 대한 fsync 호출 전에 시스템이 비정상적으로 종료되더라도 로그 파일의 로그 레코드를 통해 데이터를 복구할 수 있다.

3.5 Delta-WAL의 복구 알고리즘

DWAL은 SQLite에 의해 데이터베이스 파일이 열릴 때, 데이터베이스 파일에 대한 로그 파일이 존재하는지 확인한다. 로그 파일이 존재하면 알고리즘 2의 과정을 통해 데이터베이스에 대한 복구를 시도한다. 먼저 DWAL 로그 파일의 마지막 레코드부터 마지막 커밋 레코드까지 역으로 탐색(Backward Traversal)하면서 작업코드가 Page_stressed인 로그가 있는지 확인한다(알고리즘 2의 1-2줄). 해당 로그가 존재하면 커밋되지 않은 트랜잭션에 의해 수정된 페이지가 데이터베이스 파일에 써졌을 수 있다. 따라서 로그 레코드에 저장된 원본 페이지를 데이터베이스 파일에 덮어쓴다(줄3). 이후, 마지막 체크포인트 레코드까지 역으로 탐색하면서 로그 레코드의 PVN과 로그 레코드가 가리키는 데이터베이스 페이지의 PVN을 비교한다(줄4-5). 만약 로그 레코드의 PVN이 더 크면 해당 로그 레코드는 데이터베이스에 반영되어 있지 않으므로 이를 리두 테이블에 저장한다(줄6). 역 탐색이 끝난 이후에는 다시 마지막 커밋 레코드까지 정 탐색(Forward Traversal)을 하면서 해당 로그 레코드가 리두 테이블 안에 있는지 확인하고, 로그 레코드에 저장된 데이터를 바탕으로 표 2에 명시된 SQLite 함수를 호출하여 리두를 수행한다(줄 7-9).

Algorithm 2 : Recovery algorithm of Delta-WAL

```
1 : For each log record /from end of record to the
    last commit record // backward traversal
2 :   If /.opcode == Page_stressed
3 :     Overwrite original page back to database file
4 : For each log record /from last commit record to
    the last checkpoint record // backward traversal
5 :   If /.PVN > database page (/.pgno)th's PVN
6 :     insert / into redo table
7 : For each log record /from last checkpoint record
    to the last commit record // forward traversal
8 :   If / is in redo table
9 :     Redo log /
```

3.6 Delta-WAL의 장점

DWAL은 논리적 로깅을 하기 때문에 물리적 로깅을 하는 롤백 저널 및 WAL에 비해 로그의 크기가 작고 따라서 로그를 관리하는데 발생하는 비용 또한 적다. 추가적으로 WAL은 체크포인트 시 WAL 파일에 저장되어 있던 페이지들을 한꺼번에 데이터베이스로 반영하기 때문에 트랜잭션이 커질수록 체크포인트 수행 시간이 길어져 ANR (Application Not Responding)이 발생할 가능성이 높다. 반면에 DWAL은 체크포인트 시 데이터베이스 파일에 대한 싱크(fsync)만 호출하면 되기 때문에 체크포인트 비용이 적다.

IV. Delta-WAL의 복구 시나리오

트랜잭션은 여러 가지 내외부적 원인에 의해 실행이 중단될 수 있는데 이러한 경우라도 데이터베이스 시스템은 트랜잭션의 ACID 속성을 보장해야 한다. 본 장에서는 트랜잭션이 중단될 수 있는 각종 상황과 각 상황별 DWAL의 데이터 복구 시나리오를 소개한다.

4.1 트랜잭션 실패(Transaction Failure)

실행중인 트랜잭션이 특정한 원인에 인해 더 이상 수행될 수 없는 경우 트랜잭션 실패가 발생한다. 트랜잭션 실패는 오버플로우, 자원 부족, 교착상태 등으로 인해 발생하는데 이러한 경우 해당 트랜잭션은 롤백되어 트랜잭션 시작 직전의 상태로 되돌아가야 한다.

DWAL은 트랜잭션이 롤백될 때, 로그 상 마지막으로 저장된 레코드부터 이전 커밋 레코드까지 역 탐색을 한다. 탐색 도중 Pager_stressed 로그를 만나는 경우 원본 페이지를 데이터베이스 페이지에 덮어쓴다(알고리즘 2의 1-3줄). Pager_stressed 로그 이외의 로그는 데이터베이스에 아직 반영되기 전이므로 무시한다. 커밋 레코드에 도달하면 커밋 레코드 이후의 모든 로그 레코드를 삭제한다.

