(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0054767
(43) 공개일자 2017년05월18일

(51) 국제특허분류(Int. Cl.)
G06F 17/30 (2006.01) *G06F 11/14* (2006.01)
(52) CPC특허분류
G06F 17/30289 (2013.01)
G06F 11/1435 (2013.01)
(21) 출원번호 10-2015-0157340
(22) 출원일자 2015년11월10일
심사청구일자 없음

(71) 출원인
엘지전자 주식회사
서울특별시 영등포구 여의대로 128 (여의도동)
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
장용일
서울특별시 서초구 양재대로11길 19 LG전자 특허센터
김효성
서울특별시 서초구 양재대로11길 19 LG전자 특허센터
(뒷면에 계속)
(74) 대리인
김용인, 방해철

전체 청구항 수 : 총 10 항

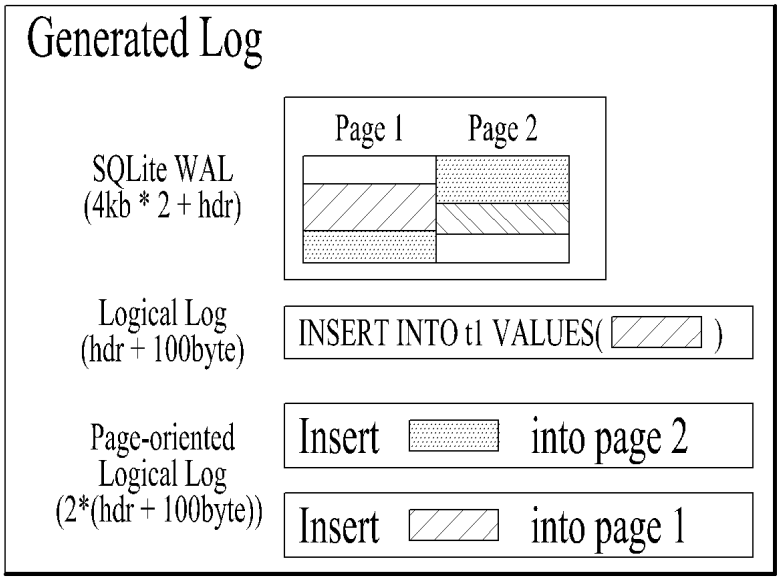
(54) 발명의 명칭 **데이터베이스 관리 시스템 및 그의 데이터 변경 및 복구 방법**

(57) 요 약

데이터를 변경 및 복구할 수 있도록 하는 데이터베이스 관리 시스템 및 및 그의 데이터 변경 및 복구 방법에 관한 것으로, 페이지 단위로 구성된 데이터베이스 파일을 저장하는 데이터베이스와, 트랜잭션(transaction)이 수행된 데이터베이스 파일에 대한 로그 파일을 생성하는 로그 파일 생성부와, 데이터베이스 파일의 트랜잭션을 수행

(뒷면에 계속)

대 표 도 - 도5



하는 트랜잭션 수행부와, 데이터베이스, 로그 파일 생성부 및 트랜잭션 수행부를 제어하는 제어부를 포함하고, 제어부는, 데이터베이스 파일의 트랜잭션이 요청되면, 데이터베이스 파일의 트랜잭션을 수행하도록, 트랜잭션 수행부를 제어하고, 데이터베이스 파일의 트랜잭션이 수행되면, 데이터베이스 파일의 페이지들 중, 내용이 변경된 더티 페이지(dirty page) 별로, 변경 내역에 대한 로그 레코드(log record)를 생성하고, 데이터베이스에 트랜잭션이 커밋(commit)되면, 생성된 전체 로그 레코드를 로그 파일에 저장하도록, 로그 파일 생성부를 제어할 수 있다.

(52) CPC특허분류
G06F 11/1448 (2013.01)
G06F 17/30227 (2013.01)
G06F 17/30368 (2013.01)

(72) 발명자
박상현
서울특별시 양천구 오목로 300, 204동 3701호 (목동, 현대하이페리온2)
신민철
인천광역시 동구 송현로 50, 107동 1702호 (송현동, 솔빛마을 주공아파트)
이준희
서울특별시 서초구 남부순환로 2343-10, 503동 602호 (서초동, 서초3차대림이편한세상)

명 세 서

청구범위

청구항 1

페이지 단위로 구성된 데이터베이스 파일을 저장하는 데이터베이스;

트랜잭션(transaction)이 수행된 데이터베이스 파일에 대한 로그 파일을 생성하는 로그 파일 생성부;

상기 데이터베이스 파일의 트랜잭션을 수행하는 트랜잭션 수행부; 그리고,

상기 데이터베이스, 로그 파일 생성부 및 트랜잭션 수행부를 제어하는 제어부를 포함하고,

상기 제어부는,

상기 데이터베이스 파일의 트랜잭션이 요청되면, 상기 데이터베이스 파일의 트랜잭션을 수행하도록, 상기 트랜잭션 수행부를 제어하고,

상기 데이터베이스 파일의 트랜잭션이 수행되면, 상기 데이터베이스 파일의 페이지들 중, 내용이 변경된 더티 페이지(dirty page) 별로, 변경 내역에 대한 로그 레코드(log record)를 생성하고, 상기 데이터베이스에 상기 트랜잭션이 커밋(commit)되면, 상기 생성된 전체 로그 레코드를 상기 로그 파일에 저장하도록, 상기 로그 파일 생성부를 제어하는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 2

제 1 항에 있어서, 상기 제어부는,

상기 생성된 전체 로그 레코드를 상기 로그 파일에 저장할 때,

상기 생성된 전체 로그 레코드의 크기가 상기 더티 페이지의 크기보다 더 작으면, 상기 생성된 전체 로그 레코드를 상기 로그 파일에 저장하고,

상기 생성된 전체 로그 레코드의 크기가 상기 더티 페이지의 크기보다 더 크면, 상기 더티 페이지를 상기 로그 파일에 저장하도록, 상기 로그 파일 생성부를 제어하는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 3

제 1 항에 있어서, 상기 로그 파일은,

로그 페이지 단위로 구성되고,

상기 로그 페이지는,

로그 페이지 정보를 포함하는 로그 페이지 헤더, 각 로그 레코드의 위치 및 길이 정보를 포함하는 로그 포인터, 로그 데이터를 포함하는 로그 레코드로 구성되는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 4

제 3 항에 있어서, 상기 로그 페이지 헤더는,

상기 로그 페이지의 시작 지점에 저장되고,

상기 로그 포인터는,

상기 로그 레코드가 상기 로그 페이지에 삽입되는 순서에 따라, 상기 로그 페이지 헤더로부터 상기 로그 페이지의 끝 지점 방향으로 순차 저장되며,

상기 로그 레코드는,

상기 로그 레코드가 상기 로그 페이지에 삽입되는 순서에 따라, 상기 로그 페이지의 끝 지점으로부터 상기 시작 지점 방향으로 순차 저장되는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 5

제 3 항에 있어서, 상기 로그 페이지 헤더는,

상기 로그 페이지의 시작 지점에 저장되고,

상기 로그 포인터와 로그 레코드는,

상기 로그 레코드가 상기 로그 페이지에 삽입되는 순서에 따라, 상기 로그 페이지 헤더로부터 상기 로그 페이지의 끝 지점 방향으로 순차 저장되는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 6

제 3 항에 있어서, 상기 제어부는,

상기 로그 페이지 내에 삽입되는 상기 로그 레코드의 개수가, 상기 로그 페이지의 빈 공간 크기를 초과하면, 새로운 추가 로그 페이지를 생성하고,

상기 로그 페이지 헤더 내에 오버플로우 플래그를 추가하도록, 상기 로그 파일 생성부를 제어하는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 7

제 1 항에 있어서, 상기 제어부는,

상기 로그 레코드 생성 시,

상기 더티 페이지에 대한 변경 이력을 확인하고, 상기 더티 페이지에 변경 이력이 존재하면, 상기 더티 페이지의 변경 내역에 대한 로그 레코드를 생성하여, 상기 생성된 전체 로그 레코드를 상기 로그 파일에 저장하고,

상기 더티 페이지에 변경 이력이 존재하지 않으면, 상기 더티 페이지의 변경 내역에 대한 로그 레코드 생성 없이, 상기 더티 페이지 전체를 상기 로그 파일에 저장하도록, 상기 로그 파일 생성부를 제어하는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 8

제 1 항에 있어서, 상기 제어부는,

상기 트랜잭션이 커밋될 때,

상기 로그 파일의 커밋 로그 레코드(commit log record)에 체크섬(checksum)을 저장하는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 9

제 8 항에 있어서, 상기 제어부는,

상기 데이터베이스 파일의 복구 시에, 상기 체크섬과 상기 로그 레코드의 내용을 비교하고,

상기 체크섬과 상기 로그 레코드의 내용이 다르면, 해당하는 트랜잭션을 실패로 판정하고, 재수행(redo) 과정에서 상기 로그 레코드를 제외시키는 것을 특징으로 하는 데이터베이스 관리 시스템.

청구항 10

트랜잭션(transaction)이 수행된 데이터베이스 파일에 대한 로그 파일을 생성하는 데이터베이스 관리 시스템의 데이터 복구 방법에 있어서,

데이터 복구 요청을 수신하는 단계;

복구하고자 하는 데이터베이스 파일에 대한 로그 파일의 존재를 확인하는 단계;

상기 로그 파일이 존재하면, 상기 로그 파일의 마지막 로그 레코드로부터 마지막 커밋 로그 레코드까지 역순서로 검색하여, 상기 복구하고자 하는 데이터베이스 파일에 대한 원본 페이지가 존재하는지를 확인하는 단계;

상기 원본 페이지가 존재하면, 상기 원본 페이지를 상기 데이터베이스 파일에 저장하는 단계;

상기 로그 파일의 마지막 체크포인트 로그 레코드로부터 마지막 커밋 로그 레코드까지 정순서로 검색하여, 그들 사이에 존재하는 로그 레코드를 재수행하는 단계; 그리고,

상기 로그 파일의 마지막 커밋 로그 레코드 이후에 존재하는 로그 레코드를 삭제하는 단계를 포함하는 것을 특징으로 하는 데이터베이스 관리 시스템의 데이터 복구 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 데이터를 변경 및 복구할 수 있도록 하는 데이터베이스 관리 시스템 및 및 그의 데이터 변경 및 복구 방법에 관한 것이다.

배 경 기 술

[0002] 일반적으로, 데이터베이스 관리 시스템(database management system)은, 기본적으로 전력 이상이나 시스템 이상으로 문제가 생겼을 때, 데이터의 일관성을 유지할 수 있도록, 고유의 로깅(logging) 방식과 로그(log)를 통한 데이터 복구 방법을 지원할 수 있다.

[0003] 따라서, 데이터베이스 관리 시스템은, 트랜잭션(transaction), 프로그램, 시스템 등의 실패(failure)로 인하여 데이터가 손실될 경우, 데이터를 복구해 줄 수 있는 매커니즘을 포함할 수 있다.

[0004] 데이터베이스 관리 시스템의 데이터 복구 방식은, 다양한 방식들을 가지고 있는데, 특히, 이동 단말기에서 사용되는 다양한 어플리케이션들은, RBJ(Rollback Journal) 방식이나 WAL(Write Ahead Log) 방식 등을 사용하고 있다.

[0005] 여기서, RBJ 방식은, 쓰기 연산이 발생하는 경우, 원본 데이터를 백업하고 이를 데이터베이스에 적용하는 방식이고, WAL 방식은, 원본 데이터를 데이터베이스 파일에 유지하고, 변경된 페이지를 WAL 파일에 저장하는 방식이다.

[0006] 하지만, RBJ 방식과 WAL 방식은, 변경되지 않는 데이터까지도 백업해야 하는 문제가 있었다.

[0007] 일 예로, RBJ 방식과 WAL 방식은, 페이지 내에서, 변경된 데이터는 일부분에 불과하지만, 변경된 데이터를 포함하는 페이지 전체를 저장해야 한다.

[0008] 또한, WAL 방식은, 읽기 연산을 수행할 때마다, 로그 파일을 탐색해야 하는 문제가 있었다.

[0009] 일 예로, WAL 방식은, 최신 수정 사항이 WAL 파일에 존재하므로, 읽기 연산이 수행되면, WAL 파일을 무조건 한 번 검색해야 한다.

[0010] 또한, WAL 방식은, 체크 포인트 연산이 지연되는 문제가 있었다.

[0011] 일 예로, WAL 방식은, 데이터에 읽기 잠금이 걸려 있는 경우, 체크 포인트의 연산을 수행하지 못하므로, WAL 파일이 데이터베이스 파일보다 수십 배 또는 수백 배 이상 커지는 경우가 발생할 수 있다.

[0012] 이와 같이, 기존의 데이터 복구 방식은, 데이터를 복구하는 시간이 너무 느려 많은 시간이 소요되고, 데이터 복구가 불안정한 문제들이 있었다.

발명의 내용

해결하려는 과제

[0013] 본 발명은 전술한 문제 및 다른 문제를 해결하는 것을 목적으로 한다. 또 다른 목적은, 로그 파일의 크기를 줄이고 전체 입출력 부하를 낮출 수 있는 데이터베이스 관리 시스템 및 그의 데이터 변경 및 복구 방법을 제공하는 것을 목적으로 한다.

