



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2020-0062348
(43) 공개일자 2020년06월03일

(51) 국제특허분류(Int. Cl.)
G06F 16/2458 (2019.01) G06F 16/18 (2019.01)
G06F 16/23 (2019.01) H04L 9/30 (2006.01)
(52) CPC특허분류
G06F 16/2471 (2019.01)
G06F 16/1873 (2019.01)
(21) 출원번호 10-2020-7014386
(22) 출원일자(국제) 2018년10월31일
심사청구일자 2020년05월20일
(85) 번역문제출일자 2020년05월20일
(86) 국제출원번호 PCT/US2018/058432
(87) 국제공개번호 WO 2019/089742
국제공개일자 2019년05월09일
(30) 우선권주장
62/580,056 2017년11월01일 미국(US)

(71) 출원인
스웰즈, 인크.
미국 75080 텍사스주 리처드슨 노스 센트럴 익스
프레스웨이 3400 스위트 470
(72) 발명자
베어드, 리몬 씨., 3세
미국 75080 텍사스주 리처드슨 에스티이. 470 노
스 센트럴 익스프레스웨이 3400
하몬, 멘스
미국 77845 텍사스주 칼리지 스테이션 서던 힐스
코트 800
(74) 대리인
양영준, 김연송, 백만기

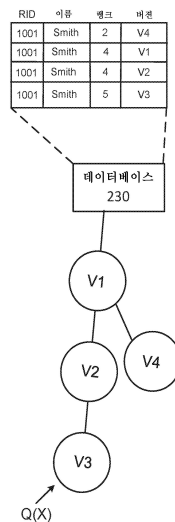
전체 청구항 수 : 총 20 항

(54) 발명의 명칭 고속 카피가능 데이터베이스를 효율적으로 구현하기 위한 방법들 및 장치

(57) 요약

고속 카피가능 데이터베이스 장치는 프로세서, 및 프로세서와 전자 통신하는 메모리를 포함한다. 메모리는 버전 식별자를 포함하는 복합 키를 각각 갖는 다수의 레코드들을 갖는 데이터베이스 테이블을 저장한다. 메모리는 또한 프로세서로 하여금, 데이터베이스 테이블의 제1 가상 클론 및 데이터베이스 테이블의 제2 가상 클론을 생성하고, 제1 가상 클론에서 질의를 수신하고, 질의를 수정하여 제1 가상 클론의 버전 식별자를 포함하게 하도록 실행 가능한 명령어들을 저장한다. 프로세서는 수정된 질의를 이용하여 데이터베이스 테이블에 질의하여, 제1 가상 클론의 버전 식별자의 조상 경로와 연관된 버전 식별자를 갖는 레코드들의 세트로부터 각각의 레코드에 부분적으로 기초하여 응답을 생성하고, 응답을 전송할 수 있다. 각각의 복합 키는 버전 식별자 및 데이터 부분을 포함한다.

대표도 - 도2a



(52) CPC특허분류

G06F 16/2365 (2019.01)

H04L 9/30 (2013.01)

명세서

청구범위

청구항 1

장치로서,

프로세서; 및

상기 프로세서와 전자 통신하는 메모리—상기 메모리는 복수의 레코드들을 포함하는 데이터베이스 테이블을 저장하고, 상기 복수의 레코드들로부터의 각각의 레코드는 버전 식별자를 포함하는 복합 키를 가짐—를 포함하고,

상기 메모리는, 상기 프로세서로 하여금:

상기 데이터베이스 테이블의 제1 가상 클론 및 상기 데이터베이스 테이블의 제2 가상 클론을 생성하고;

상기 제1 가상 클론에서 질의를 수신하고;

상기 제1 가상 클론에서 상기 질의를 수신하는 것에 기초하여, 상기 제1 가상 클론과 연관된 버전 식별자를 포함하는 수정된 질의를 정의하기 위해 상기 질의를 수정하고;

상기 수정된 질의를 이용하여 상기 데이터베이스 테이블에 질의하여, 상기 복수의 레코드들로부터의 레코드들의 세트를 포함하는 응답을, (1) 상기 질의를 충족시키는 상기 레코드들의 세트로부터의 각각의 레코드; 및 (2) 상기 제1 가상 클론과 연관된 상기 버전 식별자의 조상 경로와 연관된 버전 식별자를 갖는 상기 레코드들의 세트로부터의 각각의 레코드에 기초하여, 생성하고;

상기 레코드들의 세트를 포함하는 상기 응답을 상기 질의에 대한 응답으로서 전송하게 하도록 실행가능한 명령어들을 포함하고,

상기 복수의 레코드들로부터의 각각의 레코드에 대한 상기 복합 키는 그 복합 키에 대한 상기 버전 식별자 및 데이터 부분을 포함하고, 상기 복수의 레코드들은, (1) 상기 데이터 부분의 값 및 상기 제1 가상 클론과 연관된 상기 버전 식별자를 포함하는 복합 키; 또는 (2) 상기 데이터 부분의 값 및 상기 제2 가상 클론과 연관된 상기 버전 식별자를 포함하는 복합 키 중 하나만이 임의의 주어진 시간에 상기 데이터베이스 테이블에 존재하도록 구성되는, 장치.

청구항 2

제1항에 있어서,

상기 메모리는 상기 프로세서로 하여금:

상기 제1 가상 클론을 통해 수신된 삽입 명령어에 응답하여, 레코드를 상기 데이터베이스 테이블에 추가하게 하도록 실행가능한 명령어들을 더 포함하고, 상기 레코드는, (1) 상기 제1 가상 클론에서 후속하여 수신되고 상기 데이터를 참조하는 질의가 상기 레코드를 반환하고; (2) 상기 데이터에 대해 제3 가상 클론에서 후속하여 수신되는 질의가 상기 레코드를 반환하지 않도록, 상기 제1 가상 클론과 연관된 버전 식별자를 갖는 복합 키 및 데이터를 포함하는, 장치.

청구항 3

제1항에 있어서,

상기 명령어들은 상기 프로세서로 하여금:

상기 데이터베이스 테이블의 제1 레코드의 필드의 값을 제1 값으로부터 제2 값으로 업데이트하라는, 제3 가상 클론을 통해 수신된 명령어에 응답하여, 제2 레코드를 상기 데이터베이스 테이블에 추가하고—상기 제2 레코드는 상기 제2 값 및 상기 제3 가상 클론과 연관된 버전 식별자를 갖는 복합 키를 포함함—;

제1 레코드의 상기 필드의 값을 업데이트하라는 상기 명령어에 응답하여, 제1 레코드의 복합 키를, 상기 제3 가상 클론과 연관된 상기 버전 식별자를 포함하지 않도록 수정—상기 수정에 의해, 상기 제3 가상 클론에서 후속

하여 수신되고 상기 필드를 참조하는 질의가 상기 제2 레코드를 반환하고, 상기 제1 가상 클론에서 수신되고 상기 필드를 참조하는 질의가 상기 제1 레코드를 반환함—하게 하도록 더 실행가능한, 장치.

청구항 4

제1항에 있어서,

상기 명령어들은 상기 프로세서로 하여금:

상기 복수의 레코드들로부터의 레코드를 삭제하라는, 제3 가상 클론을 통해 수신된 명령어에 응답하여, 상기 레코드의 상기 복합 키의 상기 버전 식별자를, 상기 제3 가상 클론과 연관된 버전 식별자를 포함하지 않도록 수정—상기 수정에 의해, 상기 제3 가상 클론에서 후속하여 수신된, 상기 레코드에 대한 질의가 상기 레코드를 반환하지 않고, 상기 제1 가상 클론에서 후속하여 수신된, 상기 레코드에 대한 질의가 상기 레코드를 반환함—하게 하도록 더 실행가능한, 장치.