4.2 시스템 충돌(System Crash)

시스템 충돌은 하드웨어 오작동, 데이터베이스 소프트웨어 또는 운영체제의 버그 등으로 인해 휘발성 저장장치의 데이터 손실이 발생하고 트랜잭션 수행을 중단되는 것을 뜻한다. 단, 비휘발성 저장장치의 데이터베이스 파일 및 로그 파일은 손실되지 않는 것을 가정한다. 시스템 충돌이 발생한 시점에 따라 데이터베이스 파일 및 로그 파일의 상태가 다르기 때문에 복구 기법은 각 상황에 유동적으로 대처할 수 있어야 한다. 3.5에서 소개한 DWAL의 복구 알고리즘은 시스템 충돌에 의해 발생하는 다음과 같은 상황에 대처할 수 있다.

1) 로그 파일에 시스템 충돌 시 수행 중이던 트랜잭션의 커밋 로그가 존재하지 않음

아직 트랜잭션에 의해 수정된 페이지가 데이터베이스 파일에 반영되기 이전이며 커밋 로그가 없으므로 해당 트랜잭션이 수행되기 이전 상태로 돌아가야 한다. 따라서 4.1과 같이 Pager_stressed 로그에 대한 처리(알고리즘 2의 1-3줄)를 하고 트랜잭션과 관련된 로그를 삭제한다.

2) 로그 파일에 시스템 충돌 시 수행 중이던 트랜잭션의 커밋 로그가 존재

로그 파일에 트랜잭션을 리두하기 위한 모든 데이터가 들어있으나 데이터베이스 파일에는 트랜잭션에 의해 변경된 사항이 전부 반영되었는지 알 수 없다. 따라서 DWAL 복구 알고리즘(알고리즘 3)을 수행하면서 해당 트랜잭션과 관련된 로그가 데이터베이스 페이지에 반영되었는지 확인하고, 반영되지 않은 로그에 한해 리두를 수행한다.

4.3 디스크 실패(Disk Failure)

디스크 자체의 손상으로 인해 디스크 블록의 데이터가 소실될 수도 있다. SQLite의 모든 복구 기법은 데이터베이스 파일과 로그 파일이 같은 경로에 저장되며 따라서 동일한 디스크에 저장 된다. 때문에 디스크가 손상되어 데이터가 소실 될 경우 해당 데이터를 복구할 수 없다. DWAL 또한 로그 파일을 기반으로 체크포인트 이후의 작업에 대해서 복구 작업을 수행한다. 따라서 로그 파일 페이지 또는 데이터베이스 파일 자체가 깨진 경우 데이터 복구가 어렵다.

V. 실험결과

앞에서 기술한 DWAL의 구조를 토대로 이를 SQLite의 소스코드를 수정하여 이를 구현하였으며 기존의 롤백 저널, WAL과 비교를 통해 성능을 측정하였다.

5.1 실험환경

실험은 표 3과 같은 환경에서 수행되었으며 비교적 적은 수의 쿼리로 구성된 단일 트랜잭션과 여러 개의 복잡한 트랜잭션의 조합으로 구성된 TPC-C [17] 워크로드를 실행하였다. 단일 트랜잭션의 경우 임의의 테이블을 생성하여 사용하였고 TPC-C 워크로드에서는 TPC-C 벤치마크에 정의된 테이블 및 규칙을 따르는 데이터를 사용하였다.

표 3. 실험에 사용된 시스템 환경
Table 3. System configuration used in experiments

SQLite Ver.	3.7.13 codebase(modified)
OS	Linux(ubuntu 12.04 LTS 64bit)
Processor	AMD phenom(tm) II X6 1075T
Memory	16 GB
Storage	WDC WD10EALX-009BA0 1.0TB
Page size	4 KB Fixed