[0014] 또 다른 목적은, 체크포인트 속도를 개선할 수 있는 데이터베이스 관리 시스템 및 그의 데이터 변경 및 복구 방법을 제공하는 것을 목적으로 한다.

[0015] 또 다른 목적은, 검색 비용을 낮출 수 있는 데이터베이스 관리 시스템 및 그의 데이터 변경 및 복구 방법을 제

공하는 것을 목적으로 한다.

[0016] 본 발명에서 이루고자 하는 기술적 과제들은 이상에서 언급한 기술적 과제들로 제한되지 않으며, 언급하지 않은 또 다른 기술적 과제들은 아래의 기재로부터 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 명확하게 이해될 수 있을 것이다.

과제의 해결 수단

[0017] 본 발명의 일 실시예에 의한 데이터베이스 관리 시스템은, 페이지 단위로 구성된 데이터베이스 파일을 저장하는 데이터베이스와, 트랜잭션(transaction)이 수행된 데이터베이스 파일에 대한 로그 파일을 생성하는 로그 파일 생성부와, 데이터베이스 파일의 트랜잭션을 수행하는 트랜잭션 수행부와, 데이터베이스, 로그 파일 생성부 및 트랜잭션 수행부를 제어하는 제어부를 포함하고, 제어부는, 데이터베이스 파일의 트랜잭션이 요청되면, 데이터베이스 파일의 트랜잭션을 수행하도록, 트랜잭션 수행부를 제어하고, 데이터베이스 파일의 트랜잭션이 수행되면, 데이터베이스 파일의 페이지들 중, 내용이 변경된 더티 페이지(dirty page) 별로, 변경 내역에 대한 로그 레코드(log record)를 생성하고, 데이터베이스에 트랜잭션이 커밋(commit)되면, 생성된 전체 로그 레코드를 로그 파일에 저장하도록, 로그 파일 생성부를 제어할 수 있다.

[0018] 본 발명의 일 실시예에 의한 데이터베이스 관리 시스템의 데이터 변경 방법은, 데이터베이스 파일의 변경 요청을 수신하는 단계와, 변경 요청된 데이터베이스 파일의 트랜잭션을 수행하는 단계와, 트랜잭션이 수행된 데이터베이스 파일의 더티 페이지(dirty page)를 확인하여, 내용이 변경된 더티 페이지 별로, 변경 내역에 대한 로그 레코드(log record)를 생성하는 단계와, 트랜잭션의 커밋(commit)이 요청되면, 생성된 전체 로그 레코드를 로그 파일의 로그 페이지에 저장하는 단계와, 트랜잭션을 데이터베이스에 커밋하는 단계를 포함할 수 있다.

[0019] 본 발명의 일 실시예에 의한 데이터베이스 관리 시스템의 데이터 복구 방법은, 데이터 복구 요청을 수신하는 단계와, 복구하고자 하는 데이터베이스 파일에 대한 로그 파일의 존재를 확인하는 단계와, 로그 파일이 존재하면, 로그 파일의 마지막 로그 레코드로부터 마지막 커밋 로그 레코드까지 역순서로 검색하여, 복구하고자 하는 데이터베이스 파일에 대한 원본 페이지가 존재하는지를 확인하는 단계와, 원본 페이지가 존재하면, 원본 페이지를 상기 데이터베이스 파일에 저장하는 단계와, 로그 파일의 마지막 체크포인트 로그 레코드로부터 마지막 커밋 로그 레코드까지 정순서로 검색하여, 그들 사이에 존재하는 로그 레코드를 재수행하는 단계와, 로그 파일의 마지막 커밋 로그 레코드 이후에 존재하는 로그 레코드를 삭제하는 단계를 포함할 수 있다.

발명의 효과

[0020] 본 발명에 따른 데이터베이스 관리 시스템 및 그의 데이터 변경 및 복구 방법의 효과에 대해 설명하면 다음과 같다.

[0021] 본 발명의 실시예들 중 적어도 하나에 의하면, 변경된 페이지 전체를 로깅하지 않고 페이지마다 데이터베이스 파일을 변경하는 연산만을 로깅함으로써, 로그 파일의 크기를 줄이고 전체 입/출력 부하를 낮출 수 있다.

[0022] 본 발명의 실시예들 중 적어도 하나에 의하면, 체크 포인트 시, 데이터베이스 파일에 대한 싱크(sync) 연산만을 수행하므로, 추가적인 마이그레이션 비용(migration cost)가 필요한 WAL 방식에 비해, 체크포인트 속도를 개선할 수 있으며, 이를 통해 ANR(Application Not Responding)의 문제 해결이 가능하다.

[0023] 본 발명의 실시예들 중 적어도 하나에 의하면, 읽기 시, WAL 방식의 경우, WAL 파일을 무조건 한 번 검색해야 하므로, 선택(select) 연산에 불리하지만, 본 발명의 경우, 데이터베이스 파일만을 검색하므로, 검색 비용을 더 낮출 수 있다.

[0024] 본 발명의 적용 가능성의 추가적인 범위는 이하의 상세한 설명으로부터 명백해질 것이다. 그러나 본 발명의 사상 및 범위 내에서 다양한 변경 및 수정은 당업자에게 명확하게 이해될 수 있으므로, 상세한 설명 및 본 발명의 바람직한 실시 예와 같은 특정 실시 예는 단지 예시로 주어진 것으로 이해되어야 한다.

도면의 간단한 설명

[0025] 도 1은 본 발명 일 실시예에 따른 데이터베이스 관리 시스템을 보여주는 블럭 구성도이다.

도 2는 데이터베이스에 저장되는 컨텐츠를 보여주는 도면이다.

도 3은 데이터베이스 파일의 구조를 보여주는 도면이다.

도 4는 데이터베이스 파일의 변경에 따른 더티 페이지를 보여주는 도면이다.

도 5는 도 4의 더티 페이지에 따른 로그 레코드 생성 방식을 보여주는 도면이다.

도 6은 로그 레코드의 크기에 따른 로깅 모드 변경 과정을 설명하기 위한 도면이다.

도 7은 로그 레코드의 구조를 보여주는 도면이다.

도 8은 도 7의 로그 레코드가 삽입되는 로그 파일의 구조를 보여주는 도면이다.

도 9는 데이터베이스 파일의 변경 명령들을 보여주는 변경 명령 리스트이다.

도 10 및 도 11은 로그 페이지 내에 삽입되는 로그 레코드의 위치를 설명하기 위한 도면이다.

도 12는 로그 파일의 톤 페이지(torn page)를 설명하기 위한 도면이다.

도 13은 더티 페이지의 변경 이력을 체크하기 위한 비트맵 인덱스를 보여주는 도면이다.

도 14는 더티 페이지의 변경 이력에 따른 로그 파일을 보여주는 도면이다.

도 15는 체크섬이 저장된 로그 페이지를 보여주는 도면이다.

도 16은 본 발명 일 실시예에 따른 데이터베이스 관리 시스템의 데이터 복구 방법을 설명하기 위한 흐름도이다.

도 17 내지 도 20은 본 발명 일 실시예에 따른 데이터베이스 관리 시스템의 복구 성능을 보여주는 도면이다.

발명을 실시하기 위한 구체적인 내용

[0026] 이하, 첨부된 도면을 참조하여 본 명세서에 개시된 실시 예를 상세히 설명하되, 도면 부호에 관계없이 동일하거나 유사한 구성요소는 동일한 참조 번호를 부여하고 이에 대한 중복되는 설명은 생략하기로 한다. 이하의 설명에서 사용되는 구성요소에 대한 접미사 "모듈" 및 "부"는 명세서 작성의 용이함만이 고려되어 부여되거나 혼용되는 것으로서, 그 자체로 서로 구별되는 의미 또는 역할을 갖는 것은 아니다. 또한, 본 명세서에 개시된 실시 예를 설명함에 있어서 관련된 공지 기술에 대한 구체적인 설명이 본 명세서에 개시된 실시 예의 요지를 흐릴 수 있다고 판단되는 경우 그 상세한 설명을 생략한다. 또한, 첨부된 도면은 본 명세서에 개시된 실시 예를 쉽게 이해할 수 있도록 하기 위한 것일 뿐, 첨부된 도면에 의해 본 명세서에 개시된 기술적 사상이 제한되지 않으며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다.

[0027] 제1, 제2 등과 같이 서수를 포함하는 용어는 다양한 구성요소들을 설명하는데 사용될 수 있지만, 상기 구성요소들은 상기 용어들에 의해 한정되지는 않는다. 상기 용어들은 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 사용된다.

[0028] 어떤 구성요소가 다른 구성요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는, 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다.

[0029] 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다.

[0030] 본 출원에서, "포함한다" 또는 "가지다" 등의 용어는 명세서상에 기재된 특징, 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0031] 도 1은 본 발명 일 실시예에 따른 데이터베이스 관리 시스템을 보여주는 블럭 구성도이다.

[0032] 도 1에 도시된 바와 같이, 본 발명의 데이터베이스 관리 시스템은, 데이터베이스(100), 로그 파일(log file) 생성부(200), 트랜잭션(transaction) 수행부(300) 및 제어부(400)를 포함할 수 있다.

[0033] 여기서, 데이터베이스(100)는, 페이지 단위로 구성된 데이터베이스 파일(DB file)을 저장할 수 있다.

[0034] 데이터베이스(100)는, 데이터가 파일로 저장되는 데이터베이스 파일을 포함할 수 있다.

[0035] 데이터베이스 파일은, 페이지 단위로 구성되는데, 각 페이지에는, 실제 데이터베이스에 저장되는 정보인 레코드

(record)가 저장될 수 있다.

[0036] 따라서, 데이터베이스 파일의 하나의 페이지에는, 다수의 레코드들이 저장될 수 있다.

[0037] 그리고, 로그 파일 생성부(200)는, 트랜잭션(transaction)이 수행된 데이터베이스 파일에 대한 로그 파일을 생성할 수 있다.

[0038] 여기서, 로그 파일 생성부(200)는, 트랜잭션의 수행에 의해, 데이터베이스 파일이 변경되면, 변경된 내용에 대한 정보들이 저장된 로그 파일을 생성할 수 있다.

[0039] 따라서, 본 발명의 시스템은, 데이터베이스 파일의 복구시, 로그 파일을 이용하여, 손상된 데이터를 복구할 수 있다.

[0040] 로그 파일 생성부(200)는, 트랜잭션의 수행에 따라, 데이터베이스 파일의 내용이 변경될 때 마다, 변경된 내용을 포함하는 로그 레코드를 생성하고, 생성된 로그 레코드를 버퍼 메모리에 축적한 다음, 트랜잭션 수행이 완료되고, 커밋(commit) 명령을 수행할 때, 버퍼 메모리에 축적된 로그 레코드들을 로그 파일의 로그 페이지에 삽입할 수 있다.

[0041] 따라서, 본 발명의 시스템은, 로그 레코드들이 삽입된 로그 파일을 데이터베이스(100)에 저장하고, 트랜잭션 수행에 의해, 내용이 변경된 데이터베이스 파일을 데이터베이스(100)에 저장할 수 있다.

[0042] 다음, 트랜잭션 수행부(300)는, 데이터베이스 파일의 트랜잭션을 수행할 수 있다.

[0043] 여기서, 트랜잭션 수행부(300)는, 변경하고자 하는 데이터베이스 파일의 페이지를 선택하고, 선택된 데이터베이스 파일의 페이지에 대한 트랜잭션을 수행하여, 데이터베이스 파일의 내용을 변경할 수 있다.

[0044] 이어, 제어부(400)는, 데이터베이스(100), 로그 파일 생성부(200) 및 트랜잭션 수행부(300)를 제어할 수 있다.

[0045] 여기서, 제어부(400)는, 데이터베이스 파일의 트랜잭션이 요청되면, 데이터베이스 파일의 트랜잭션을 수행하도록, 트랜잭션 수행부(300)를 제어할 수 있다.

[0046] 그리고, 제어부(400)는, 데이터베이스 파일의 트랜잭션이 수행되면, 데이터베이스 파일의 페이지들 중, 더티 페이지(dirty page)를 확인하여, 내용이 변경된 더티 페이지 별로, 변경 내역에 대한 로그 레코드(log record)를 생성하고, 데이터베이스(100)에 트랜잭션이 커밋(commit)되면, 생성된 전체 로그 레코드를 로그 파일에 저장하도록, 로그 파일 생성부(200)를 제어할 수 있다.

[0047] 여기서, 데이터베이스 파일의 더티 페이지는, 트랜잭션 수행에 의해, 내용이 변경된 페이지를 의미할 수 있다.

[0048] 즉, 제어부(400)는, 트랜잭션이 수행되어, 데이터베이스 파일의 페이지가 변경될 때마다, 변경 내역에 대한 로그 레코드를 생성하여 버퍼 메모리에 생성된 로그 레코드를 축적한 다음, 커밋 명령 수행시에, 축적된 전체 로그 레코드를 한 번에 로그 파일에 저장할 수 있다.

[0049] 따라서, 본 발명은, 데이터베이스 파일의 페이지가 변경될 때마다, 변경 페이지 전체를 저장하지 않고, 변경 내역에 대한 로그 레코드만을 생성하여 로그 파일에 저장하므로, 데이터베이스 파일의 변경 및 복구가 신속하고 빠르며, 로그 파일의 크기를 줄일 수 있다.

[0050] 또한, 본 발명은, 로그 레코드를 압축하여 로그 파일에 저장할 수 있으므로, 로그 파일의 크기를 더욱 줄일 수도 있다.