청구항 5

제1항에 있어서,

상기 응답은 제1 응답이고, 상기 수정된 질의는 제1 수정된 질의이고, 상기 명령어들은 상기 프로세서로 하여금:

상기 제1 응답이 널인 경우 상기 제2 가상 클론의 버전 식별자를 포함하는 제2 수정된 질의를 생성하고—상기 제2 가상 클론은 상기 제1 가상 클론에 대한 조상임—;

상기 제2 수정된 질의를 이용하여 상기 데이터베이스 테이블에 질의하여, 상기 복수의 레코드들로부터의 레코드들의 세트를 포함하는 제2 응답을, (1) 상기 제2 수정된 질의를 충족시키는 상기 제2 응답의 상기 레코드들의 세트로부터의 각각의 레코드; 및 (2) 상기 제2 가상 클론의 상기 버전 식별자의 조상 경로와 연관된 버전 식별자를 갖는 상기 제2 응답의 상기 레코드들의 세트로부터의 각각의 레코드에 기초하여, 생성하고;

상기 레코드들의 세트를 포함하는 상기 제2 응답을 상기 질의에 대한 응답으로서 전송하게 하도록 더 실행가능한, 장치.

청구항 6

제1항에 있어서,

상기 명령어들은 상기 프로세서로 하여금, 상기 제1 가상 클론이 상기 제2 가상 클론의 자손이라는 표시를 상기 메모리에 저장하게 하도록 더 실행가능한, 장치.

청구항 7

제1항에 있어서,

상기 명령어들은 상기 프로세서로 하여금, 상기 데이터베이스 테이블의 제3 가상 클론을 생성하게 하도록 더 실행가능하고, 상기 제3 가상 클론은 상기 제1 가상 클론의 조상도 자손도 아니며, 상기 복수의 레코드들은 상기 데이터 부분의 값 및 상기 제3 가상 클론과 연관된 상기 버전 식별자를 포함하는 복합 키를 갖는 레코드를 포함하는, 장치.

청구항 8

장치로서,

프로세서; 및

상기 프로세서와 전자 통신하는 메모리—상기 메모리는 데이터베이스 테이블을 저장함—를 포함하고,

상기 프로세서는 상기 프로세서로 하여금:

상기 데이터베이스 테이블의 복수의 가상 클론들의 표현들을 포함하는 계층적 트리를 생성하고—상기 복수의 가상 클론들은 제1 가상 클론, 상기 제1 가상 클론의 자식 가상 클론인 제2 가상 클론, 및 상기 제2 가상 클론의 후속자 가상 클론인 제3 가상 클론을 포함함—;

상기 제2 가상 클론을 삭제하라는 명령어에 응답하여, 상기 계층적 트리 내의 그리고 상기 제3 가상 클론과 연관된 후속자 관계를 수정하고;

상기 제2 가상 클론을 삭제하라는 상기 명령어에 응답하여, 상기 제2 가상 클론을 삭제하고;

상기 제2 가상 클론을 삭제하라는 상기 명령어에 응답하여, 상기 제1 가상 클론이 상기 제2 가상 클론의 삭제 후에 자식 가상 클론을 갖지 않을 때, 상기 복수의 가상 클론들로부터 상기 제1 가상 클론을 삭제하게 하도록 실행가능한 명령어들을 포함하는, 장치.

청구항 9

제8항에 있어서,

상기 복수의 가상 클론들은 상기 제1 가상 클론의 후속자 가상 클론인 제4 가상 클론을 더 포함하고, 상기 명령어들은 상기 프로세서로 하여금, 상기 제2 가상 클론을 삭제하라는 명령어, 및 상기 제4 가상 클론이 상기 제2 가상 클론의 삭제 후에 상기 제1 가상 클론의 유일한 자손이라는 결정에 응답하여, 상기 제4 가상 클론과 연관된 후속자 관계를 수정하게 하도록 더 실행가능한, 장치.

청구항 10

제8항에 있어서,

상기 명령어들은 상기 프로세서로 하여금:

상기 복수의 가상 클론들로부터의, 자식 가상 클론을 갖지 않는 제4 가상 클론을 자동으로 식별하고;

상기 자동 식별에 기초하여 상기 제4 가상 클론을 삭제하게 하도록 더 실행가능한, 장치.

청구항 11

제8항에 있어서,

상기 명령어들은 상기 프로세서로 하여금:

상기 복수의 가상 클론들로부터의, 자식 가상 클론을 갖지 않는 제4 가상 클론을 자동으로 식별하고;

상기 자동으로 식별하는 것에 응답하여:

상기 복수의 가상 클론들로부터의 제5 가상 클론과 연관된 후속자 관계를 수정하고;

상기 제4 가상 클론을 삭제하게 하도록 더 실행가능한, 장치.

청구항 12

제8항에 있어서,

상기 명령어들은 상기 프로세서로 하여금, 상기 제3 가상 클론과 연관된 상기 후속자 관계를 수정하는 것에 응답하여, 상기 복수의 가상 클론들로부터의 적어도 하나의 추가 가상 클론의 삭제를 트리거하게 하도록 더 실행가능한, 장치.

청구항 13

제8항에 있어서,

상기 후속자 관계는 제1 후속자 관계이고, 상기 메모리는 상기 제1 후속자 관계를 포함하는 복수의 후속자 관계를 더 저장하는, 장치.

청구항 14

제8항에 있어서,

상기 데이터베이스 테이블은 복수의 레코드들을 포함하고, 상기 복수의 레코드들로부터의 각각의 레코드는 상기 복수의 가상 클론들로부터의 가상 클론과 연관된 버전 식별자를 포함하는 복합 키를 포함하는, 장치.

청구항 15

장치로서,

프로세서; 및

상기 프로세서와 전자 통신하는 메모리—상기 메모리는 데이터베이스 테이블을 저장함—를 포함하고,

상기 프로세서는 상기 프로세서로 하여금:

상기 데이터베이스 테이블의 복수의 가상 클론들의 표현들을 포함하는 계층적 트리를 생성하고;

상기 복수의 가상 클론들로부터 제1 가상 클론을 삭제하라는 명령어에 응답하여:

제2 가상 클론과 연관된 후속자 관계를 수정하고—상기 제2 가상 클론은 상기 제1 가상 클론의 후속자 가상 클론임—;

상기 제1 가상 클론을 삭제하고;

제4 가상 클론과 연관된 후속자 관계를 수정하고—상기 제4 가상 클론은 제3 가상 클론의 후속자 가상 클론이고, 상기 제3 가상 클론은 상기 제1 가상 클론의 조상 가상 클론임—;

상기 제3 가상 클론을 삭제하게 하도록 실행가능한 명령어들을 포함하는, 장치.

청구항 16

제15항에 있어서,

상기 명령어들은 상기 프로세서로 하여금:

상기 제2 가상 클론이 유일한 자손 가상 클론이고 상기 제4 가상 클론의 후속자 가상 클론인 것을 검출하고;

상기 제2 가상 클론이 유일한 자손 가상 클론이고 상기 제4 가상 클론의 상기 후속자 가상 클론인 것을 검출하는 것에 응답하여 상기 제2 가상 클론을 삭제하게 하도록 더 실행가능한, 장치.

청구항 17

제15항에 있어서,

상기 데이터베이스 테이블은 복수의 레코드들을 포함하고, 상기 복수의 레코드들로부터의 각각의 레코드는 상기 복수의 가상 클론들로부터의 가상 클론과 연관된 버전 식별자를 포함하는 복합 키를 포함하는, 장치.

청구항 18

제15항에 있어서,

상기 명령어들은 상기 프로세서로 하여금, 상기 제1 가상 클론을 삭제하라는 명령어에 응답하여, 상기 복수의 가상 클론들로부터의 적어도 하나의 추가 가상 클론의 삭제를 트리거하게 하도록 더 실행가능한, 장치.

청구항 19

제15항에 있어서,

상기 명령어들은 상기 프로세서로 하여금:

상기 복수의 가상 클론들로부터의, 자손 가상 클론을 갖지 않는 제5 가상 클론을 자동으로 식별하고,

상기 자동으로 식별하는 것에 응답하여:

제6 가상 클론과 연관된 후속자 관계를 수정하고;

상기 제5 가상 클론을 삭제하게 하도록 더 실행가능한, 장치.