5.2 단일 트랜잭션

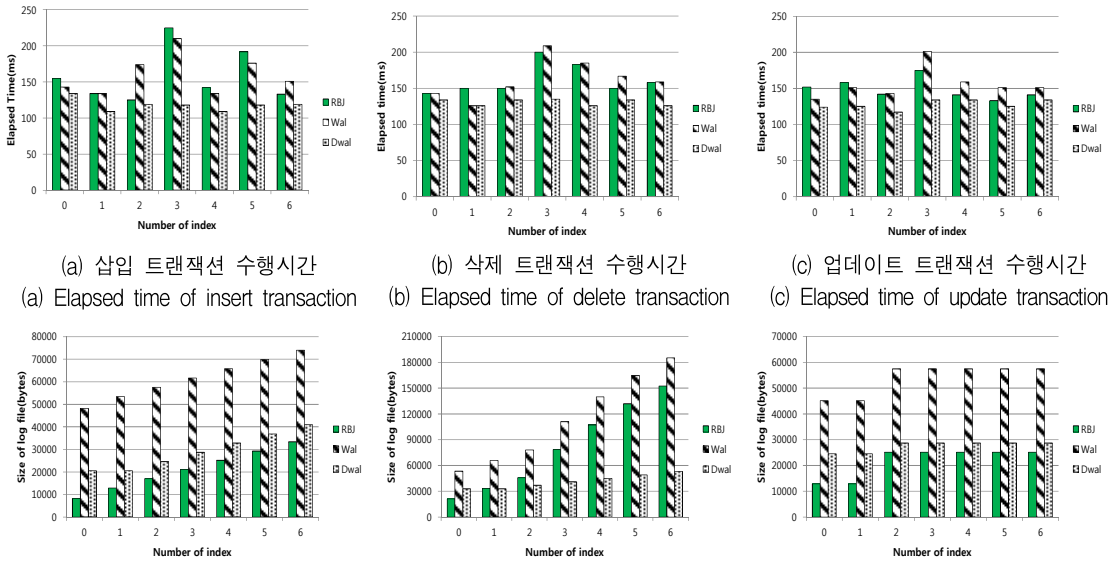
DWAL의 트랜잭션 실행 성능을 알아보기 위해 각각 삽입, 삭제, 업데이트 쿼리로 구성된 3가지 트랜잭션을 생성하였다. 삽입 트랜잭션은 빈 테이블에 100개의 레코드를 삽입하고 삭제 트랜잭션은 1000개의 레코드가 들어있는 테이블에서 100개의 레코드를 삭제한다. 업데이트 쿼리는 마찬가지로 1000개의 레코드 중 100개 레코드의 특정 속성값을 수정한다. 트랜잭션에 사용된 테이블은 임의로 생성된 테이블로 기본키를 포함한 3개의 정수, 2개의 가변길이 문자열, 2개의 실수 속성으로 이루어졌다. 테

이블의 기본키를 제외한 각 속성에 인덱스를 추가하면서 그림 5와 같이 트랜잭션 실행시간 및 생성되는 최종 로그의 크기를 측정하였다.

각 트랜잭션에서 WAL과 롤백 저널의 실행시간은 인덱스의 개수에 따라 증가하다가 3개 이후 감소하는 경향을 보였으며 DWAL의 실행시간은 인덱스의 개수와 크게 상관없이 일정하였다. 로그의 크기는 트랜잭션 별로 조금 다른 결과가 나타났다. 삽입 트랜잭션에서 로그의 크기는 WAL이 다른 두 기법보다 2배 이상 더 컸으며 롤백 저널의 로그파일이 근소한 차이로 DWAL의 로그 파일보다 작았다. 삭제 트랜잭션에서는 롤백 저널과 WAL의 로그파일은 인덱스 수에 비례해서 증가하였고 DWAL의 로그파일은 거의 증가하지 않았다. 마지막으로 업데이트 트랜잭션은 삽입 트랜잭션과 비슷한 경향을 띄었다.

SQLite에서 인덱스는 테이블과 마찬가지로 B-트리 형태로 구성된다. 인덱스가 존재하는 테이블에 어떤 변화가 발생하면 테이블 B-트리가 수정되고 이와 동시에 인덱스 B-트리도 수정된다. 따라서 인덱스의 개수가 많을수록 하나의 SQL문에 의해 수정되는 B-트리의 수가 증가한다. 이러한 이유로 페이지 단위로 로그를 저장하는 롤백 저널과 WAL은 인덱스 개수가 늘어날수록 저장해야 하는 페이지 수가 더 많아로그 파일 또한 커졌다.

롤백 저널은 다른 기법과 다르게 트랜잭션의 종류에 따라 로그의 크기가 변동하였다. 이는 원본 페이지를 저장하는 롤백 저널의 특징에 기인한 결과이다. 삭제 트랜잭션의 경우 특정 페이지의 레코드가 일정 수 이하로 떨어지면 언더플로우가 발생하여 주변 페이지와 합병하게 된다. 따라서 한 페이지의 내의 변화가 다른 페이지로 전파되며 전파된 페이지의 원본 또한 저장해야 하므로 삭제 트랜잭션에서의 로그 크기가 가장 컸다. 반면 입력 및 업데이트 트랜잭션에서는 일반적으로 페이지 내의 변화가 다른 페이지에 영향을 주지 않아 로그의 크기가 DWAL보다도 작았다. 반면 WAL은 변경되는 페이지를 저장하고, 저장된 페이지에 쉽게 접근하기 위해 추가적인 인덱스를 구축하기 때문에 로그의 크기가 가장 컸다. DWAL은 논리적 로깅 덕분에 평균적으로 로그 파일의 크기가 가장 작았으며 이에 가장 빠른 트랜잭션 실행 성능을 보였다.