[0051] 다음, 제어부(400)는, 생성된 전체 로그 레코드를 로그 파일에 저장할 때, 생성된 전체 로그 레코드의 크기와 더티 페이지의 크기를 비교할 수 있다.

[0052] 일 예로, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 작으면, 생성된 전체 로그 레코드를 로그 파일에 저장하고, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 크면, 더티 페이지를 로그 파일에 저장하도록, 로그 파일 생성부(200)를 제어할 수 있다.

[0053] 예를 들면, 하나의 더티 페이지에 대해 많은 변경이 있는 경우, 더티 페이지에 대한 로그 레코드의 개수도 많아져서, 축적된 전체 로그 레코드의 크기가 더티 페이지보다 더 클 수가 있다.

[0054] 이 경우, 더티 페이지보다 큰 크기의 로그 레코드를 로그 파일에 저장하는 제 1 로깅 방식은, 더티 페이지를 로그 파일에 저장하는 제 2 로깅 방식보다 속도면이나 효율면에서 저하될 수 있다.

[0055] 따라서, 본 발명은, 생성된 전체 로그 레코드의 크기와 더티 페이지의 크기를 비교하여, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 작으면, 제 1 로깅 방식으로 생성된 전체 로그 레코드를 로그 파일에 저장하고, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 크면, 제 2 로깅 방식으로, 더티 페이지를 로그 파일에 저장할 수 있다.

[0056] 그러므로, 본 발명은, 생성된 전체 로그 레코드의 크기와 더티 페이지의 크기를 비교 결과에 따라, 제 1 로깅 방식 또는 제 2 로깅 방식을 선택함으로써, 로깅 속도 및 효율을 향상시킬 수 있다.

[0057] 또한, 제어부(400)는, 로그 페이지 내에 삽입되는 로그 레코드의 개수가, 로그 페이지의 빈 공간 크기를 초과하면, 새로운 추가 로그 페이지를 생성하고, 로그 페이지 헤더 내에 오버플로우 플래그를 추가하도록, 로그 파일 생성부(200)를 제어할 수 있다.

[0058] 따라서, 본 발명은, 로그 레코드의 개수가 로그 페이지를 초과하여도, 로그 레코드의 손실 없이, 생성된 모든 로그 레코드를 로그 파일에 모두 반영할 수 있다.

[0059] 그리고, 제어부(400)는, 데이터베이스 파일의 더티 페이지를 확인할 때, 더티 페이지에 대한 변경 이력을 확인하고, 더티 페이지에 변경 이력이 존재하면, 더티 페이지의 변경 내역에 대한 로그 레코드를 생성하여, 생성된 전체 로그 레코드를 로그 파일에 저장하고, 더티 페이지에 변경 이력이 존재하지 않으면, 더티 페이지의 변경 내역에 대한 로그 레코드 생성 없이, 더티 페이지 전체를 로그 파일에 저장하도록, 로그 파일 생성부(200)를 제어할 수 있다.

[0060] 여기서, 제어부(400)는, 더티 페이지에 대한 변경 이력을 확인할 때, 비트맵 인덱스(bitmap index)를 이용하여 변경 이력을 확인할 수 있다.

[0061] 예를 들면, 로그 파일이 데이터베이스에 저장될 때, 로그 파일 내의 페이지 중, 일부 로그 레코드는, 손실되거나 손상될 수 있다.

[0062] 따라서, 본 발명은, 데이터베이스 파일의 더티 페이지에 대한 변경 이력을 확인하고, 변경 이력이 없으면, 현재 변경 내역에 대한 로그 레코드 생성 없이, 트랜잭션이 수행된 더티 페이지 전체를 로그 파일에 저장하고, 변경 이력이 있으면, 현재 변경 내역에 대한 로그 레코드를 생성하여 로그 파일에 저장할 수 있다.

[0063] 즉, 본 발명은, 변경 이력이 없으면, 로그 레코드 생성 없이, 트랜젝션에 의해 변경된 더티 페이지의 전체 내용을, 로그 파일에 저장함으로써, 추후 로그 레코드의 손실이 있어도 데이터베이스 파일의 데이터 복구가 가능하다.

[0064] 또한, 본 발명은, 변경 이력이 있으면, 이전에 이미 더티 페이지의 전체 내용이 로그 파일에 저장되어 있으므로, 더티 페이지의 전체 내용을 저장하는 과정을 생략할 수 있다.

[0065] 따라서, 본 발명은, 일부 로그 레코드가 손실된 톤 페이지(torn page)를 포함하는 로그 파일이 존재하여도, 최초 트랜잭션이 수행된 원본 페이지가 로그 파일에 존재하므로, 데이터베이스 파일의 데이터 복구가 가능하다.

[0066] 다음, 제어부(400)는, 트랜잭션이 커밋될 때, 로그 파일의 커밋 로그 레코드(commit log record)에 체크섬(checksum)을 저장할 수 있다.

[0067] 여기서, 제어부(400)는, 특정 트랜잭션에 의해 생성된 다수의 로그 레코드들이 제 1 로그 페이지로부터 제 2 로그 페이지로 이어질 때, 제 1 로그 페이지의 마지막 지점에 제 1 체크섬을 저장하고, 특정 트랜잭션의 커밋 로그 레코드에 제 2 체크섬을 저장할 수 있다.

[0068] 그리고, 체크섬은, 로그 레코드와 로그 레코드를 지시하는 로그 포인터를 대상으로 수행될 수 있다.

[0069] 이어, 제어부(400)는, 데이터베이스 파일의 복구 시에, 체크섬과 로그 레코드의 내용을 비교하고, 체크섬과 로그 레코드의 내용이 다르면, 해당하는 트랜잭션을 실패로 판정하고, 재수행(redo) 과정에서 로그 레코드를 제외시킬 수 있다.

[0070] 이와 같이, 본 발명은, 트랜잭션 커밋 시에, 커밋 로그 레코드에 체크섬을 저장하여, 데이터 복구 과정에서, 로그 레코드의 손상을 확인하여 복구 에러를 방지할 수 있다.

[0071] 또한, 제어부(400)는, 생성된 전체 로그 레코드를 로그 파일에 저장할 때, 전체 로그 레코드를 한 번에 압축하고, 압축된 전체 로그 레코드를 로그 파일에 저장할 수 있다.

[0072] 다른 경우로서, 제어부(400)는, 생성된 전체 로그 레코드를 로그 파일에 저장할 때, 각각의 로그 레코드를 개별적으로 압축하고, 개별적으로 압축된 전체 로그 레코드를 로그 파일에 저장할 수도 있다.

[0073] 이와 같이, 본 발명은, 로그 레코드를 압축함으로써, 로그 파일의 크기를 줄일 수 있어, 변경 및 복구 속도가 빠르고 효율성이 우수하다.

[0074] 도 2는 데이터베이스에 저장되는 컨텐츠를 보여주는 도면이고, 도 3은 데이터베이스 파일의 구조를 보여주는 도면이다.

[0075] 도 2 및 도 3에 도시된 바와 같이, 데이터베이스는, 데이터베이스 파일(DB file)(120)을 저장하고, 데이터베이스 파일(120)은, 페이지 단위로 구성될 수 있다.

[0076] 그리고, 데이터베이스 파일(120)의 각 페이지는, 실제 데이터베이스에 저장되는 정보인 레코드(record)가 저장될 수 있다.

[0077] 따라서, 데이터베이스 파일(120)의 하나의 페이지에는, 다수의 레코드(122)들이 저장될 수 있다.

[0078] 일 예로, 도 2와 같이, 고객 정보 리스트(110)라는 정보가 데이터베이스에 저장되어 있는 경우, 고객 정보 리스트(110)에는, 고객의 이름, 나이, 성별 등과 같은 고객 정보(112)를 포함할 수 있다.

[0079] 여기서, 고객 정보(112)는, 도 3과 같이, 데이터베이스 파일(120)의 각 페이지 내에 순차적으로 저장될 수 있다.

[0080] 각 고객 정보(112)는, 해당하는 페이지 내에 레코드(122) 형태로 저장될 수 있다.

[0081] 도 3은, 하나의 페이지 내에 3개의 레코드(122)가 저장된 데이터베이스 파일(120)의 구조를 보여주고 있다.

[0082] 도 4는 데이터베이스 파일의 변경에 따른 더티 페이지를 보여주는 도면이고, 도 5는 도 4의 더티 페이지에 따른 로그 레코드 생성 방식을 보여주는 도면이다.

[0083] 도 4에 도시된 바와 같이, 본 발명의 시스템은, 데이터베이스 파일의 페이지 1 내에, 테이블 t1을 추가하는 명령이 수신되면, 수신된 명령에 따라, 데이터베이스 파일의 페이지 1에 테이블 t1을 삽입하여, 데이터베이스 파일의 변경 과정을 수행할 수 있다.

[0084] 여기서, 데이터베이스 파일의 페이지 1 내에는, 테이블 t1이 삽입되고, 데이터베이스 파일의 페이지 1에 있던 기존의 데이터는, 데이터베이스 파일의 페이지 2로 이동될 수 있다.

[0085] 따라서, 데이터베이스 파일은, 변경 명령을 수행한 결과, 내용이 변경된 2개의 더티 페이지(dirty page)를 생성할 수 있다.

[0086] 다음, 도 5에 도시된 바와 같이, 내용이 변경된 2개의 더티 페이지에 대한 로그 레코드를 생성할 수 있다.

[0087] 일 예로, WAL 로깅 방식은, 변경된 2개의 더티 페이지에 대한 전체 내용을 로그 파일에 모두 저장하는 방식이므로, 로그 파일의 크기가 매우 클 수 있다.

[0088] 그리고, 논리적 로깅 방식은, 수행된 작업(modification operation)의 종류와 로그 레코드 자체만을 로그 파일에 저장하는 방식이므로, 데이터 복구 수행시, 질의 전체를 다시 수행해야 하는 문제들이 있었다.

[0089] 일 예로, 논리적 로깅 방식은, 데이터베이스 파일의 페이지 1에 테이블 t1이 삽입되었다는 내용만이 로그 파일에 저장될 뿐이다.

[0090] 다음, 본 발명의 시스템에 따른 로깅 방식은, 페이지 지향 논리적 로깅 방식(page-oriented logical logging method)으로서, 각 페이지 별로 변경 내역을 각각의 변경 작업의 종류와 함께 저장하는 방식으로서, 로그 파일의 크기도 작고, 데이터 복구 수행시, 각 페이지 별로 독립적인 복구가 가능하며 복구 속도가 빠르다.

[0091] 일 예로, 본 발명의 페이지 지향 논리적 로깅 방식은, 데이터베이스 파일의 페이지 1 내에, 테이블 t1이 삽입되고, 데이터베이스 파일의 페이지 2 내에, 페이지 1의 기존 데이터가 삽입되었다는 내용이 페이지 별로 각각 로그 파일에 저장될 수 있다.

[0092] 도 6은 로그 레코드의 크기에 따른 로깅 모드 변경 과정을 설명하기 위한 도면이다.

[0093] 본 발명의 시스템은, 내용이 변경된 더티 페이지 별로 각 더티 페이지의 변경 내역에 대한 로그 레코드를 생성할 수 있다.

[0094] 여기서, 본 발명의 시스템은, 생성된 전체 로그 레코드를 로그 파일에 저장할 때, 생성된 전체 로그 레코드의 크기와 더티 페이지의 크기를 비교할 수 있다.

[0095] 그리고, 시스템은, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 작으면, 생성된 전체 로그 레코드를 로그 파일에 저장하고, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 크면, 더티 페이지를 로그 파일에 저장할 수 있다.

[0096] 도 6에 도시된 바와 같이, 데이터베이스 파일의 페이지들 중, 내용이 변경된 더티 페이지로서, 더티 페이지 13과 더티 페이지 17이 있다면, 더티 페이지 13과 17의 변경 내역에 대한 로그 레코드들이 생성될 수 있다.

[0097] 여기서, 로그 레코드들의 전체 크기가, 로그 페이지 5개의 크기일 경우, 더티 페이지들의 크기보다 더 클 수 있다.

[0098] 이 경우, 로그 레코드를 로그 파일에 저장하는 제 1 로깅 방식보다 로그 페이지를 로그 파일에 저장하는 제 2 로깅 방식으로 로깅 모드를 변경함으로써, 로깅 속도와 효율을 향상시킬 수 있다.

[0099] 일 예로, 하나의 더티 페이지에 대해 많은 변경이 있는 경우, 더티 페이지에 대한 로그 레코드의 개수도 많아져서, 축적된 전체 로그 레코드의 크기가 더티 페이지보다 더 클 수가 있다.

[0100] 이 경우, 더티 페이지보다 큰 크기의 로그 레코드를 로그 파일에 저장하는 제 1 로깅 방식은, 더티 페이지를 로그 파일에 저장하는 제 2 로깅 방식보다 속도면이나 효율면에서 저하될 수 있다.

[0101] 따라서, 본 발명의 시스템은, 생성된 전체 로그 레코드의 크기와 더티 페이지의 크기를 비교하여, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 작으면, 제 1 로깅 방식으로 생성된 전체 로그 레코드를 로그 파일에 저장하고, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 크면, 제 2 로깅 방식으로, 더티 페이지를 로그 파일에 저장할 수 있다.