청구항 20

제15항에 있어서,

상기 메모리는 상기 제2 가상 클론과 연관된 상기 후속자 관계 및 상기 제4 가상 클론과 연관된 상기 후속자 관계를 포함하는 복수의 후속자 관계들을 더 저장하는, 장치.

발명의 설명

기술 분야

[0001] 관련 출원들에 대한 상호 참조

[0002] 본 출원은 2017년 11월 1일자로 출원된, 발명의 명칭이 "Methods and Apparatus for Efficiently Implementing a Distributed Database within a Network"인 미국 가출원 제62/580,056호의 우선권 및 이익을 주장하며, 그러한 가출원의 개시내용 전체가 본 명세서에 참조로 포함된다.

배경 기술

[0003] 본 명세서에 설명된 실시예들은 일반적으로 증가된 효율 데이터베이스 시스템(increased-efficiency database system)을 구현하기 위한 방법들 및 장치들에 관한 것이다.

발명의 내용

[0004] 개요

[0005] 일부 실시예들에서, 고속 카피가능 데이터베이스 장치(fast-copyable database apparatus)는 프로세서, 및 프로세서와 전자 통신하는 메모리를 포함한다. 메모리는 각각 버전 식별자(version identifier)를 포함하는 복합 키(compound key)를 갖는 다수의 레코드들(records)을 갖는 데이터베이스 테이블(database table)을 저장한다. 메모리는 또한, 프로세서로 하여금, 데이터베이스 테이블의 제1 가상 클론(first virtual clone) 및 데이터베이스 테이블의 제2 가상 클론을 생성하고, 제1 가상 클론에서 질의(query)를 수신하고, 질의가 제1 가상 클론의 버전 식별자를 포함하도록 질의를 수정하게 하도록 실행가능한 명령어들을 저장한다. 프로세서는 수정된 질의를 이용하여 데이터베이스 테이블에 질의하여, 제1 가상 클론의 버전 식별자의 조상 경로(ancestral path)와 연관된 버전 식별자를 갖는 레코드들의 세트로부터의 각각의 레코드에 부분적으로 기초하여 응답을 생성하고, 응답을 전송할 수 있다. 각각의 복합 키는 버전 식별자 및 데이터 부분을 포함한다.

도면의 간단한 설명

[0006] 도 1은 실시예에 따른, 고속 카피가능 데이터베이스를 구현하기 위한 시스템을 도시하는 블록도이다.

도 2a는 실시예에 따른, 질의가능 계층적 트리(queryable hierarchical tree) 및 연관된 데이터베이스 테이블의 도면이다.

도 2b는 실시예에 따른, 충돌 제약(conflict constraint)이 시행되는 질의가능 계층적 트리 및 연관된 데이터베이스 테이블의 도면이다.

도 3a는 실시예에 따른, 계층적 트리의 가상 클론들을 생성하고 그와 상호작용하기 위한 방법을 도시하는 흐름도이다.

도 3b는 도 3a의 계층적 트리를 통한 데이터베이스 테이블의 시간적 진화 및 데이터베이스 테이블과의 상호작용을 도시하는 일련의 도면들을 포함한다.

도 4a 내지 도 4f는 실시예에 따른, 가상 클론들의 표현들, 및 연관된 데이터베이스 테이블을 포함하는 계층적 트리의 시간적 진화를 도시하는 도면들을 포함한다.

도 4g는 실시예에 따른, 질의 프로세스를 도시하는 도면을 포함한다.

도 5는 실시예에 따른, 증분적 가비지 수집 프로세스(incremental garbage collection process)를 도시하는 일련의 도면들을 포함한다.

도 6은 실시예에 따른, 고속 카피 데이터베이스를 위한 프로세서에 의해 구현된 가비지 수집(또는 캐스케이딩 삭제(cascading deletion)) 방법에 대한 프로세스 흐름도이다.

도 7은 실시예에 따른, 고속 카피 데이터베이스를 위한 프로세서에 의해 구현된 가비지 수집(또는 캐스케이딩

삭제) 방법에 대한 프로세스 흐름도이다.

도 8a 내지 도 8d는 실시예에 따른, 고속 카피가능 데이터베이스 시스템의 구현을 도시하는 도면들이다.

도 9는 실시예에 따른, 고속 카피 동작들을 도시하는 일련의 도면들을 포함한다.

발명을 실시하기 위한 구체적인 내용

[0007]

본 개시내용의 고효율 데이터베이스 시스템들은, 각각이 비밀시적인, 프로세서 판독가능 메모리에 저장되는 (본 명세서에서, "데이터베이스 테이블"을 포함하고/하거나 그것과 상호교환가능하게 이용되는) 하나 이상의 물리적 관계형 데이터베이스(physical relational database), 및 관계형 데이터베이스의 데이터가, 미리 결정된 조건들 하에서 및/또는 미리 정의된 기간들 내에, 액세스 및/또는 수정될 수 있게 하는 다수의 가상 클론들을 각각 포함하는 하나 이상의 연관된 동적 계층적 트리를 포함한다. 본 명세서에 개시된 데이터베이스 구현 및 관리의 방법들은 알려진 데이터베이스 복제(replication) 및 상호작용의 방법들보다 상당히 더 빠르고 더욱 효율적이다. 본 명세서에 설명된 관계형 데이터베이스들은, 본 개시내용에 따라 구현될 때, 많은 상이한 "카피들(copies)"이 새로운 가상 클론의 인스턴스화(instantiation)를 통해 신속하게 생성 및/또는 정의될 수 있기 때문에 "고속 카피가능"이라고 지칭될 수 있으며, 여기서 각각의 상이한 "카피"는 "가상" 카피-상이한 연관된 가상 클론의 관점으로부터 인지되고 연관된 가상 클론으로부터 액세스가능한 콘텐츠임-를 지칭한다. 달리 말하면, 관계형 데이터베이스는 관계형 데이터베이스를 고속 카피가능 데이터베이스로 변환하는 클래스에서 "랩핑(wrapped)"되는 것으로 언급될 수 있다.

[0008]

일부 구현들에서, 본 개시내용의 고효율 데이터베이스 시스템들은, 다수의 연관된 지리적으로 분산된 장소들에서, 선택적으로 각각 상이한 연관된 관리자를 갖지만, 관계형 데이터베이스의 중앙집중형 "마스터(master)" 또는 "리더(leader)" 카피가 없는, 연관된 동적 계층적 트리들을 갖는 다수의 물리적 관계형 데이터베이스(즉, 관계형 데이터베이스의 "인스턴스들(instances)")가 존재한다는 점에서, "리더리스(leaderless)"로서 기술될 수 있다. 즉, 상이한 변경들이 다수의 물리적 관계형 데이터베이스들 각각에서, 상이한 순서들로 및/또는 상이한 시간들에 적용될 수 있다. 분산 데이터베이스 시스템(distributed database system)에서 리더를 갖지 않는 것은, 분산 데이터베이스 시스템의 보안성을 증가시킬 수 있다. 예를 들어, 리더가 있으면, 공격 및/또는 장애의 단일점(single point)이 존재하게 된다. 악성 소프트웨어가 리더를 감염시키고/시키거나 리더의 데이터베이스 테이블에서의 값이 악의적으로 변경되어 그것이 부정확한 값을 갖는다면, 감염 및/또는 부정확한 값이 데이터베이스 인스턴스들의 네트워크 전체에 걸쳐 전파되거나, 다른 분산 데이터베이스 인스턴스에 전송될 수 있다. 대조적으로, 본 개시내용의 리더리스 시스템들에서는, 공격 및/또는 장애의 단일점이 존재하지 않는다. 본 명세서에 설명된 리더리스 분산 데이터베이스 시스템들은 또한, 디바이스들 사이에서 송신되는 데이터의 양을 감소시키면서, 하나 이상의 합의 알고리즘(consensus algorithm)의 수렴 속도를 증가시킬 수 있다. 본 명세서에 설명된 분산 데이터베이스 시스템들은 또한 데이터베이스 테이블의 콘텐츠에 대한 이질적인 증분적 변경들(disparate incremental changes)을 추적하는데 이용될 수 있는 가상 클론들의 생성을 통해 "고속 카피가능"하다. 본 명세서에서 이용된 바와 같이, "고속 카피"는 "가상 클론"이라는 문구와 동일한 의미를 갖는다. 그와 같이, 일부 알려진 분산 데이터베이스와는 달리, 특히, 관리되고 있는 데이터의 코퍼스(corpus)가 클 때, 리소스 집약적(resource-intensive)이고 시간 소모적(time-consuming)일 수 있는 복제(replication) 및 중복(duplication)을 이용하지 않으면서 업데이트들이 달성될 수 있다.