(a) 삽입 트랜잭션 수행시간
(a) Elapsed time of insert transaction

(b) 삭제 트랜잭션 수행시간
(b) Elapsed time of delete transaction

(c) 업데이트 트랜잭션 수행시간
(c) Elapsed time of update transaction

(d) 삽입 트랜잭션 로그 크기
(d) Log size of insert transaction

(e) 삭제 트랜잭션 로그 크기
(e) Log size of delete transaction

(f) 업데이트 트랜잭션 로그 크기
(f) Log size of update transaction

그림 5. 단일 트랜잭션 성능
Fig. 5. Performance of single transaction

5.3 TPC-C 워크로드

DWAL이 실제 워크로드(real workload) 실행에도 적합한지 알아보는 실험도 수행하였다. TPC-C 공식 문서에 정의된 8개의 테이블을 만들고 scale factor 1로 테이블 안에 레코드를 삽입하여 약 350MB 크기의 데이터베이스 파일을 생성하였다. TPC-C는 총 5개의 트랜잭션(New Order, Payment, Delivery, Order Status, Stock Level)을 정의하는데 이 중 Order Status와 Stock Level 두 트랜잭션은 읽기 전용 트랜잭션이다. 읽기 전용 트랜잭션은 실행 시 로그가 생성되지 않으므로 두 트랜잭션의 실행 비율은 최소화하였고, 쓰기가 포함된 나머지 트랜잭션의 실행 비율을 높여 쓰기 집중적인 워크로드를 생성하였다.

정확한 실행 비율은 표 4와 같으며 전체 워크로드를 10분간 실행하여 그림 6과 같이 초당 실행되는 트랜잭션 수를 측정하였다. 그림 6에서 읽기 전용 트랜잭션 2개는 수치가 너무 높아 생략하였으며 이 2개의 트랜잭션은 나머지 3개의 트랜잭션과 함께 Overall에 포함시켰다.

표 4. TPC-C 워크로드
Table 4. TPC-C workload

Transaction	Ratio
Delivery	4%
Order Status	4%
Payment	43%
Stock Level	4%
New Order	45%

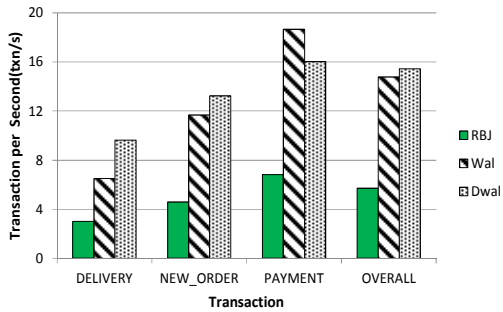


그림 6. TPC-C 워크로드 실행결과
Fig. 6. Result of TPC-C workload

TPC-C 워크로드와 같은 무거운 워크로드에서도 DWAL은 우수한 성능을 보였다. DWAL은 3개의 쓰기 트랜잭션 중 DELIVERY, NEW_ORDER 2개의 트랜잭션에서 가장 좋은 성능을 보였으며 PAYMENT 트랜잭션에서는 가장 높은 성능의 WAL과 근소한 차이를 보였다. 읽기 전용 트랜잭션 2개를 포함해서 5개의 트랜잭션을 종합한 결과(그림 6의 Overall) DWAL의 성능이 가장 우수하였다.

VI. 결 론

본 논문은 오픈소스 DBMS인 SQLite를 위한 새로운 복구 기법인 DWAL을 제안하였다. DWAL은 기존 SQLite의 롤백 저널, WAL과 다르게 논리적 로깅을 함으로써 저장되는 로그의 크기를 줄이고 트랜잭션 처리 성능을 향상시킨다.

DWAL은 단일 트랜잭션과 TPC-C 워크로드모두에서 롤백 저널과 WAL보다 우수한 트랜잭션 처리 성능을 보였다. 또한 DWAL은 삭제 트랜잭션에서 다른 기법에 비해 가장 작은 로그를 생성하였으며 다른 트랜잭션에서도 롤백 저널과 비슷한 수준의 로그를 생성하였다. 마지막으로 DWAL은 다른 기법에 비해 인덱스 증가에 덜 민감하였고 안정적인 성능을 보였다. 종합적으로 DWAL은 트랜잭션 처리 성능 및 로그 크기 모두 기존의 기법보다 우수한 것으로 나타났다.