[0102] 그러므로, 본 발명은, 생성된 전체 로그 레코드의 크기와 더티 페이지의 크기를 비교 결과에 따라, 제 1 로깅 방식 또는 제 2 로깅 방식을 선택함으로써, 로깅 속도 및 효율을 향상시킬 수 있다.

[0103] 도 7은 로그 레코드의 구조를 보여주는 도면이고, 도 8은 도 7의 로그 레코드가 삽입되는 로그 파일의 구조를 보여주는 도면이며, 도 9는 데이터베이스 파일의 변경 명령들을 보여주는 변경 명령 리스트이다.

[0104] 도 7 및 도 8에 도시된 바와 같이, 로그 파일(500)은, 로그 페이지 단위로 구성되는데, 로그 파일 헤더(log file header)(510)와 다수의 로그 페이지(520)들을 포함할 수 있다.

[0105] 그리고, 각 로그 페이지(520)는, 로그 페이지 정보를 포함하는 로그 페이지 헤더(log page header)(610), 각 로그 레코드의 위치 및 길이 정보를 포함하는 로그 포인터(log pointer)(620), 로그 데이터(log data)를 포함하는 로그 레코드(log record)(630)로 구성될 수 있다.

[0106] 여기서, 로그 페이지 헤더(610)는, 로그 페이지 번호, 로그 페이지 내의 로그 레코드 개수, 사용 가능한 공간, 그리고 플래그 중, 적어도 어느 하나의 정보를 포함할 수 있다.

[0107] 이어, 로그 포인터(620)는, 로그 레코드(630)의 위치 정보를 저장하는 제 1 유닛(622)과, 로그 레코드(630)의 길이 정보를 저장하는 제 2 유닛(624)을 포함할 수 있다.

[0108] 그리고, 로그 레코드(630)는, 데이터베이스 파일을 변경하는 명령을 리두(redo)하는데, 필요한 데이터를 포함한다.

[0109] 여기서, 로그 레코드(630)가 압축될 경우, 본 발명의 시스템은 처리 속도 및 처리 효율면에서, 더욱 큰 성능 향상을 이룰 수 있다.

[0110] 일 예로, 로그 레코드(630)는, 작업 명령 정보를 저장하는 제 1 유닛(632), 작업이 수행된 데이터베이스 파일(120)의 더티 페이지 번호를 저장하는 제 2 유닛(634), 더티 페이지 내에서, 작업이 수행된 셀 번호를 저장하는 제 3 유닛(636), 작업 명령에 따른 로그 데이터를 저장하는 제 4 유닛(638)을 포함할 수 있다.

[0111] 다음, 로그 포인터(620)와 로그 레코드(630)는, 서로 일대일 대응하여 배치될 수 있다.

[0112] 여기서, 로그 포인터(620)의 개수와 로그 레코드(630)의 개수는, 서로 동일할 수 있다.

[0113] 로그 포인터(620)는, 로그 레코드(630)가 로그 페이지(520) 내에 삽입될 때, 생성될 수 있다.

[0114] 경우에 따라, 로그 포인터(620)는, 로그 레코드(630)가 로그 페이지(520) 내에 삽입되기 전인 로그 레코드(630)가 생성될 때, 생성될 수도 있다.

[0115] 이처럼, 로그 포인터(620)는, 로그 레코드(630)가 생성되거나 또는 로그 페이지에 로그 레코드(630)가 삽입될 때, 생성되므로, 각 로그 레코드(630) 마다, 그에 상응하는 로그 포인터(620)가 존재할 수 있다.

[0116] 본 발명의 시스템은, 각 페이지 별로 논리적 로깅을 하기 때문에 논리적 로깅과 마찬가지로 논리적 작업(logical operation)에 대한 정의가 필요하다.

[0117] 따라서, 본 발명은, 2가지 조건을 만족하는 논리적 작업을 도 9와 같이, 수행할 수 있다.

[0118] 첫째, 하나의 논리적 작업은, 데이터베이스 파일을 변경하는 하나의 작업에 대응되어야 한다.

[0119] 둘째, 데이터베이스 파일을 변경하는 작업은, 로그 레코드를 생성할 수 있는 논리적 작업으로 표현될 수 있어야 한다.

[0120] 본 발명의 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.

[0121] 이 중 Page_ stressed는, 다른 작업에서 나타나지 않는 없는 독특한 특징을 갖는데, 이는 본 발명의 버퍼 관리 정책(Buffer Management Policy)과 깊은 관련이 있다.

[0122] 본 발명의 버퍼 관리자는, 기본적으로 특정 트랜잭션에 의해 수정된 데이터베이스 파일의 더티 페이지를 해당 트랜잭션이 커밋되기 전까지 데이터베이스 파일에 반영하지 않는다.

[0123] 그러나, 예외적으로 수정된 페이지의 수가 증가하여 이들이 차지하는 메모리가 본 발명이 정해놓은 한계를 넘어가게 되면, 본 발명의 PagerStress 함수에 의해 수정된 페이지가 트랜잭션이 커밋되기 전에 데이터베이스 파일에 쓰여질 수 있다.

[0124] 따라서, 트랜잭션 롤백 및 데이터 복구 시에, 이에 대한 처리를 하기 위해 별도의 로그를 저장할 수 있다.

[0125] 또한, 본 발명은, 데이터베이스 파일을 변경하는 논리적 작업이 시스템에 의해 수행되면, 로그 레코드를 생성하고, 이를 로그 파일의 로그 페이지에 삽입한다.

[0126] 로그 레코드(630)는, 하나의 논리적 작업을 리두(Re-do)하는데 필요한 데이터로서, 도 7과 같이 구성될 수 있다.

[0127] 도 7과 같이, 로그 레코드(630)의 Opcode는, 도 9에서 정의한 작업 중 어떤 작업이 수행되었는지를 나타낸다.

[0128] 그리고, Page Number는, 이 작업으로 인해 변경된 데이터베이스의 페이지 번호를, Cell index는 페이지 내의 몇 번째 셀에 관한 작업인지를 의미한다.

[0129] 다음, Cell index 뒤의 Data는, Opcode의 특성에 따라 필요한 데이터를 저장한다.

[0130] 대부분의 경우, 도 9에 명시된 함수를 호출하는데 필요한 파라미터를 저장하지만, 앞서 언급한 Page_stressed 및 Whole_page에 대해서는, 데이터베이스 페이지를 통째로 저장한다.

[0131] 본 발명은, 커밋되지 않은 트랜잭션의 작업을 데이터베이스 파일에 반영하지 않기 때문에, 논리적 로그 기반 복구를 할 때, 로그를 언두(undo)할 필요가 없다.

[0132] 단, Pager Stress에 한해서, 변경된 페이지가 커밋 전에 데이터베이스 파일에 반영될 수 있고, 이후 해당 트랜잭션이 롤백되면, 미리 쓰여진 페이지 또한 트랜잭션 시작 전으로 돌아가야 한다.

[0133] 해당 페이지에 연관된 모든 작업에 대해 언두(undo)를 수행하는 것은 비효율적이므로, 이것을 방지하기 위해서 봉 발명은, PagerStress 함수 호출 시, 수정되기 전 원본 페이지를 로그 레코드(630)의 Data 필드에 저장한다.

[0134] 로그 파일(500)은, 데이터베이스 파일과 마찬가지로 여러 개의 페이지가 모여서 이루어진다.

[0135] 각각의 로그 페이지(520)는, slotted-page 구조로 구현될 수 있다.

[0136] 도 8과 같이, 로그 페이지(520)의 가장 앞부분은, 로그 페이지 헤더가 존재한다.

[0137] 로그 페이지 헤더에는, 페이지 번호, 페이지 안의 로그 엔트리 개수, 사용 가능한 공간(lower, upper), 기타 플

래그 등이 저장된다.

[0138] 페이지 헤더 뒤의 포인터들은, 페이지 끝에서부터 저장되어 있는 로그 엔트리의 시작 주소와 크기(offset, length)를 저장하고 있다.

[0139] 따라서, Pointer n이 가리키고 있는 주소에서 해당 byte만큼을 가져옴으로써, n번째 로그 엔트리에 접근할 수 있다.

[0140] 마찬가지로 로그 레코드(630)가 로그 페이지(520) 내에 삽입될 때는, 프포퐁포인터(pointer)의 끝에 새로운 포인터가 만들어지고, 마지막 로그 엔트리가 저장된 주소 앞에, 삽입될 로그 엔트리가 저장된다.

[0141] 또한, 본 발명은, 로그 페이지의 내부 단편화를 최소화하기 위해, 오버플로우 레코드를 지원한다.

[0142] 삽입되는 로그 레코드가 로그 페이지의 빈 공간보다 클 경우, 2개 이상의 로그 페이지에 걸쳐서 저장될 수 있으며, 페이지 헤더의 오버플로우 플래그를 통해 이를 관리한다.

[0143] 도 8은, '주소록' 테이블에 레코드를 삽입하여, 데이터베이스 파일을 변경할 경우, 로그 파일의 로그 레코드 삽입 과정을 보여줄 수 있다.

[0144] 먼저, 본 발명의 시스템은, 트랜잭션에 의해 데이터베이스 파일이 변경되고, 데이터베이스 파일의 변경 내역에 대한 로그 레코드를 생성하며, 생성된 로그 레코드를 로그 파일의 로그 페이지 내에 삽입할 수 있다.

[0145] 그리고, 로그 페이지를 포함하는 로그 파일은, 버퍼 메모리에 상주하다가, 트랜잭션에 대한 커밋 명령이 수행되면, 데이터베이스에 해당하는 로그 파일이 저장될 수 있다.

[0146] 도 10 및 도 11은 로그 페이지 내에 삽입되는 로그 레코드의 위치를 설명하기 위한 도면이다.

[0147] 도 10 및 도 11과 같이, 본 발명의 로그 파일은, 다양한 구조의 로그 페이지를 사용할 수 있다.

[0148] 도 10의 로그 페이지는, 스롯 페이지 구조(slotted-page structure)로서, 각 로그 레코드의 위치와 길이를 저장하는 포인터의 어레이(array)를 로그 페이지의 한쪽 끝에서부터 저장하고, 로그 레코드를 로그 페이지의 다른 끝에서부터 저장하는 방식이다.

[0149] 또한, 도 11의 로그 페이지는, 어펜드 온리 로그 페이지 구조(append-only log page structure)로서, 로그 레코드의 길이와 이전 로그 레코드의 위치를 가지는 로그 헤더를 저장하고, 로그 레코드를 순차적으로 저장하는 방식이다.

[0150] 도 10에 도시된 바와 같이, 스롯 페이지 구조의 로그 페이지(700)는, 로그 페이지 헤더(710)가, 로그 페이지(700)의 시작 지점에 저장될 수 있다.

[0151] 그리고, 로그 포인터(720)는, 로그 레코드(730)가 로그 페이지(700)에 삽입되는 순서에 따라, 로그 페이지 헤더(720)로부터 로그 페이지(700)의 끝 지점 방향으로 순차 저장될 수 있다.

[0152] 여기서, 로그 포인터(720)는, 상응하는 현재 로그 레코드의 길이 정보와 위치 정보를 포함할 수 있다.

[0153] 일 예로, 로그 포인터(720)는, 플래그(722), 길이(724), 오프셋(726) 정보를 포함할 수 있다.

[0154] 다음, 로그 레코드(730)는, 로그 레코드(730)가 로그 페이지(700)에 삽입되는 순서에 따라, 로그 페이지(700)의 끝 지점으로부터 시작 지점 방향으로 순차 저장될 수 있다.

[0155] 도 11에 도시된 바와 같이, 어펜드 온리 로그 페이지 구조의 로그 페이지(800)는, 로그 페이지 헤더(810)가, 로그 페이지(800)의 시작 지점에 저장될 수 있다.

[0156] 그리고, 로그 포인터(820)와 로그 레코드(830)는, 로그 레코드(830)가 로그 페이지(800)에 삽입되는 순서에 따라, 로그 페이지 헤더(810)로부터 로그 페이지(800)의 끝 지점 방향으로 순차 저장될 수 있다.

[0157] 여기서, 로그 포인터(820)는, 상응하는 현재 로그 레코드의 길이 정보와, 이전 로그 레코드의 위치 정보를 포함할 수 있다.

[0158] 일 예로, 로그 포인터(820)는, 플래그(822), 길이(824), 이전 오프셋(826) 정보를 포함할 수 있다.

[0159] 한편, 본 발명의 시스템은, 로그 페이지 내에 삽입되는 로그 레코드의 개수가, 로그 페이지의 빈 공간 크기를 초과하면, 새로운 추가 로그 페이지를 생성하고, 로그 페이지 헤더 내에 오버플로우 플래그를 추가할 수 있다.

[0160] 그리고, 본 발명의 시스템은, 생성된 전체 로그 레코드를 로그 파일에 저장할 때, 전체 로그 레코드를 한 번에 압축하고, 압축된 전체 로그 레코드를 로그 파일에 저장할 수 있다.

[0161] 또한, 본 발명의 시스템은, 생성된 전체 로그 레코드를 로그 파일에 저장할 때, 각각의 로그 레코드를 개별적으로 압축하고, 개별적으로 압축된 전체 로그 레코드를 로그 파일에 저장할 수도 있다.

[0162] 이와 같이 저장하는 이유는, 데이터 복구 시, 복구 속도 및 복구 효율이 크게 향상될 수 있기 때문이다.