[0009]

일부 실시예들에서, 계층적 트리의 가상 클론들의 세트로부터의 각각의 가상 클론은, 관계형 데이터베이스의 데이터가 그 가상 클론과 (예를 들면, 질의들을 통해) 상호작용하는 사용자에게 의해 액세스될 수 있어서, 비록 그런 것은 아니지만(즉, 저장 및 질의는 관계형 데이터베이스/테이블 상에서 발생되지만), 사용자에게는 데이터가 그 가상 클론에서 저장되고/되거나 국부적으로 검색되는 것으로 보여지도록 한다는 점에서, "가상 데이터베이스"로서 기능할 수 있다. 관계형 데이터베이스의 데이터 테이블에 대한 수정들은 가상 클론들의 세트로부터의 다수의 가상 클론들에 적용될 수 있고, "복합 키"를 이용하여(예를 들어, 데이터베이스 테이블 및 데이터베이스 테이블의 레거시 키 필드(legacy key field)에 추가된 속성(attribute)을 이용하여) 관계형 데이터베이스 내의 그 가상 클론과 연관될 수 있다. 복합 키는 적어도 2개의 값, 즉, 적어도 하나의 가상 클론과 연관된 버전 식별자("버전 ID"), 및 그 데이터베이스 테이블의 레거시 키 필드(본 명세서에서 "가상 기본 키 필드(virtual primary key field)"라고도 지칭됨)와 같은, 참조되는 데이터베이스 테이블의 특성(예를 들어, 행(row)(즉, "레코드들(records)" 또는 "튜플들(tuples)") 식별자 또는 이름, 열(column)(즉, "속성") 식별자 또는 이름, 데이터 라벨, 데이터 값 등)을 지칭하는 필드를 포함할 수 있다. 예시적인 속성들은, 제한적인 것은 아니지만, 가상 기본 키 이름, 값(예를 들어, 잔고(balance), 차변(debit), 대변(credit), 이체 금액(transfer

amount)), 통화(currency), 설명(description), 날짜(date), 수량(quantity), 집계(tally), 버전, 지속기간(duration), 주문(order) 등을 포함한다. 복합 키들의 부분들은 데이터베이스 테이블 레코드들의 일부로서 저장될 수 있어서, 하나의 가상 클론에 의해 행해진 편집들은 임의의 다른 가상 클론의 관점에서 데이터베이스 테이블의 데이터에 영향을 미치지 않는다. 일부 실시예들에서, 주어진 복합 키의 다수의 인스턴스들은 임의의 주어진 시간에 데이터베이스 테이블 내에 공존하는 것이 허용되지 않는다(즉, 데이터베이스 테이블 내에 주어진 복합 키의 하나의 인스턴스만이 존재할 수 있다). 특정 가상 클론에 의해 행해진 관계형 데이터베이스의 데이터 테이블에 대한 수정들은 그 가상 클론의 버전 식별자를 통해 그 가상 클론과 연관되어, 그 가상 클론을 통해 행해진 후속 질의들 또는 수정들이 그 가상 클론/버전 ID를 통해 데이터베이스 테이블에 대해 행해진 누적 상호작용들(cumulative interactions)에 의해 정의된 것으로서 관계형 데이터베이스의 가장 최근의 형태에 적용될 수 있다. 그러나, 특정 가상 클론에 의해 행해진 데이터베이스 테이블에 대한 수정들은, 다른 가상 클론들의 관점에서 명백하지 않거나 액세스가능하지 않다. 이러한 분산 방식으로 변경들을 구현하는 것은, 가상 클론들의 연관된 버전 ID들에 기초하여 그러한 변경들의 연관된 수정 가상 클론들에 대한 변경들의 속성에 의해, (예를 들어, 가상 클론들을 호스팅하는 서버 노드들의 세트의 각각에서) 각각의 관계형 데이터베이스의 데이터의 다수의 카피들을 복제, 분배 및 국부적으로 저장할 필요성을 제거한다. 본 명세서에 개시된 방법들은 그것과의 네트워크 통신에서 각각의 머신에 의해 행해진 각각의 변경에 응답하여 관계형 데이터베이스의 전체 콘텐츠를 많은 머신들로 다수화 송신할 필요성을 제거할 수 있다.

[0010] 고속 카피가능 데이터베이스 시스템

[0011] 도 1은 일부 실시예들에 따른, 고속 카피가능 데이터베이스 시스템(100)의 컴포넌트들을 도시하는 시스템도이다. 도 1에 도시된 바와 같이, 시스템(100)은 네트워크(105)와 무선 또는 유선 통신하는 제1 컴퓨팅 디바이스(110), 및 네트워크(105)와 또한 무선 또는 유선 통신하는 하나 이상의 추가 컴퓨팅 디바이스(120)를 포함한다. 제1 컴퓨팅 디바이스(110)는 온보드 메모리(onboard memory)(112) 및 디스플레이(113)와 동작가능하게 통신하는 프로세서(111)를 포함한다. 메모리(112)는 하나 이상의 데이터베이스 테이블(114)(즉, 관계형 데이터베이스의 인스턴스(들)), 고속 카피어(fast-copier)(115), 및 (예를 들어, 프로세서로 하여금 합의 알고리즘을 구현하게 하는) 합의 명령어들(116)을 저장한다. 각각의 데이터베이스 테이블(114)은 아래에 더 상세히 설명되는 하나 이상의 복합 키(114A)를 포함한다. 하나 이상의 추가 컴퓨팅 디바이스(120)로부터의 각각의 컴퓨팅 디바이스는 온보드 메모리(122)와 동작가능하게 통신하는 프로세서(121), 및 선택적으로 하나 이상의 인터페이스(예를 들어, 디스플레이, 그래픽 사용자 인터페이스(GUI), 무선 송수신기 등)(도시되지 않음)를 포함한다. 메모리(122)는 하나 이상의 데이터베이스 테이블(124)(즉, 관계형 데이터베이스의 인스턴스(들)), 고속 카피어(125), 및 (예를 들어, 프로세서로 하여금 합의 알고리즘을 구현하게 하는) 합의 명령어들(126)을 포함한다.

[0012] 프로세서들(111 및 121) 각각은, 범용 프로세서, 중앙 처리 유닛(central processing unit)(CPU), 마이크로프로세서, 디지털 신호 프로세서(digital signal processor)(DSP), 제어기, 마이크로제어기, 상태 머신 등 중에서의 하나 이상을 포함할 수 있다. 일부 환경들 하에서, 본 명세서에서 이용된 "프로세서"는 ASIC(application specific integrated circuit), PLD(programmable logic device), FPGA(field programmable gate array) 등을 지칭할 수 있다. "프로세서"라는 용어는 처리 디바이스들의 조합, 예를 들어, DSP와 마이크로프로세서의 조합, 다수의 마이크로프로세서들, DSP 코어와 결합된 하나 이상의 마이크로프로세서, 또는 임의의 다른 그러한 구성을 지칭할 수 있다.