SQLite는 각종 응용 프로그램에 사용되며 점점 그 사용량이 증가하는 추세이다. 이러한 SQLite에 본 논문에서 제시한 DWAL을 적용함으로써 데이터베이스 처리 성능을 개선하고 이를 사용하는 어플리케이션의 응답 속도 및 사용자 경험(UX) 향상을 기대해 볼 수 있을 것이다.

References

[1] SQLite, http://sqlite.org/
[2] MySQL, http://www.mysql.com/
[3] PostgreSQL, www.postgresql.org/
[4] Oracle Database, http://docs.oracle.com/cd/E11882_01/server.112/e25789/intro.htm
[5] S. Abraham, K. Henry, and S. Sudarshan, "*Database System Concepts 6th edition*", McgrewHill, pp.

628, 2011.

[6] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones", In Proceedings of USENIX Conference on File and Storage Technologies, 2012.

[7] K. Lee and Y. Won, "Smart Layers and Dumb Result: IO Characterization of an Android-Based Smartphone", In Proceedings of ACM EMSOFT, pp. 23-32, 2012.

[8] Well-Known Users of SQLite, http://www.sqlite.org/famous.html

[9] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis and Evolution of Journaling File Systems", In proceedings of USENIX Annual Technical Conference, pp. 105-120, 2005.

[10] D. Woodhouse, "JFFS : The Journalling Flash File System", In Proceedings of the Ottawa Linux Symposium, 2001.

[11] SQLite Database Header, http://www.sqlite.org/fileformat2.html#database_header

[12] S. Abraham, K. Henry, and S. Sudarshan, "*Database System Concepts 6th edition*", Mcgrew Hill, pp. 1147, 2011.

[13] S. Jeoung, K. Lee, S. Lee, S. Son, and Y. Won, "I/O Stack Optimization for Smartphones", 2013 USENIX Annual Technical Conference, 2013.

[14] W. Kang, S. Lee, and B. Moon, "*X-FTL: Transactional FTL for SQLite Databases*", SIGMOD'13, 2013.

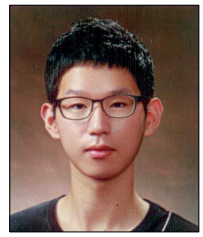
[15] S. Jeon, J. Bang, K. Byun, and S. Lee, "A recovery method of deleted record for SQLite database", Personal and Ubiquitous Computing, Vol. 16, Issue 6, pp. 707-715, Aug. 2012.

[16] Gyu-Won Lee, Seung-Jei Yang, Hyun-Uk Hwang, Kibom Kim, Taejoo Chang, and Ki-Wook Sohn, "A Recovery Scheme for the Deleted Overflow Data in SQLite Database", Journal of KIIT, Vol. 10, No. 11, pp. 143-153, Nov. 2012.

[17] Transaction Processing Performance Council, TPC BENCHMARK™ C Standard Specification Revision 5.11, 2010.

저자소개

이 준 희 (Joonhee Lee)



2013년 : 연세대학교 컴퓨터과학과 졸업(학사)
2013년 ~ 현재 : 연세대학교 컴퓨터과학과 석사과정
관심분야 : 데이터베이스 시스템, 플래시SSD, 빅 데이터

신 민 철 (Mincheol Shin)

2011년 : 연세대학교 컴퓨터과학과 졸업(학사)
2011년 ~ 현재 : 연세대학교 컴퓨터과학과 석박통합과정
관심분야 : 데이터베이스 시스템, 분산처리 시스템, 빅 데이터

장 용 일 (Yongil Jang)

2007년 : 인하대학교 컴퓨터공학과(박사)
2007년 ~ 2008년 : 텔코웨어
2008년 ~ 현재 : LG전자
관심분야 : 모바일, 데이터베이스

박 상 현 (SangHyun Park)

1989년 : 서울대학교 컴퓨터공학과 졸업(학사)
1991년 : 서울대학교 대학원 컴퓨터공학과(공학석사)
2001년 : UCLA 대학원 컴퓨터과학과(공학박사)
2002년 ~ 2003년 : 포항공과대학교 컴퓨터공학과 조교수
2003년 ~ 2006년 : 연세대학교 컴퓨터과학과 조교수
2006년 ~ 2011년 : 연세대학교 컴퓨터과학과 부교수
2011년 ~ 현재 : 연세대학교 컴퓨터과학과 교수
관심분야 : 데이터베이스, 데이터마이닝, 바이오인포매틱스, 적응적 저장장치 시스템