[0163] 도 12는 로그 파일의 톤 페이지(torn page)를 설명하기 위한 도면이고, 도 13은 더티 페이지의 변경 이력을 체크하기 위한 비트맵 인덱스를 보여주는 도면이며, 도 14는 더티 페이지의 변경 이력에 따른 로그 파일을 보여주는 도면이다.

[0164] 도 12에 도시된 바와 같이, 본 발명의 시스템은, 데이터베이스 파일을 변경하는 명령을 수행할 때마다, 로그 레코드를 생성할 수 있다.

[0165] 그리고, 로그 레코드는, 버퍼 메모리 내에 있는 로그 파일의 로그 페이지 내에 순차적으로 저장될 수 있다.

[0166] 일 예로, 버퍼 메모리 내에 있는 로그 파일의 제 1 로그 페이지(1010) 내에 제 1, 제 2, 제 3 로그 레코드(1021, 1022, 1023)이 순차적으로 저장되고, 커밋 명령이 수행되면, 로그 파일의 제 1 로그 페이지(1010)는, 데이터베이스 내에 저장될 수 있다.

[0167] 여기서, 제 1 로그 페이지(1010)는, 데이터베이스에 저장될 때, 수행 실패로 인하여, 제 2, 제 3 로그 레코드(1022, 1023)가 손실되는 문제가 발생할 수 있다.

[0168] 따라서, 제 1 로그 페이지(1010)는, 제 2, 제 3 로그 레코드(1022, 1023)가 손실된 톤 페이지(1020)가 되어, 데이터 복구시, 톤 페이지(1020)인 제 1 로그 페이지(1010)를 사용할 수 없다.

[0169] 이러한 톤 페이지 문제를 해결하기 위하여, 본 발명의 시스템은, 데이터베이스 파일의 변경 명령을 수행할 때, 데이터베이스 파일의 더티 페이지에 대한 변경 이력을 확인하고, 더티 페이지에 변경 이력이 존재하면, 더티 페이지의 변경 내역에 대한 로그 레코드를 생성하여, 생성된 전체 로그 레코드를 로그 파일에 저장할 수 있다.

[0170] 그리고, 본 발명의 시스템은, 더티 페이지에 변경 이력이 존재하지 않으면, 더티 페이지의 변경 내역에 대한 로그 레코드 생성 없이, 더티 페이지 전체를 로그 파일에 저장할 수 있다.

[0171] 여기서, 본 발명의 시스템은, 데이터베이스 파일의 더티 페이지에 대한 변경 이력을 확인할 때, 도 13과 같이, 비트맵 인덱스(bitmap index)(1030)를 이용하여 변경 이력을 확인할 수 있다.

[0172] 여기서, 비트맵 인덱스(1030)는, 데이터베이스 파일의 각 페이지당 1비트만이 소모되므로, 4KB만으로 32768 페이지를 커버할 수 있다.

[0173] 만일, 데이터베이스 파일의 페이지가 32768개 이상이 될 경우, 본 발명의 시스템은, 비트맵을 확장시킬 수 있다.

[0174] 또한, 비트맵은, 쉐어드 메모리 파일로도 구현되어, 여러 프로세스에서 한 데이터베이스 파일에 접근하는 경우도 지원이 가능하다.

[0175] 이처럼, 본 발명의 시스템은, 더티 페이지에 변경 이력이 존재하면, 더티 페이지의 변경 내역에 대한 로그 레코드를 생성하여, 생성된 전체 로그 레코드를 로그 파일에 저장하고, 더티 페이지에 변경 이력이 존재하지 않으면, 더티 페이지의 변경 내역에 대한 로그 레코드 생성 없이, 더티 페이지 전체를 로그 파일에 저장할 수 있다.

[0176] 따라서, 도 14와 같이, 로그 파일(1040)은, 각 페이지마다 데이터베이스 파일의 원본 페이지가 항상 저장되므로, 데이터베이스 내에 톤 페이지가 발생되어도, 로그 파일(1040) 내에 원본 페이지가 존재하여, 데이터 복구가 가능하다.

[0177] 본 발명은, 데이터베이스 파일을 변경하는 명령이 수행될 때마다, 이에 해당되는 로그 엔트리를 생성하여 로그 페이지에 저장하고, 복구(Recovery) 시에, 로그페이지의 로그 레코드를 리두(redo)한다.

[0178] 그러나, 갑작스런 전원 차단 등으로 데이터베이스 파일의 페이지 일부만이 데이터베이스에 저장될 수 있다.

[0179] 이를 톤 페이지 문제(torn-page problem)라고 하는데, 디바이스에 따라 자동 쓰기(automic-write)를 지원하지

않을 수 있기 때문에, 복구 시스템(recovery system)이 반드시 해결해야 할 문제 중 하나이다.

[0180] 본 발명은, 이러한 톤 페이지 문제를 해결하기 위해, 변경되는 데이터베이스 파일의 페이지 상태에 따라 로그를 다르게 저장한다.

[0181] 예를 들면, 현재 변경되는 데이터베이스 파일의 페이지가 이전에 커밋된 트랜잭션에 의해 변경 이력이 있는 경우, 로그 엔트리를 생성하여 로그 페이지에 저장한다.

[0182] 그러나, 만약 현재 변경되는 데이터베이스 파일의 페이지가 과거에 변경된 이력이 없는 경우, 로그 엔트리를 생성하지 않고, 이후 과정인 트랜잭션이 커밋될 때, 변경된 페이지 전체(수행 명령 Opcode: Whole_Page)를 로그 페이지에 저장한다.

[0183] 따라서, 각 데이터베이스 파일의 페이지마다, 최초 한번 씩, 변경된 페이지 전체가 로그 파일에 저장되므로, 데이터 복구시, 이를 통해 원본 페이지를 복구할 수 있다.

[0184] 그리고, 데이터베이스 파일의 페이지별로 수정 이력을 확인하기 위해, 더티 익스피어리언스 비트(Dirty Experience-bit)인 비트맵을 사용할 수 있다.

[0185] 여기서, 비트맵은, 한 페이지 당 1bit 만을 필요로 하므로(0 or 1) 4KB 만으로 32768개의 DB 페이지를 표현할 수 있다.

[0186] 또한, 비트맵은, 쉐어드 메모리 파일(Shared memory File)로 구현되어 여러 프로세스에서 한 DB파일에 접근하는 경우도 지원하며 “<DB파일명>-shm” 형태로 저장될 수 있다.

[0187] 일 예로, 도 13과 같이, 데이터베이스 파일의 페이지가 이전 트랜잭션에 의해 수정되어 그 페이지가 로그 파일에 존재하는 경우, 비트맵은, 1로, 그렇지 않은 경우, 0으로 저장되고, 비트맵은, 1인 페이지의 변정 작업에 대해서만 로그 엔트리가 생성된다.

[0188] 도 15는 체크섬이 저장된 로그 페이지를 보여주는 도면이다.

[0189] 도 15에 도시된 바와 같이, 본 발명의 시스템은, 트랜잭션이 커밋될 때, 로그 파일의 커밋 로그 레코드(commit log record)(1050)에 체크섬(checksum)(1052)을 저장할 수 있다.

[0190] 여기서, 본 발명의 시스템은, 특정 트랜잭션에 의해 생성된 다수의 로그 레코드들이 제 1 로그 페이지로부터 제 2 로그 페이지로 이어질 때, 제 1 로그 페이지의 마지막 지점에 제 1 체크섬(1054)을 저장하고, 특정 트랜잭션의 커밋 로그 레코드에 제 2 체크섬(1056)을 저장할 수 있다.

[0191] 이때, 체크섬은, 로그 레코드와 로그 레코드를 지시하는 로그 포인터를 대상으로 수행될 수 있다.

[0192] 따라서, 본 발명의 시스템은, 데이터베이스 파일의 복구 시에, 체크섬과 로그 레코드의 내용을 비교하고, 체크섬과 로그 레코드의 내용이 다르면, 해당하는 트랜잭션을 실패로 판정하고, 재수행(redo) 과정에서 로그 레코드를 제외시킬 수 있다.

[0193] 이와 같이, 구성되는 본 발명에 따른 데이터베이스 관리 시스템의 데이터 변경 방법을 설명하면 다음과 같다.

[0194] 먼저, 본 발명의 시스템은, 데이터베이스 파일의 변경 요청을 수신할 수 있다.

[0195] 그리고, 시스템은, 변경 요청된 데이터베이스 파일의 트랜잭션을 수행할 수 있다.

[0196] 여기서, 시스템은, 데이터베이스로부터 변경 요청된 데이터베이스 파일을 추출하고, 트랜잭션을 수행하여, 데이터베이스 파일의 페이지를 변경할 수 있다.

[0197] 다음, 시스템은, 트랜잭션이 수행되어 내용이 변경된 데이터베이스 파일의 더티 페이지(dirty page)를 확인하여, 내용이 변경된 더티 페이지 별로 변경 내역에 대한 로그 레코드(log record)를 생성한다.

[0198] 이어, 시스템은, 트랜잭션의 커밋(commit)이 요청되면, 생성된 전체 로그 레코드를 로그 파일의 로그 페이지에 저장한다.

[0199] 그리고, 시스템은, 트랜잭션을 데이터베이스에 커밋하여, 변경된 데이터베이스 파일을 데이터베이스에 저장하고, 그에 상응하는 로그 파일을 데이터베이스에 저장할 수 있다.

[0200] 여기서, 시스템은, 트랜잭션을 데이터베이스에 커밋할 때, 커밋 로그 레코드를 생성할 수 있다.

[0201] 그리고, 시스템은, 생성된 커밋 로그 레코드를 로그 레코드가 저장된 로그 페이지에 삽입하고, 로그 페이지를

로그 파일에 저장할 수 있다.

[0202] 여기서, 로그 페이지는, 버퍼 메모리에 저장될 수 있다.

[0203] 이때, 버퍼 메모리에 저장된 로그 페이지를 로그 파일에 저장할 때, 버퍼 메모리에 저장된 로그 페이지는, 삭제될 수 있다.

[0204] 다음, 시스템은, 로그 파일에 저장되지 않는 잔여 로그 페이지를 검색한다.

[0205] 이어, 시스템은, 검색 결과, 로그 파일에 저장되지 않는 잔여 로그 페이지가 있으면, 잔여 로그 페이지를 로그 파일에 저장하고, 트랜잭션을 데이터베이스에 커밋할 수 있다.

[0206] 또한, 시스템은, 생성된 전체 로그 레코드를 로그 파일에 저장할 때, 체크포인트 로그 레코드를 생성할 수 있다.

[0207] 그리고, 시스템은, 생성된 체크포인트 로그 레코드를 로그 레코드가 저장된 로그 페이지에 삽입하고, 로그 페이지를 로그 파일에 저장할 수 있다.

[0208] 여기서, 로그 페이지는, 버퍼 메모리에 저장될 수 있다.

[0209] 이때 버퍼 메모리에 저장된 로그 페이지를 로그 파일에 저장할 때, 버퍼 메모리에 저장된 로그 페이지는, 삭제될 수 있다.

[0210] 다음, 시스템은, 로그 파일에 저장되지 않는 잔여 로그 페이지를 검색하고, 검색 결과, 로그 파일에 저장되지 않는 잔여 로그 페이지가 있으면, 잔여 로그 페이지를 로그 파일에 저장하며, 생성된 전체 로그 레코드를 로그 파일에 저장할 수 있다.

[0211] 도 16은 본 발명 일 실시예에 따른 데이터베이스 관리 시스템의 데이터 복구 방법을 설명하기 위한 흐름도이다.

[0212] 도 16에 도시된 바와 같이, 본 발명의 시스템은, 데이터 복구 요청을 수신할 수 있다.(S10)

[0213] 그리고, 본 발명의 시스템은, 복구하고자 하는 데이터베이스 파일에 대한 로그 파일의 존재를 확인한다.(S20)

[0214] 이어, 본 발명의 시스템은, 로그 파일이 존재하면, 로그 파일의 마지막 로그 레코드로부터 마지막 커밋 로그 레코드까지 역순서로 검색하여(S30), 복구하고자 하는 데이터베이스 파일에 대한 원본 페이지가 존재하는지를 확인한다.(S40)

[0215] 다음, 본 발명의 시스템은, 원본 페이지가 존재하면, 원본 페이지를 데이터베이스 파일에 저장한다.(S50)

[0216] 그리고, 본 발명의 시스템은, 로그 파일의 마지막 체크포인트 로그 레코드로부터 마지막 커밋 로그 레코드까지 정순서로 검색하여(S60), 그들 사이에 로그 레코드가 존재하는지를 확인한다.(S70)

[0217] 다음, 본 발명의 시스템은, 확인 결과, 그들 사이에 로그 레코드가 존재하면, 로그 레코드를 재수행한다.(S80)

[0218] 이어, 본 발명의 시스템은, 로그 파일의 로그 레코드를 검색하여(S90), 로그 파일의 마지막 커밋 로그 레코드 이후에 로그 레코드가 존재하는지를 확인한다.(S100)

[0219] 그리고, 본 발명의 시스템은, 로그 파일의 마지막 커밋 로그 레코드 이후에 로그 레코드가 존재하면, 해당하는 로그 레코드를 삭제한다.(S110)

[0220] 이와 같이, 본 발명은, 변경된 페이지 전체를 로깅하지 않고, 페이지마다 DB파일을 변경하는 연산만을 로깅하므로, 로그 파일의 크기를 줄이고 전체 I/O 부하를 낮출 수 있다.