[0013] 메모리들(112 및 122) 각각은 데이터 및 코드(예를 들어, 명령어들)와 같은 전자 정보를 저장할 수 있는 임의의 전자 컴포넌트를 포함할 수 있다. 용어 메모리는 RAM(random access memory), ROM(read-only memory), NVRAM(non-volatile random access memory), PROM(programmable read-only memory), EPROM(erasable programmable read only memory), EEPROM(electrically erasable PROM), 플래시 메모리, 자기 또는 광학 데이터 스토리지, 레지스터 등과 같은 다양한 타입의 비일시적, 프로세서 판독가능 매체들을 지칭할 수 있다. 메모리는, 프로세서가 메모리로부터 정보를 판독하고/하거나 메모리에 정보를 기입할 수 있는 경우, 프로세서와 동작가능하게 또는 전자 통신한다고 언급된다. 프로세서와 통합된 메모리는 프로세서와 전자 통신하고 있다. 비일시적 컴퓨터 판독가능 매체들의 예들은, 제한적인 것은 아니지만, 하드 디스크들, 플로피 디스크들, 및 자기 테이프와 같은 자기 저장 매체들; 콤팩트 디스크/디지털 비디오 디스크(CD/DVD)들, 콤팩트 디스크 판독 전용 메모리(CD-ROM)들, 및 홀로그래픽 디바이스들과 같은 광학 저장 매체들; 광학 디스크들과 같은 자기 광학 저장 매체들; 반송파 신호 프로세서들; 및 ASIC들, PLD들, ROM 및 RAM 디바이스들과 같은, 프로그램 코드를 저장 및 실행하도록 특별하게 구성되는 하드웨어 디바이스들을 포함한다. 본 명세서에서 설명된 다른 실시예들은, 예를

들어, 본 명세서에서 논의된 명령어들 및/또는 컴퓨터 코드를 포함할 수 있는 컴퓨터 프로그램 제품에 관한 것이다.

[0014] 본 개시내용의 고속 카피가능 데이터베이스 시스템들은, 도 1을 참조하여 이하에서 도시되고 설명되는 바와 같이, 일부 실시예들에서, 하드웨어로서 또는 소프트웨어, 예를 들어, "고속 카피어"로서, 또는 비일시적 메모리에 저장되고 메모리와 동작가능하게 통신하는 프로세서에 의해 실행가능한 명령어들을 포함하는 다른 형태의 소프트웨어로서 구현될 수 있다. 메모리는 또한 관계형 데이터베이스를 저장할 수 있다(및/또는 프로세서가 관계형 데이터베이스에 액세스할 수 있다). 고속 카피어는 그것이 연관되는 관계형 데이터베이스를 "랩(wrap)"하는 "랩퍼 클래스(wrapper class)"("랩퍼 패턴" 또는 "랩퍼 함수"라고도 지칭됨)일 수 있다. 예를 들어, 소프트웨어가 실행중이고 SQL(Structured Query Language) 질의가 프로세서에서 수신되면, 프로세서는 랩퍼 클래스에서의 방법을 호출함으로써 SQL 질의를 관계형 데이터베이스에 전송/패스할 수 있으며, 랩퍼 클래스에서의 방법은 질의를 관계형 데이터베이스에 전송하기 전에 질의를 수정할 수 있다. 그 다음, SQL 질의에 응답하여 관계형 데이터베이스에 의해 반환된 결과들은 랩퍼 클래스에서의 방법에 의해 반환/전송될 수 있다. 고속 카피 소프트웨어는 다음 중 하나 이상을 임의의 조합으로 수행하도록 구성될 수 있다: 관계형 데이터베이스에 대한 호출들을 인터셉트하는 것, 관계형 데이터베이스로부터의 호출들을 인터셉트하는 것, 관계형 데이터베이스에 필드(예를 들면, 아래에 논의되는 바와 같이 "버전")를 추가하는 것, 관계형 데이터베이스의 필드를 수정하는 것, 관계형 데이터베이스의 필드를 삭제하는 것, 관계형 데이터베이스의 구조를 수정하는 것(예를 들어, 레코드/행 또는 속성/열을 추가 또는 제거하는 것), 또는 도 2 내지 도 7을 참조하여 이하에서 도시되고 설명되는 임의의 기능.

[0015] 일부 실시예들에서, 고속 카피가능 데이터베이스 시스템은 단일 머신(또는 "노드") 또는 다수의 머신들, 또는 "노드들"을 포함하고, 이들 각각은 관계형 데이터베이스의 인스턴스 및 고속 카피 소프트웨어를 호스팅하는 비일시적 메모리에 동작가능하게 결합된 프로세서를 포함한다. 초기에(즉, 임의의 변경들이 적용되기 전에), 관계형 데이터베이스의 각각의 인스턴스는 동일하다. 노드들은 서로 지리적으로 분리되고/되거나 상이한 관리자 개인들 및/또는 엔티티들과 연관될 수 있다. 각각의 노드는 관계형 데이터베이스가 수정 및/또는 질의될 수 있게 하는 하나 이상의 가상 클론(본 명세서에서 "객체들" 또는 "가상 객체들"이라고도 지칭됨)을 포함하는 연관된 계층적 트리 및/또는 계층적 트리의 후속 부분들(또는 계층적 트리의 표현)을 생성 및/또는 저장할 수 있다. 계층적 트리를 생성하는 것은 기본(underlying) 계층적 트리가 확장 및/또는 수정됨에 따라 다수회 수행될 수 있다. 예를 들어, 각각의 노드는 적어도 2개의 가상 클론들, 즉, 제1 가상 클론 및 제2 가상 클론을 생성할 수 있다. 제1 가상 클론은, 예를 들어, 활동 데이터(activity data)가 해당 노드에서(예를 들어, 다른 노드로부터) 수신됨에 따라 관계형 데이터베이스의 필드들에 대해 변경들을 적용하는데 이용될 수 있다. 이러한 변경들은, 예를 들어, 활동 데이터를 수신하는 것에 응답하여 및/또는 활동 데이터에 기초하여 자동으로 및/또는 관리자에 의해 적용될 수 있다. 활동 데이터는 수신 노드가 당사자인 노드들의 네트워크 내에서 발생하거나 또는 발생하였던 것으로 간주되는 하나 이상의 이벤트 또는 트랜잭션과 관련된 데이터를 포함할 수 있다. 제1 가상 클론에 의해 적용된 변경들은, 하나 이상의 트랜잭션 또는 이벤트와 연관된 수신된 상태 그대로의 활동 데이터의 "실행 집계(running tally)"를 집합적으로 나타내는 것으로서 보여질 수 있다.

[0016] 일부 구현들에서, 활동 데이터는, 노드들 중 하나 이상에 의해 개시되고/되거나 고속 카피 소프트웨어에 의해 자동으로 구현되는 "동기화" 이벤트의 결과로서, 주어진 노드에서 수신된다(예를 들어, 노드들의 네트워크 내의 검출된 변경에 의해 및/또는 미리 정의된 시간 스케줄에 따라 트리거된다). 동기화 동안, 노드들의 네트워크로부터의 적어도 2개의 노드들은 그들의 연관된 데이터베이스 테이블들로부터의 데이터를 교환한다. 데이터는, 예를 들어, 활동 데이터 또는 그 일부일 수 있다. 적어도 2개의 노드들 사이에서/중에서 교환된 데이터는 노드 네트워크 내의 총 활동(aggregate activity)의 서브세트 또는 부분적 뷰(view)를 나타낼 수 있다. 일부 그러한 경우들에서, 제1 가상 클론을 통해 노드의 데이터베이스 테이블에 적용된 변경들은 합의 순서화(consensus ordering) 및/또는 유효성(validity)이 도달되지 않았고/않았거나, 중간(intervening) 변경들이 데이터베이스 테이블로부터 누락(missing)되는지에 관한 결정이 아직 행해지지 않은 "잠정적 변경들(tentative changes)"이다(즉, 합의는 그러한 변경들에 대해 아직 도달/충족되지 않음). 제1 가상 클론을 통해 노드에서 수정되는 데이터베이스 테이블 내의 레코드들은 제1 가상 클론과 연관된 제1 버전 ID를 포함하는 복합 키를 포함할 수 있다.