[0221] 그리고, 본 발명은, 체크포인트 시, DB 파일에 대한 싱크(sync) 연산만을 수행하므로 추가적인 마이그레이션 비용(migration cost)이 필요한 WAL 방식에 비해 체크포인트 속도가 빠르고, 이를 통해 ANR(Appplication Not Responding)의 문제를 해결할 수 있다.

[0222] 또한, WAL 방식의 경우, 읽기 시, WAL 파일을 무조건 한번 검색하므로 선택(Select) 연산에 불리하지만, 본 발명은, DB 파일만 검색하므로 검색 비용이 더 저렴하다.

[0223] 이어, 본 발명은, 다른 방식보다 월등한 성능을 보이고 있어, 기존 방식의 문제점을 해결할 수 있다.

[0224] 한편, 본 발명은, 다양한 상황에서 데이터 복구를 수행할 수 있다.

[0225] 첫째, 본 발명은, 트랜잭션 실패(Transaction Failure) 상황에서, 데이터 복구가 가능하다.

[0226] 실행중인 트랜잭션이 특정한 원인에 인해 더 이상 수행될 수 없는 경우, 트랜잭션 실패가 발생한다.

[0227] 트랜잭션 실패는, 오버플로우, 자원 부족, 교착상태 등으로 인해 발생하는데, 이러한 경우 해당 트랜잭션은, 롤백되어 트랜잭션 시작 직전의 상태로 되돌아가야 한다.

[0228] 본 발명은, 트랜잭션이 롤백될 때, 로그 페이지 내에 마지막으로 저장된 레코드부터 이전 커밋 레코드까지 역탐색을 한다.

[0229] 탐색 도중, Pager_stressed 로그 레코드를 만나는 경우, 원본 페이지를 데이터베이스 페이지에 덮어쓴다.

[0230] Pager_stressed 로그 레코드 이외의 로그 레코드는, 데이터베이스에 아직 반영되기 전이므로 무시한다.

[0231] 이어, 커밋 레코드에 도달하면, 커밋 레코드 이후의 모든 로그 레코드를 삭제한다.

[0232] 둘째, 본 발명은, 시스템 충돌(System Crash) 상황에서, 데이터 복구가 가능하다.

[0233] 시스템 충돌은, 하드웨어 오작동, 데이터베이스 소프트웨어 또는 운영체제의 버그 등으로 인해 휘발성 저장장치의 데이터 손실이 발생하고, 트랜잭션 수행을 중단되는 것을 뜻한다.

[0234] 단, 비휘발성 저장장치의 데이터베이스 파일 및 로그 파일은, 손실되지 않는 것을 가정한다.

[0235] 시스템 충돌이 발생한 시점에 따라, 데이터베이스 파일 및 로그 파일의 상태가 다르기 때문에 복구 기법은, 각 상황에 유동적으로 대처할 수 있어야 한다.

[0236] 본 발명은, 시스템 충돌에 의해 발생하는 다음과 같은 상황에 대처할 수 있다.

[0237] 일 예로, 로그 파일에 시스템 충돌 시, 수행 중이던 트랜잭션의 커밋 로그가 존재하지 않는 경우, 트랜잭션에 의해 수정된 페이지가 데이터베이스 파일에 반영되기 이전에는, 커밋 로그가 없으므로, 해당 트랜잭션이 수행되기 이전 상태로 돌아가야 한다.

[0238] 따라서, 본 발명은, Pager_stressed 로그 레코드에 대한 처리를 하고, 트랜잭션과 관련된 로그 레코드를 삭제한다.

[0239] 다른 예로, 로그 파일에 시스템 충돌 시, 수행 중이던 트랜잭션의 커밋 로그가 존재하는 경우, 로그 파일에는, 트랜잭션을 리두(redo)하기 위한 모든 데이터가 들어있으나, 데이터베이스 파일에는, 트랜잭션에 의해 변경된 사항이 전부 반영되었는지 알 수 없다.

[0240] 따라서, 본 발명은, 복구 알고리즘을 수행하면서, 해당 트랜잭션과 관련된 로그 레코드를 모두 리두(redo)할 수 있다.

[0241] 셋째, 본 발명은, 디스크 실패(Disk Failure) 상황에서, 데이터 복구가 가능하다.

[0242] 디스크 자체의 손상으로 인해, 디스크 블록의 데이터는, 소실될 수도 있다.

[0243] 따라서, 본 발명은, 데이터베이스 파일과 로그 파일이 같은 경로에 저장되므로, 동일한 디스크에 저장된다.

[0244] 디스크 실패가 발생하여 특정 DB페이지가 손상을 입더라도, 로그 파일에는, 해당 페이지가 저장되어 있으므로, 데이터를 복구할 수 있다.

[0245] 도 17 내지 도 20은 본 발명 일 실시예에 따른 데이터베이스 관리 시스템의 복구 성능을 보여주는 도면이다.

[0246] 도 17은, RL 벤치마크(benchmark)를 이용하여, 체크포인트 수행 횟수에 따른 수행 시간을 비교한 그래프이다.

[0247] 도 17에 도시된 바와 같이, 체크포인트 수행 횟수에 따른 수행 시간을 비교한 결과, 본 발명의 DWAL 방식은, 기존의 WAL 방식에 비해, 수행 시간이 약 13.4 - 22.85% 감소한 것을 알 수 있다.

[0248] 따라서, 본 발명의 복구 방식이, 기존 방식에 비해, 복구 시간을 줄일 수 있다는 것을 알 수 있다.

[0249] 도 18은 최초의 커밋 수행 없이, 1000개의 로그 레코드를 한 번에 로그 파일에 삽입하는데 걸리는 시간을 비교한 그래프이다.

[0250] 여기서, 가로축의 50, 1000, 1000000은, 체크포인트의 수행 범위를 의미한다.

[0251] 도 18에 도시된 바와 같이, 1000개의 로그 레코드를 한 번에 로그 파일에 삽입하는데 걸리는 시간을 비교한 결

과, 본 발명의 DWAL 방식은, 기존의 WAL 방식에 비해, 수행 처리 시간이 빠른 것을 알 수 있다.

[0252] 따라서, 본 발명의 복구 방식이, 기존 방식에 비해, 약 2배 내지 약 6배 정도 좋은 성능을 보인다는 것을 알 수 있다.

[0253] 도 19는 다수의 트랜잭션 수행 시, 25000개의 로그 레코드를 한 번에 로그 파일에 삽입하는데 걸리는 시간을 비교한 그래프이다.

[0254] 여기서, 가로축의 5, 10, 20, 50, 100은, 하나의 트랜잭션당 삽입되는 로그 레코드의 개수를 의미한다.

[0255] 도 19에 도시된 바와 같이, 25000개의 로그 레코드를 한 번에 로그 파일에 삽입하는데 걸리는 시간을 비교한 결과, 본 발명의 DWAL 방식은, 기존의 WAL 방식에 비해, 수행 처리 시간이 빠른 것을 알 수 있다.

[0256] 따라서, 본 발명의 복구 방식이, 기존 방식에 비해, 좋은 성능을 보인다는 것을 알 수 있다.

[0257] 도 20은, RL 벤치마크(benchmark)를 이용하여, 생성되는 로그 파일의 크기를 측정한 도표이다.

[0258] 본 발명은, 더티 페이지의 크기와 전체 로그 레코드의 크기를 비교하여, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 작으면, 생성된 전체 로그 레코드를 로그 파일에 저장하는 제 1 로깅 방식을 수행하고, 생성된 전체 로그 레코드의 크기가 더티 페이지의 크기보다 더 크면, 더티 페이지를 로그 파일에 저장하는 제 2 로깅 방식을 수행할 수 있다.

[0259] 즉, 제 1 로깅 방식은, 변경 이력만을 로깅하는 page-oriented logging only 방식이고, 제 2 로깅 방식은, 더티 페이지의 크기와 전체 로그 레코드의 크기를 비교한 결과에 따라, 변경 이력 로그와 변경본 로그 중, 작은 로그를 선택하는 bimodal logging 방식이다.

[0260] 도 20에 도시된 바와 같이, 본 발명은, 제 2 로깅 방식으로 로깅을 수행할 경우, 로그 파일의 크기가 기존 방식보다 더 많이 줄어들었음을 알 수 있다.

[0261] 이와 같이, 본 발명은, 변경된 페이지 전체를 로깅하지 않고 페이지마다 데이터베이스 파일을 변경하는 연산만을 로깅함으로써, 로그 파일의 크기를 줄이고 전체 입/출력 부하를 낮출 수 있다.

[0262] 그리고, 본 발명은, 체크 포인트 시, 데이터베이스 파일에 대한 싱크(sync) 연산만을 수행하므로, 추가적인 마이그레이션 비용(migration cost)가 필요한 WAL 방식에 비해, 체크포인트 속도를 개선할 수 있으며, 이를 통해 ANR(Application Not Responding)의 문제 해결이 가능하다.

[0263] 또한, 본 발명은, 읽기 시, WAL 방식의 경우, WAL 파일을 무조건 한 번 검색해야 하므로, 선택(select) 연산에 불리하지만, 본 발명의 경우, 데이터베이스 파일만을 검색하므로, 검색 비용을 더 낮출 수 있다.

[0264] 이상, 본 발명은 본 발명의 정신 및 필수적 특징을 벗어나지 않는 범위에서 다른 특정한 형태로 구체화될 수 있음은 당업자에게 자명하다.

[0265] 전술한 본 발명은, 프로그램이 기록된 매체에 컴퓨터가 읽을 수 있는 코드로서 구현하는 것이 가능하다. 컴퓨터가 읽을 수 있는 매체는, 컴퓨터 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록장치를 포함한다. 컴퓨터가 읽을 수 있는 매체의 예로는, ROM, RAM, CD-ROM, 자기 테이프, 플로피 디스크, 광 데이터 저장 장치 등이 있으며, 또한 캐리어 웨이브(예를 들어, 인터넷을 통한 전송)의 형태로 구현되는 것도 포함한다. 또한, 상기 컴퓨터는 단말기의 제어부를 포함할 수도 있다.

[0266] 따라서, 상기의 상세한 설명은 모든 면에서 제한적으로 해석되어서는 아니되고 예시적인 것으로 고려되어야 한다. 본 발명의 범위는 첨부된 청구항의 합리적 해석에 의해 결정되어야 하고, 본 발명의 등가적 범위 내에서의 모든 변경은 본 발명의 범위에 포함된다.

부호의 설명

[0267] 100: 데이터베이스

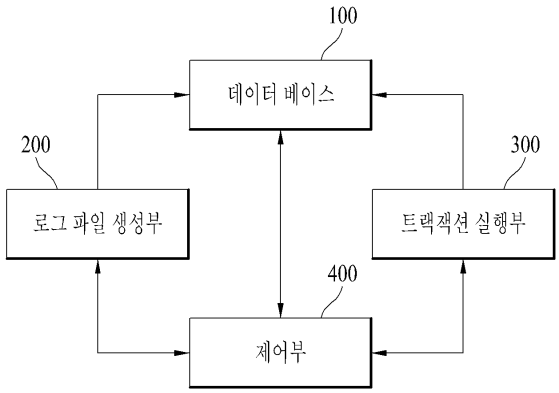
200: 로그파일 생성부

300: 트랜잭션 수행부

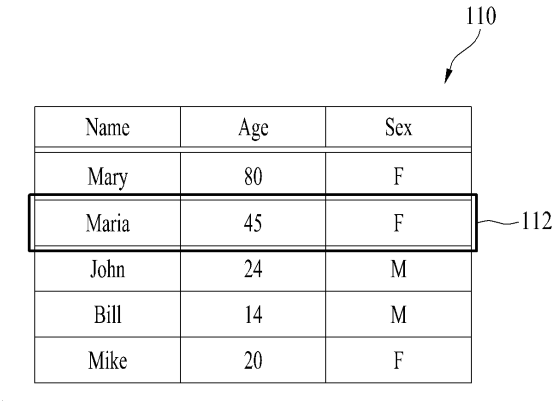
400: 제어부

도면

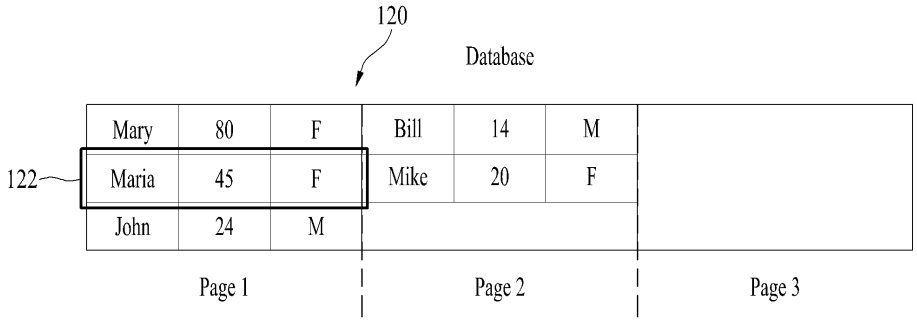
도면1



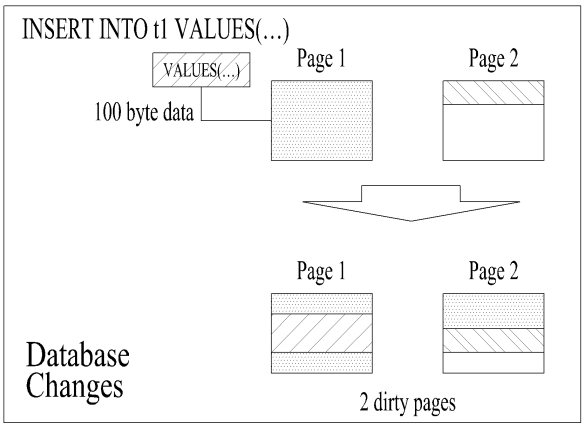
도면2



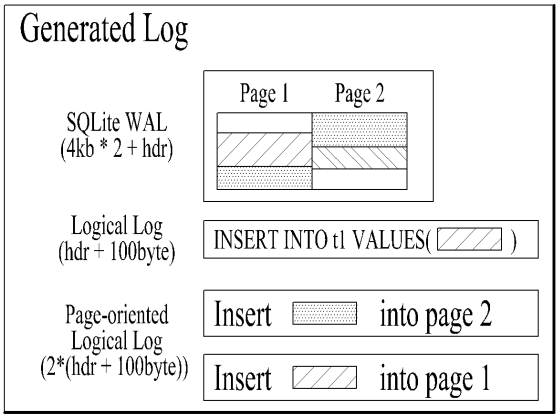
도면3



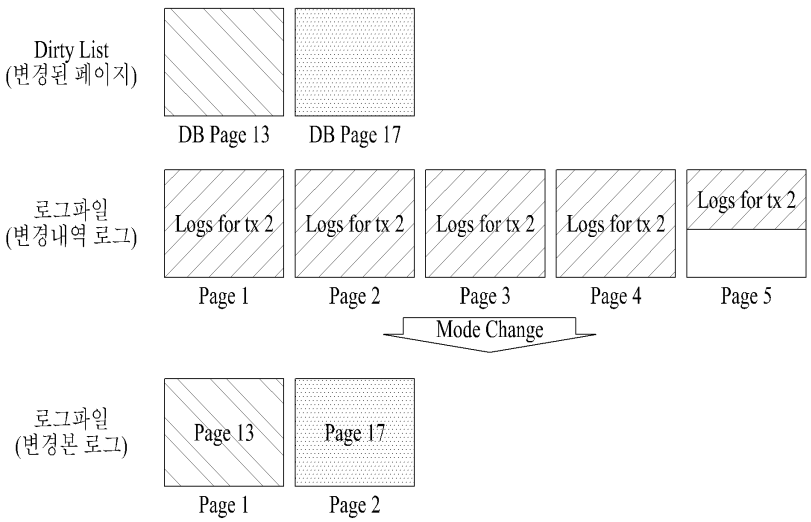
도면4


INSERT INTO t1 VALUES(...)
VALUES(...)
100 byte data
Page 1
Page 2
Page 1
Page 2
Database Changes
2 dirty pages

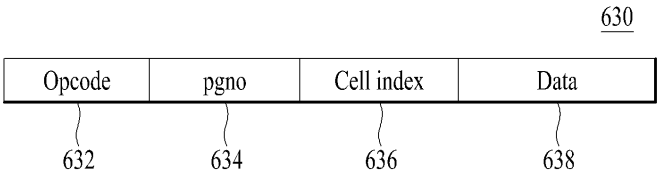
도면5


Generated Log
SQLite WAL
(4kb * 2 + hdr)
Page 1
Page 2
Logical Log
(hdr + 100byte)
INSERT INTO t1 VALUES( )
Page-oriented Logical Log
(2*(hdr + 100byte))
Insert into page 2
Insert into page 1

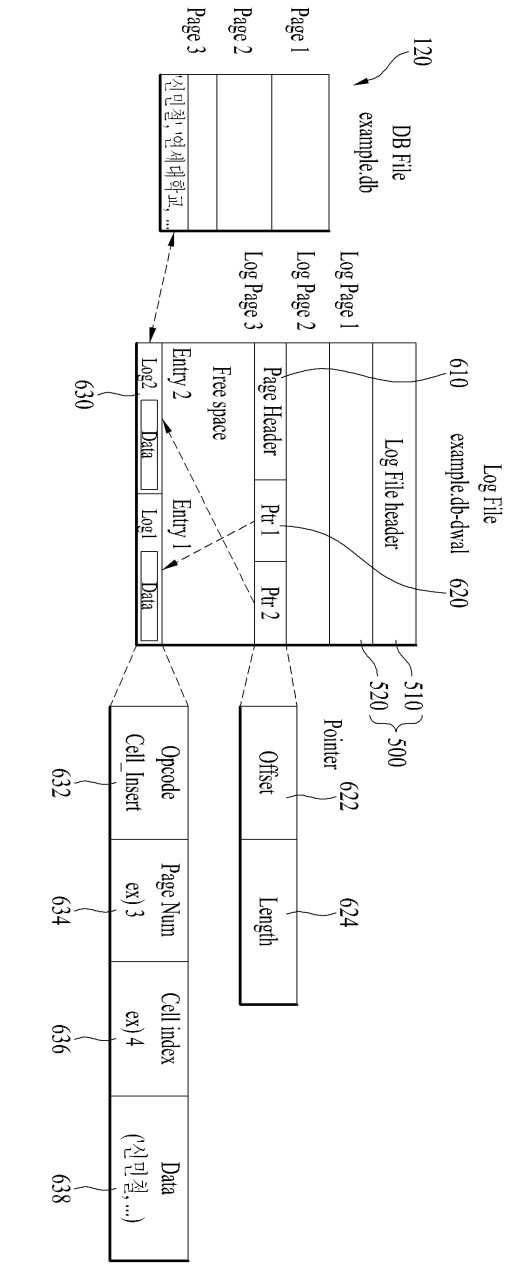
도면6


Dirty List
(변경된 페이지)
DB Page 13
DB Page 17
로그파일
(변경내역 로그)
Logs for tx 2
Logs for tx 2
Logs for tx 2
Logs for tx 2
Logs for tx 2
Page 1
Page 2
Page 3
Page 4
Page 5
Mode Change
로그파일
(변경본 로그)
Page 13
Page 17
Page 1
Page 2

도면7


630
Opcode
pgno
Cell index
Data
632
634
636
638

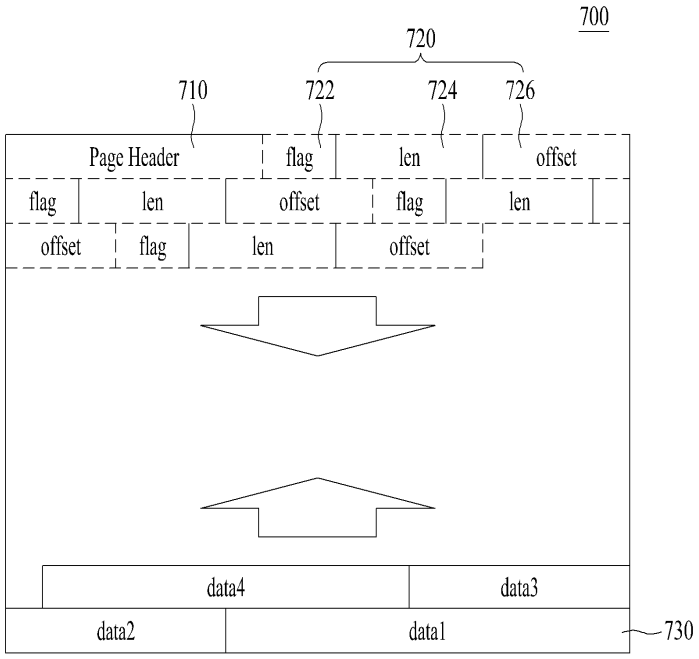
도면8


Page 3
Page 2
Page 1
120
DB File
example.db
Log Page 3
Log Page 2
Log Page 1
630
Log2
Data
Entry 2
Free space
Entry 1
Log1
Data
Page Header
Ptr 1
Ptr 2
610
620
Log File header
Log File
example.db-dwal
510
520
500
Pointer
622
Offset
Length
624
632
Opcode
Cell_Insert
634
Page Num
ex) 3
636
Cell index
ex) 4
638
Data
(신민철,...)

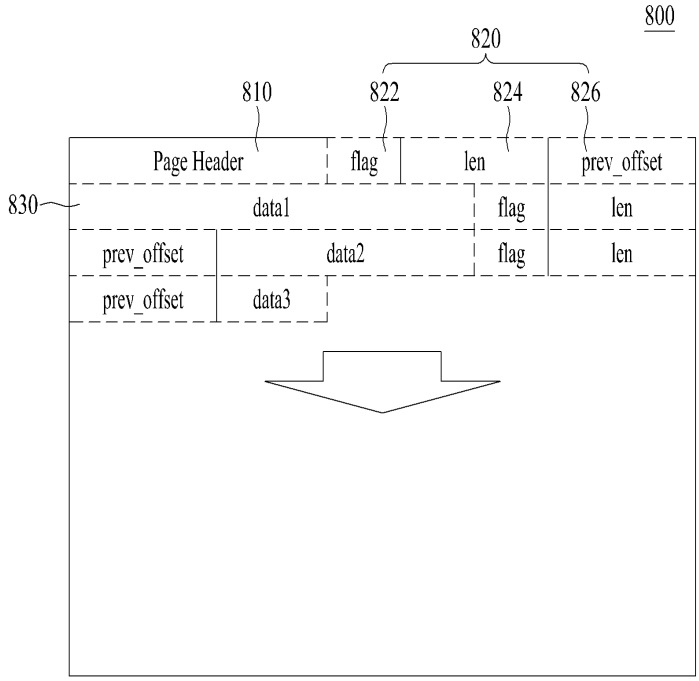
도면9

Opcode	SQLite Function	Description
Cell_Insert	insertCell()	특정 페이지에 셀 삽입
Cell_Drop	dropCell()	특정 페이지의 특정 셀을 삭제
Copy_Cells	copyNodeContent()	B-트리 데이블 밸런싱 관련 작업 수행
Assemble_Page	balance_nonroot( ) balance_quick( )	
Page_alloc	allocateBtreePage( )	새로운 페이지 할당
Page_empty	zeroPage( )	페이지 초기화
Page_erase	freePage2( )	특정 페이지를 free-list에 삽입
Page_stressed	pagerStress( )	트랜잭션 커밋 이전에 dirty 페이지를 write
Commit	sqlite3PagerCommitPhaseOne( )	현재 수행중인 트랜잭션이 커밋
Checkpoint	sqlite3PagerCheckpoint( )	체크포인트 연산 수행
Hdr_change	various functions	DB 파일 헤더 및 DB 페이지 헤더 수정
Whole_Page	NULL	DB 전체 페이지 로깅에 사용

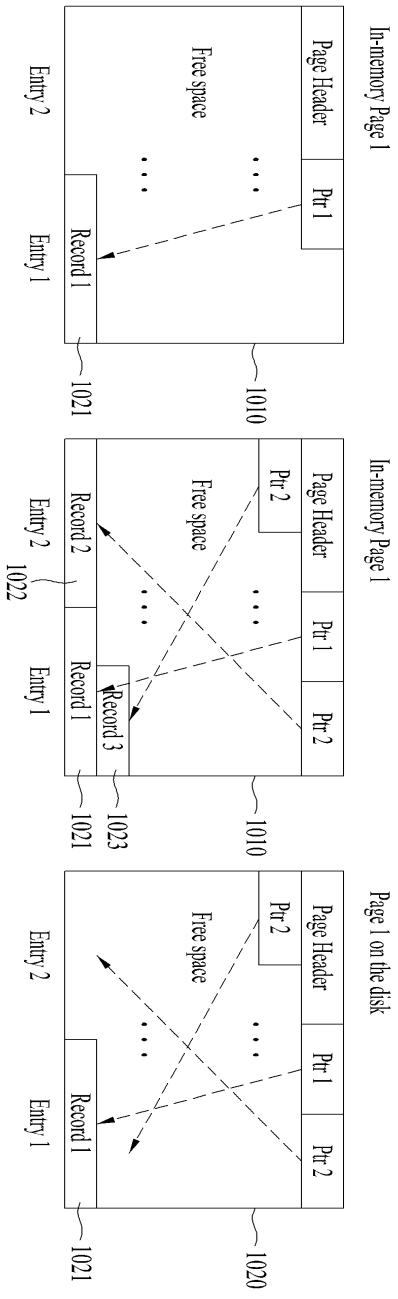
도면10



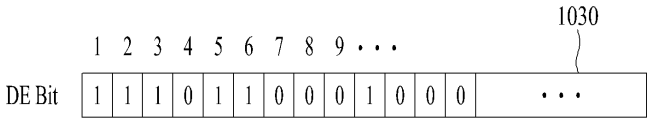
도면11


800
820
810
822
824
826
Page Header
flag
len
prev_offset
830
data1
flag
len
prev_offset
data2
flag
len
prev_offset
data3