[0017] 일부 구현들에서, 동기화 프로세스는 (1) (예를 들어, 하나 이상의 트랜잭션을 포함하는) 하나 이상의 이벤트 또는 동기화 동안 수신되는 트랜잭션들에 기초하여 (예를 들어, 노드에 의해 정의되는 데이터베이스 상태 변수에 의해 캡처된 것으로서의) 데이터베이스 테이블의 "현재 상태"를 반복적으로 업데이트하는 것, 및 (2) 이벤트들 또는 트랜잭션들의 시퀀싱에 관한 새로운 정보가 수신될 때마다, 예를 들어, 이전 상태의 카피로 돌아가는 것에 의해, 그 현재 상태를 반복적으로 재형성(rebuilding)(예를 들어, 이벤트들의 순서를 재배열), 및 이벤트

들 또는 트랜잭션들을 새로운 순서로 처리함으로써 현재 상태를 재계산하는 것을 포함한다. 따라서, 예를 들어, 각각의 컴퓨팅 디바이스는 상태의 2개의 버전을 유지할 수 있는데, 하나는, (예를 들어, 위에서 논의된 제1 가상 클론에 의해 구현된 것으로서의) 새로운 이벤트들 및 트랜잭션들이 수신됨에 따라 업데이트되는 것이고, 하나는, (예를 들어, 위에서 논의된 제2 가상 클론에 의해 구현된 것으로서의) 합의가 달성된 후에만 업데이트되는 것이다. (예를 들어, 미리 결정된 시간 기간 이후, 미리 결정된 수의 이벤트들이 정의되고/되거나 수신된 이후 등의) 어떤 시점에서, 새로운 이벤트들 및 트랜잭션들이 수신됨에 따라 업데이트되는 상태의 버전이 계층적 트리로부터 폐기 또는 삭제될 수 있고, 합의가 달성된 후에 업데이트되는 상태의 새로운 카피가, 새로운 이벤트들 및 트랜잭션들이 수신됨에 따라 업데이트되는 상태의 새로운 버전으로서 (예를 들어, 위에서 논의된 제3 가상 클론에 의해 구현된 것으로서) 생성될 수 있다. 전술한 방안은 두 상태들의 동기화를 보장할 수 있다. 일부 경우들에서, (상태의 상이한 카피들을 포함하는) 상태는, 예를 들어, "고속 카피가능" 어레이리스트(arrayList)(본 명세서에서 고속 클론 어레이리스트, 고속 클론 어레이 또는 고속 카피가능 어레이라고도 지칭됨), "고속 카피가능" 해시 테이블, "고속 카피가능" 관계형 데이터베이스, 또는 고속 카피가능 파일 시스템에서 파일의 각각의 N바이트(예를 들어, 4096바이트 또는 1024바이트) 부분에 대해 호스트 오퍼레이팅 시스템 상에 생성 및/또는 정의되는 분리된 파일을 선택적으로 갖는 "고속 카피가능" 파일 시스템(본 명세서에서 고속 클론 파일 시스템 및/또는 고속 카피가능 파일시스템이라고도 지칭됨)과 같은 임의의 적절한 데이터 구조에 저장될 수 있다.

[0018] 본 명세서에서 이용된 바와 같이, "합의"는, 합의 알고리즘을 이용하여 도달된, 합의 알고리즘에 의해 고려되는 하나 이상의 변경이 정확하게 순서화되고, 중복들이 아니고, 어떠한 중간 변경들도 누락되지 않고/않거나 합법적(즉, 악의적이거나 허위가 아님)이라는 결정을 지칭한다. 합의는, 예를 들어, 모든 또는 적어도 미리 정의된 임계 수치의 노드들이 동의하거나, 또는 적어도 미리 정의된 임계 레벨의 확실성으로 결론지어질 때, 변경들이 정확하게 순서화되고, 중복들이 아니고, 어떠한 중간 변경들도 누락되지 않고/않거나 유효한 경우에 도달된다고 언급될 수 있다. 예를 들어, 합의에 도달한 변경들의 시퀀스는 이벤트들 및/또는 트랜잭션들의 시퀀스의 "참(true)" 또는 정확한 표현을 반영하는 것으로 언급될 수 있다. 일부 구현들에서, 합의는 미리 결정된 시간 프레임 내에서, 합의 알고리즘의 미리 결정된 횟수의 반복들 후에, 및/또는 미리 결정된 스케줄에 따라, "라운드들(rounds)"에 도달된다.

[0019] 제1 가상 클론을 통해 적용된 변경 또는 변경들의 세트에 대한 합의에 도달한 후에, 그러한 변경 또는 변경들의 세트는 제2 가상 클론을 통해 데이터베이스 테이블에 적용될 수 있고, 데이터베이스 테이블의 각각의 영향받은 레코드는 제2 가상 클론과 연관된 제2 버전 ID를 포함하는 복합 키를 대응적으로 포함할 수 있다. 변경 또는 변경들의 세트가 제2 가상 클론을 통해 데이터베이스 테이블에 적용된 후에, 및 선택적으로 자동으로 및/또는 그 변경들이 적용되는 것에 응답하여, 제1 가상 클론이 삭제될 수 있고, 제2 가상 클론의 자손들(descendants)인 적어도 2개의 추가 가상 클론들(예를 들어, 제3 가상 클론 및 제4 가상 클론)이 생성될 수 있다. 그 다음, 고속 카피 프로세스는, 제3 가상 클론을 통해 행해지는 데이터베이스 테이블에 대한 잠정적 변경들, 및 제4 가상 클론을 통해 행해지는 데이터베이스 테이블에 대한 합의후(post-consensus) 변경들을, 전술한 것과 유사한 방식으로 계속할 수 있다.

[0020] 일부 구현들에서, 합의 알고리즘 또는 방법은 플랫폼(즉, 도 1의 컴퓨팅 디바이스(110)의 메모리(112)에 저장되고, 프로세서(111)에 의해 실행되는 합의 명령어들(116)과 같은, 컴퓨팅 디바이스 및/또는 시스템의 프로세서 상에서 실행되는 소프트웨어 프로그램)에 의해 구현될 수 있다. 플랫폼은 트랜잭션들을 수집(collect)하고, 유포(disseminate)하고, 순서화(order)하고, 합의 프로세스들과 연관된 다른 적절한 작업들을 수행할 수 있다.

[0021] 일부 구현들에서, 질의들은, 가상 클론 및/또는 그 연관 노드에 매달린(즉, 그로부터 직접 확장하고, 계층적 트리에서 하향으로 이동하는) 하나 이상의 자손 노드가 생성되는 경우, 하나 이상의 자손 노드가 계층적 트리로부터 삭제 또는 제거되는 경우에도 및 그후에, 가상 클론 및/또는 그 연관 노드를 통해 더 이상 행해질 수 없다.

[0022] 일부 구현들에서, 본 개시내용의 하나 이상의 데이터베이스 테이블(들)은 하나 이상의 가상 클론을 통해 질의될 수 있다. 도 2a 및 도 2b는 실시예에 따른, 질의가능 계층적 트리(본 명세서에서 "버전 트리"라고도 지칭됨)의 도면들을 도시한다. 일부 구현들에서, 계층적 트리(본 명세서에서 "버전 트리"라고도 지칭됨) 및/또는 계층적 트리의 표현이 비일시적 메모리에 유지되고/되거나 저장된다. 도 2a 및 도 2b에 도시된 바와 같이, 질의가능 계층적 트리는 관계형 데이터베이스 인스턴스(230) 및 그의 다수의 가상 클론들 V1, V2, V3 및 V4를 포함한다. 가상 클론들 V1, V2 및 V3의 직렬 제시는 가상 클론 V1이 먼저 생성되었고(선택적으로, V1을 통해 데이터베이스(230)에 대해 변경이 행해졌고), 가상 클론 V2(V1의 "자손" 가상 클론)가 두번째로 생성되었고(선택적으로, V2를 통해 데이터베이스(230)에 대해 변경들이 행해졌고), 가상 클론 V3(V2의 "자손" 가상 클론)이 세번째로 생성

되었음(그리고, 선택적으로, V3을 통해 데이터베이스(230)에 대해 변경들이 행해졌음)을 나타낸다. V3은, V3이 V3으로부터 확장하는/그에 매달리는 자식(child) 노드들을 갖지 않는다는 점에서 "리프 노드(leaf node)"이다. 일부 구현들에서, 자손 가상 클론이 생성되면, 데이터베이스(230)에 대한 변경들은 직접 혈통의(direct-lineage) 조상(ancestor) 가상 클론들을 통해 더 이상 행해질 수 없다는 점에 유의한다. 즉, 가상 클론 V3이 생성되면, 데이터베이스(230)에 대한 변경들은 가상 클론 V2 또는 가상 클론 V1을 통해 더 이상 적용될 수 없다.

[0023] (도 2a에 도시된) 실시예에서, V3에서 수신된 질의는 다음과 같이 순차적 방식으로 구현/수행될 수 있다: 데이터베이스(230)가 (또한 데이터베이스(230)에서의) 해당 레코드의 조상 버전과 동일한 가상 기본 키를 갖는 레코드를 포함하는 것으로 가정한다. 예를 들어, 도 2a의 데이터베이스 테이블은 속성들, 즉, 레코드 ID(RID)(가상 기본 키), 이름(Name), 랭크(Rank), 및 버전(Version)을 포함하고, (1) 복합 키 (1001, V3)을 갖는 데이터베이스(230)에서의 레코드, (2) 복합 키 (1001, V2)를 갖는 데이터베이스(230)에서의 레코드, 및 (3) 복합 키 (1001, V1)을 갖는 데이터베이스(230)에서의 레코드가 존재한다. 그러한 구현에서, 버전 트리가 도 2a에 도시된 바와 같고(여기서, V3은 V2의 자손 가상 클론이고, V2는 V1의 자손 가상 클론임), 질의가 가상 기본 키 RID=1001에 대해 가상 클론 V3을 통해 수신되는 경우, 프로세서는 먼저 (1001, V3)의 복합 키를 갖는 레코드를 검색할 것이다. (도 2a의 예에서 그러할) 그러한 복합 키를 갖는 레코드가 발견되면, 이 레코드는 반환되고 질의가 종료된다. 그러나, 버전 트리에 기초하여, 복합 키 (1001, V3)을 갖는 레코드가 데이터베이스(230)에서 발견되지 않았다면, 프로세서는 다음으로 복합 키 (1001, V2)에 대해 데이터베이스(230)에 질의할 것이다. 그러한 복합 키를 갖는 레코드가 데이터베이스(230)에서 발견되면, 그 레코드는 반환되고 질의가 종료된다. 그렇지 않으면, 프로세서는 복합 키 (1001, V1)에 대해 데이터베이스(230)에 질의한다. 따라서, 질의는, 결과들이 반환되지 않는 경우, 우선 버전 트리에서 최고 번호 버전(highest-numbered version)을 참조하고, 단계별/순차적 방식으로 버전 트리 위로 올라감으로써 수행될 수 있다. 전술한 질의 프로세스는 충돌 제약이 시행되지 않는 질의 프로세스로서 설명될 수 있다. 즉, 계층적 트리의 동일한 조상 체인/분기(ancestral chain/branch) 내에서 동일한 가상 기본 키 및 버전 ID를 갖는 복합 키들을 갖는 레코드들이 데이터베이스(230) 내에 공존할 수 있다.

[0024] 대신에, 도 2a에 도시된 계층적 트리가 주어지면, 가상 기본 키 1001에 대해 가상 클론 V4(V1의 자손 가상 클론)를 통해 질의가 수신되고, 프로세서는 먼저 (1001, V4)의 복합 키를 갖는 레코드를 검색할 것이다. (도 2a의 예에서 그러할) 그러한 복합 키를 갖는 레코드가 발견되면, 이 레코드는 반환되고 질의가 종료된다. 그러나, 버전 트리에 기초하여, 복합 키 (1001, V4)를 갖는 레코드가 데이터베이스(230)에서 발견되지 않으면, V1이 V4의 조상 가상 클론이기 때문에, 프로세서는 다음으로 복합 키 (1001, V1)에 대해 데이터베이스(230)에 질의할 것이다. 그러한 복합 키를 갖는 레코드가 데이터베이스(230)에서 발견되면, 그 레코드는 반환되고 질의가 종료된다. 그렇지 않으면, 프로세서는 "널(null)" 또는 "결과 없음(no results found)"의 응답을 반환한다.

[0025] 전술한 설명은 가상 기본 키에 대한 질의들을 언급하지만, 질의들은 매우 다양한 다른 방식으로 구조화될 수 있다. 예를 들어, 질의는 명시된 속성에 대한 값을 갖는 모든 레코드들(예를 들어, "Smith"의 속성 "이름"에 대한 값을 갖는 모든 레코드들), 2개 이상의 명시된 속성들에 대한 값들과 매칭되는 모든 레코드들(예를 들어, "Smith"의 속성 "이름" 및 "4"의 랭크에 대한 값을 갖는 모든 레코드들), 명시된 기본 가상 키 및 지정된 속성에 대한 값을 갖는 모든 레코드들(예를 들어, "1001"의 기본 가상 키 및 "Smith"의 속성 "이름"에 대한 값을 갖는 모든 레코드들) 등을 요청할 수 있다. 일부 구현들에서, 질의는 가상 기본 키에 대한 참조를 포함하지 않는다.

[0026] (도 2b에 도시된) 다른 실시예에서, V3에서 수신된 질의는 시행된 충돌 제약으로 구현/수행될 수 있다. 충돌 제약은 가상 클론 "X"에 대한 가상 기본 키 및 버전 ID를 포함하는 주어진 복합 키에 대해, X의 조상과 연관되는 동일한 가상 기본 키 및 버전 ID를 갖는 다른 복합 키가 데이터베이스(230)에 존재하지 않음을 나타내는 규칙 명시 및/또는 데이터베이스 아키텍처를 포함할 수 있다. 달리 말하면, 충돌 제약이 적용되고/시행되고/준수될 때, 임의의 주어진 가상 기본 키에 대해, 계층적 트리의 주어진 "분기" 상의 임의의 가상 클론과 연관된 버전 ID 및 가상 기본 키를 갖는 하나의 복합 키만이 주어진 시간에 데이터베이스 테이블 내에 저장될 수 있다. 따라서, 각각이 계층적 트리의 조상 경로를 따른 가상 클론과 연관된 가상 기본 키 및 버전 ID를 참조하는 다수의 복합 키들이 존재할 수 없다. 질의는 예로서 다음과 같이 시행되는 충돌 제약으로 구현/수행될 수 있다: 버전 트리 및 데이터베이스(230)가 도 2b에 도시된 바와 같은 것으로 가정한다. 가상 기본 키 RID=1001에 대해 가상 클론 V3을 통해 질의가 수신되면, 프로세서는 RID=1001 및 (버전=3 또는 버전=2 또는 버전=1)을 갖는 레코드들을 검색하여 찾는다. 질의는, 초기에 V3을 통해 수신된 바와 같이, 버전 ID 참조를 포함하지 않거나, 버전

ID V3만을 참조할 수 있다는 것에 유의한다. (예를 들어, 고속 카피 소프트웨어를 실행하는) 프로세서는 버전 ID들 V3, V2 및 V1 각각이 포함되도록 질의를 수정할 수 있다. 데이터베이스가 충돌 제약을 충족시키면, 결과적인 레코드들의 세트는 동일한 가상 기본 키를 갖는 2개의 엔트리를 포함하지 않을 것이며, 그 이유는, 결과들은 버전 트리에서의 단일 경로로부터 오는 버전들을 갖는 레코드들을 갖기 때문이고, 따라서 2개의 레코드가 동일한 가상 기본 키를 갖는 경우, 그들은 충돌 제약을 위반할 것이다. 결과들의 테이블이 생성될 수 있고, 결과들의 테이블이 검색의 결과로서 반환되기 전에 버전 속성(열)이 결과들의 테이블로부터 (예를 들어, 프로세서에 의해) 삭제될 수 있다.