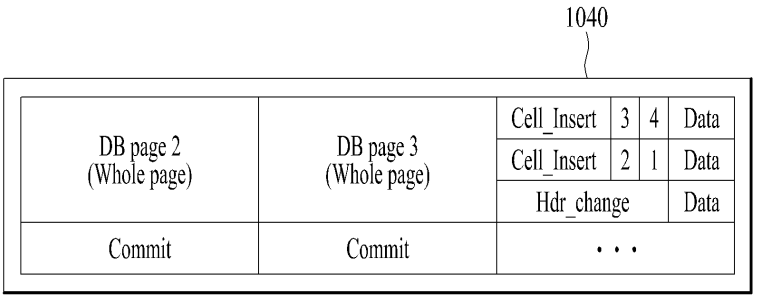
도면12


In-memory Page 1
Page Header
Ptr 1
Free space
Entry 2
Entry 1
Record 1
1021
1010
In-memory Page 1
Page Header
Ptr 2
Ptr 1
Ptr 2
Free space
Entry 2
Record 2
1022
Entry 1
Record 1
Record 3
1021
1023
1010
Page 1 on the disk
Page Header
Ptr 2
Ptr 1
Ptr 2
Free space
Entry 2
Entry 1
Record 1
1021
1020

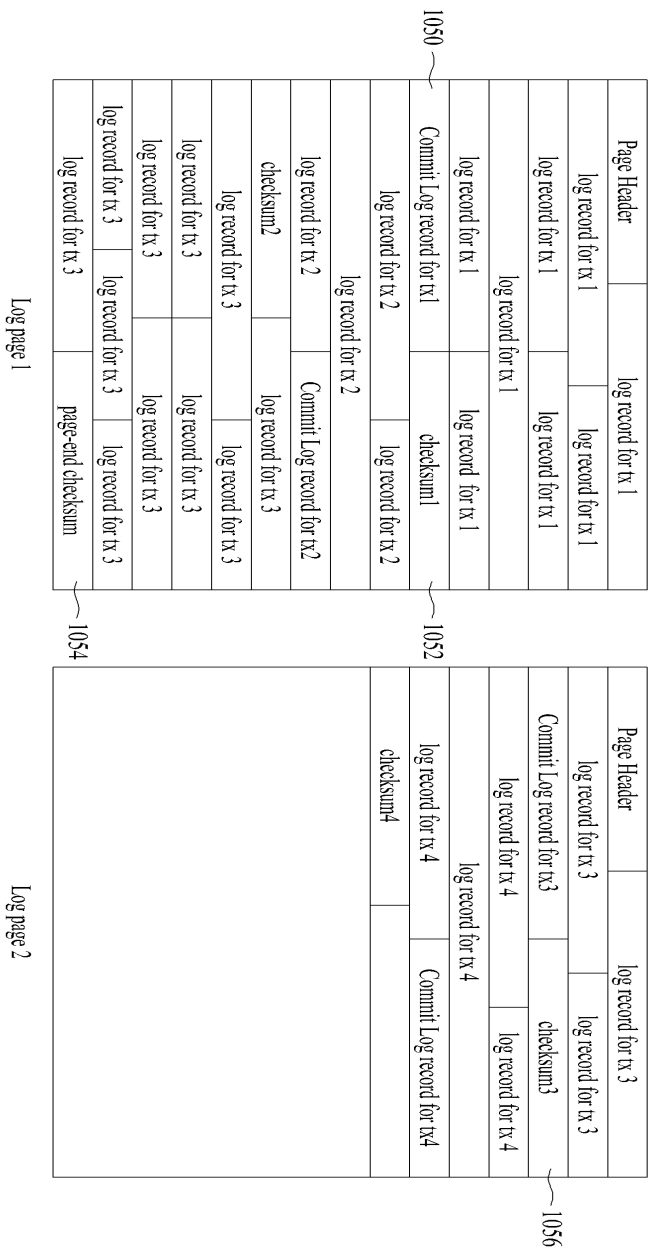
도면13


1030
1 2 3 4 5 6 7 8 9 • • •
DE Bit
1 1 1 0 1 1 0 0 0 1 0 0 0 • • •

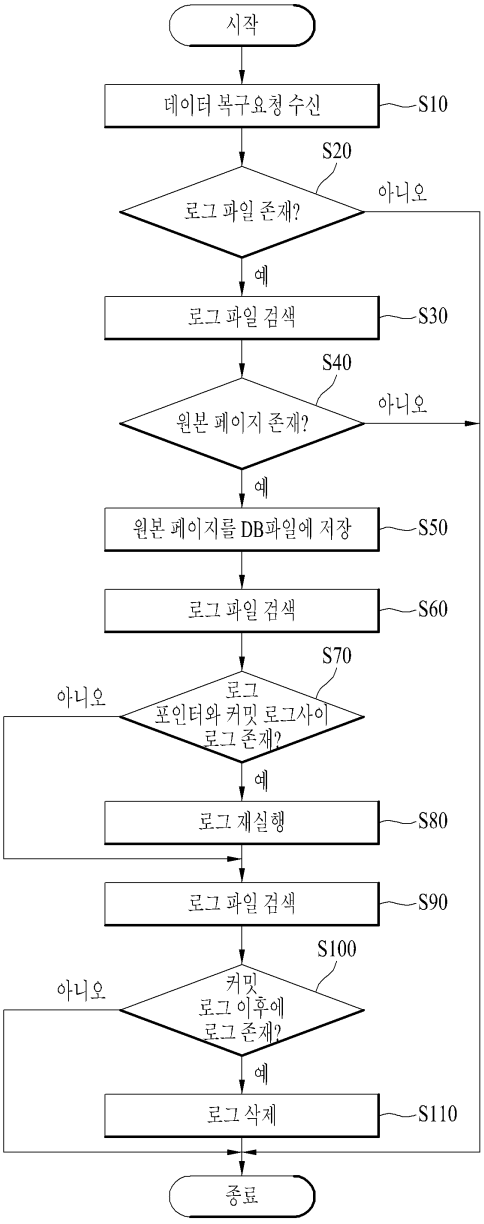
도면14


1040
DB page 2 (Whole page)
DB page 3 (Whole page)
Cell_Insert 3 4 Data
Cell_Insert 2 1 Data
Hdr_change Data
Commit
Commit
• • •

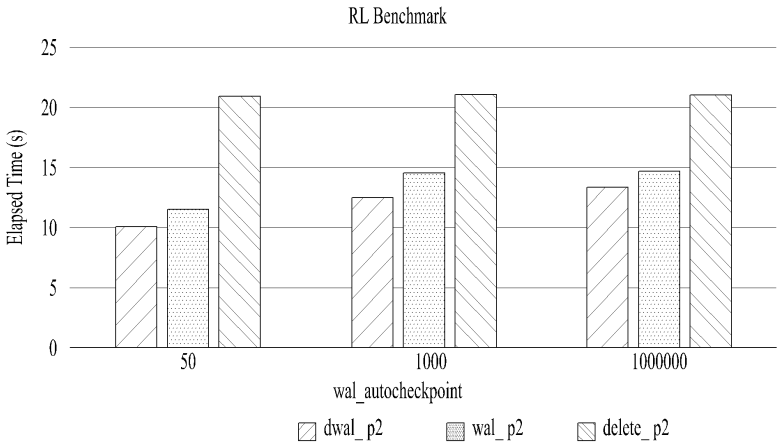
도면15


1050
1052
1054
1056
Log page 1
Log page 2
Page Header
log record for tx 1
log record for tx 1
log record for tx 1
log record for tx 1
log record for tx 1
log record for tx 1
log record for tx 1
log record for tx 1
log record for tx 2
log record for tx 2
log record for tx 2
log record for tx 2
log record for tx 2
log record for tx 2
Commit Log record for tx2
checksum2
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
page-end checksum
Page Header
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 3
log record for tx 4
log record for tx 4
log record for tx 4
log record for tx 4
log record for tx 4
log record for tx 4
Commit Log record for tx4
checksum4
Commit Log record for tx3
checksum3

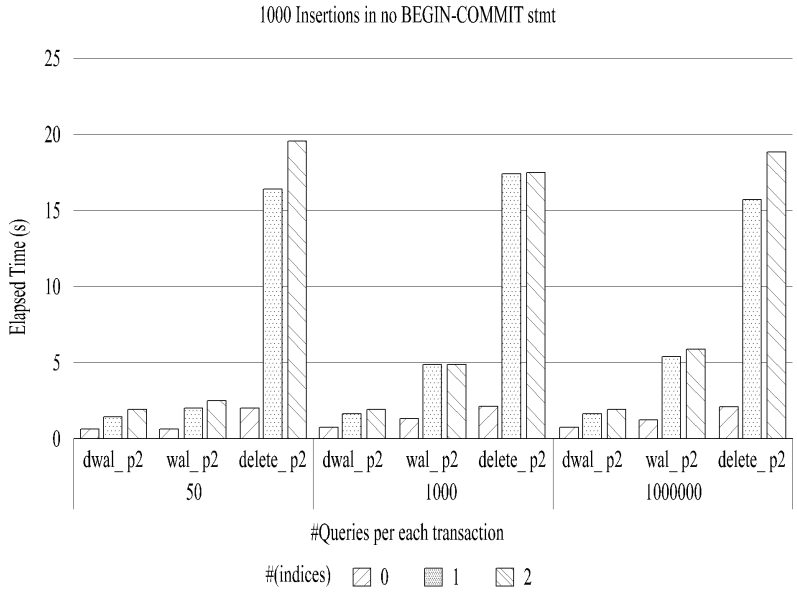
도면16


시작
데이터 복구요청 수신
S10
S20
로그 파일 존재?
아니오
예
로그 파일 검색
S30
S40
원본 페이지 존재?
아니오
예
원본 페이지를 DB파일에 저장
S50
로그 파일 검색
S60
S70
로그
포인터와 커밋 로그사이
로그 존재?
아니오
예
로그 재실행
S80
로그 파일 검색
S90
S100
커밋
로그 이후에
로그 존재?
아니오
예
로그 삭제
S110
종료

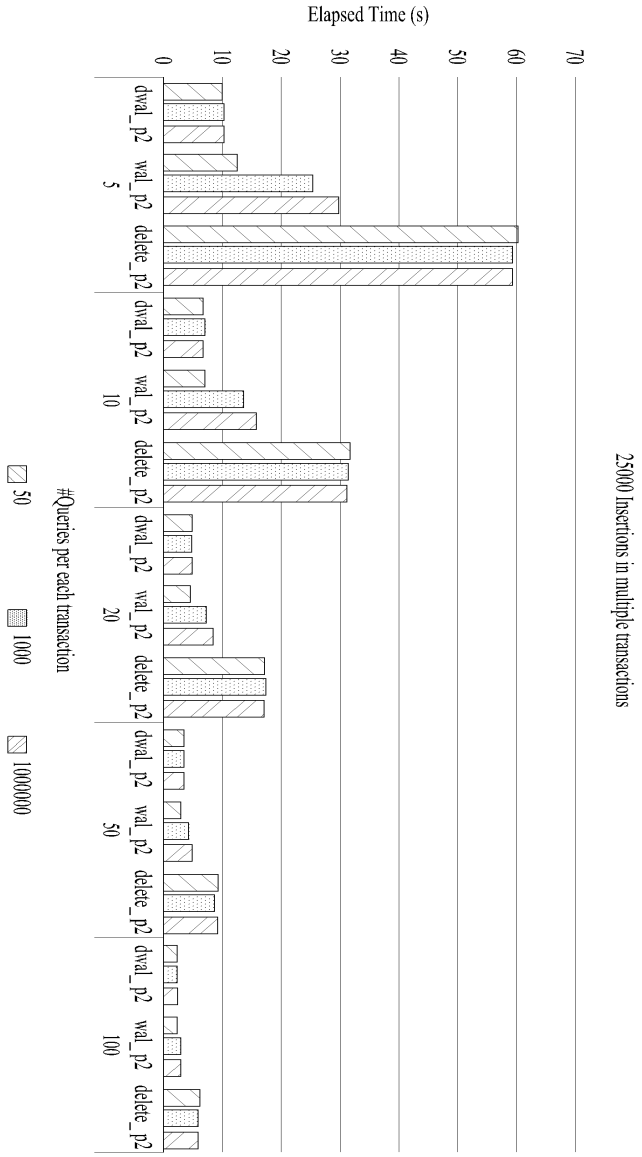
도면17


RL Benchmark
25
20
15
10
5
0
Elapsed Time (s)
50
1000
1000000
wal_autocheckpoint
dwal_p2
wal_p2
delete_p2

도면18


1000 Insertions in no BEGIN-COMMIT stmt
25
20
15
10
5
0
Elapsed Time (s)
dwal_p2
wal_p2
delete_p2
50
1000
1000000
#Queries per each transaction
#(indices)
0
1
2

도면19



도면20

Transaction	DWAL page-oriented logging only	DWAL-Bimodal Logging	WAL	상대크기 (WAL 파일 = 1)
1000 insertion into t1 (no "BEGIN-COMMIT" stmt)	200 KB	236 KB	4152 KB	x0.06
25000 insertion into t2 (in one transaction stmt)	1,004 KB	996 KB	976 KB	x1.02
25000 insertion into t3 (in one transaction stmt)	1,008 KB	992 KB	972 KB	x1.02
1000 range updates on t1	23,904 KB	60 KB	56 KB	x1.07
25000 updates on t2	3,380 KB	1340 KB	1336 KB	x1.00
Insert into t1 from t2; Insert into t2 from t1;	16,492 KB	3184 KB	3200 KB	x1.00
Deletion with "LIKE" statement	1,304 KB	2528 KB	3180 KB	x0.79
Deletion using Range	5,480 KB	2536 KB	1340 KB	x1.89
Drop t1	40 KB	36 KB	692 KB	x0.05