[0027] 대신에, 도 2b에 도시된 계층적 트리가 주어지면, 가상 기본 키 1001에 대해 가상 클론 V4를 통해 질의가 수신되고, (예를 들어, 고속 카피어를 실행하는) 프로세서는 버전 ID들 V4 및 V1 각각이 포함되도록 질의를 수정할 수 있다. 그 다음, 프로세서는 RID=1001 및 (버전=V4 또는 버전=V1)을 갖는 레코드를 검색하여 찾는다. 프로세서가 수정된 질의와 매칭되는 어떠한 레코드들도 찾지 못하는 경우, 프로세서는 "널" 또는 "결과 없음"의 응답을 반환한다.

[0028] 위에서 언급된 바와 같이, 도 2b의 진술한 설명은 가상 기본 키에 대한 질의들을 언급하지만, 질의들은 매우 다양한 다른 방식으로 구조화될 수 있다. 예를 들어, 질의는 명시된 속성에 대한 값을 갖는 모든 레코드들(예를 들어, "Smith"의 속성 "이름"에 대한 값을 갖는 모든 레코드들), 2개 이상의 명시된 속성에 대한 값들과 매칭되는 모든 레코드들(예를 들어, "Smith"의 속성 "이름" 및 "4"의 랭크에 대한 값을 갖는 모든 레코드들), 명시된 기본 가상 키 및 명시된 속성에 대한 값을 갖는 모든 레코드들(예를 들어, "1001"의 기본 가상 키 및 "Smith"의 속성 "이름"에 대한 값을 갖는 모든 레코드들) 등을 요청할 수 있다. 일부 구현들에서, 질의는 가상 기본 키에 대한 참조를 포함하지 않는다.

[0029] 도 3a는 계층적 트리의 가상 클론들을 생성하고 그와 상호작용하기 위한 방법을 도시하는 흐름도이다. 도 3b는 도 3a의 방법에 따른 계층적 트리의 시간적 진화, 및 계층적 트리를 통한 데이터베이스 테이블과의 상호작용을 도시하는 일련의 도면들을 포함한다. 이해를 돕기 위해, 도 3a 및 도 3b는 함께 읽혀지고 이해될 수 있다. 도 3b에 도시된 바와 같이, 도 3a의 방법(200)의 개시 이전에, 하나 이상의 데이터베이스 테이블을 포함하는 데이터베이스(230)가 존재한다. 도 3a에서, 202에서, 방법(200)은 제1 가상 클론을 생성하는 것을 포함한다. 그에 대응하여 및 동시에, (b)에서, 데이터베이스(230)의 적어도 하나의 데이터베이스 테이블의 제1 가상 클론이 정의되고, 그에 의해 222에서 "V1"로서 표현되는 바와 같이 계층적 트리를 개시한다. 이어서, 204에서, 방법(200)은 제2 가상 클론을 생성하는 것을 포함하고, 그에 대응하여 및 동시에, (c)에서, 제2 가상 클론 "V2"(224)가 계층적 트리에 추가된다. 도 3b에서는 V1로부터 유래하고/하거나 시작하도록 도시되지만(따라서 V1의 "자식" 또는 "자손" 가상 클론이고, V1을 V2의 "조상" 또는 "선행자(predecessor)" 가상 클론으로 만듦), 다른 경우들에서, V2는 데이터베이스(230)로부터 직접 유래하고/하거나 시작할 수 있어서, 선행자 가상 클론을 갖지 않게 된다(이 경우 V1 및 V2는 계층적 트리 내의 형제(sibling) 가상 클론들이라고 지칭될 것이다). 방법(200)을 계속하면, 206에서, 도 3b의 (d)에 도시된 바와 같이, 질의가 V2를 통해 수신된다. 질의 Q(X)는 V2를 통해 수신되고, 레코드 ID "X"를 참조한다. 질의를 수신하는 것에 응답하여, 고속 카피 소프트웨어(예를 들어, 도 1의 고속 카피어(115))를 실행하는 프로세서는, 도 3a의 208에서, 레코드 ID "X" 뿐만 아니라, 버전 또는 다수의 버전들(이 경우, 도 3b의 (e)에 도시된 바와 같이, V1 또는 V2)을 참조하는 수정된 질의 Q(X, V1 또는 V2)를 생성하기 위해 질의를 수정한다. 수정된 질의는 관계형 데이터베이스가 요청된 레코드 ID "X"와 매칭되고 V1 또는 V2와 연관된 버전 식별자를 갖는 데이터베이스 테이블 레코드들에 대해 검색되도록 한다(가상 클론들 모두 데이터베이스(230)로부터 유래하는 직접적인 조상 라인에 있기 때문임). 210에서, 수정된 Q(X, V1 또는 V2) 질의가 도 3b의 (f)에 나타난 바와 같이 수행되고, 질의(응답) R(X, V1) 또는 R(X, V2)의 결과가 도 3b의 (g)에 도시된 바와 같이, 212에서 V2를 통해 반환 및 전송된다. (g)에 도시된 바와 같이, 레코드 ID X 및 V1과 연관된 응답 또는 레코드 ID X 및 V2와 연관된 응답이 반환된다. 가상 클론 V2를 통해 레코드 ID X에 대한 어떠한 수정도 아직 행해지지 않았다면, 레코드 ID X 및 V2를 참조하는 레코드는 존재하지 않을 것이고, 레코드 ID X와 연관된 레코드의 가장 최신의 형태는 V1과 연관된 버전 ID를 가질 것이다.

[0030] 도 4a 내지 도 4f는, 충돌 제약이 적용된 계층적 트리의 시간적 진화(단계들 (a) 내지 (f))를 도시하는, 가상 클론들의 표현들 및 연관된 데이터베이스 테이블을 포함한다. 도 4a의 시작(300)에서, 데이터베이스 테이블(230a)은 속성들 "FID"("Fruit ID", 기본 데이터베이스 테이블에 대한 레코드 ID 및/또는 키로서 기능함), "항목(Item)", "과일" 및 "버전", 및 단일의 채워진 레코드(single populated record)(FID=1001, 항목=Pie, 과일=Apple 및 버전=V1)를 포함한다. 또한, 300에서, ((a)에 도시된 계층적 트리에 의해 표현된 바와 같이) 가상 클론들 V1 및 V2(V1의 직접 자손)가 이미 존재한다. 이어서, 302에서, 새로운 가상 클론 V3(226)이 인스턴스화

되고, 계층적 트리에 추가되며, 데이터베이스 테이블(230a)에 대해 행해진 변경들은 아직 없다. 후속하여, 304에서, 파일에 대한 값이 "Pear"가 되도록, Pie에 대한 레코드를 변경하라는 요청이 V3를 통해 수신된다. 이 요청을 수신하는 것에 응답하여, 306에서, 버전 V3을 갖는 새로운 레코드가 데이터베이스 테이블(230a)에 추가된다. 도 4a에 도시된 바와 같이, FID 1001과 연관된 제2 레코드는 항목 속성에 대한 값 "Pie"를 보유하고, 파일 속성에 대해 "Apple" 대신에 "Pear"를 포함하고, 그것을 통해 변경이 요청되었던 가상 클론인 V3의 연관된 버전 식별자를 갖는다. 또한, 충돌 제약이 준수되기 때문에, 데이터베이스 테이블(230a)에서의 제1 레코드에서의 버전 속성에 대한 값은 (306에 도시된 바와 같이) V1로부터 V2로 변경되어, 충돌 제약이 충족되는 것을 보장한다. 충돌 제약이 시행되지 않았다면, 그러한 변경은 제1 레코드에 대해 행해질 필요가 없다는 것에 유의한다. (도 4a의 306에 도시된 바와 같은) 다수의 레코드들이 동일한 FID (또는 "가상 기본 키")를 가질 수 있는데, 그 이유는 복합 키가 FID 및 버전 속성 둘다를 포함하기 때문이라는 점에 유의한다. 즉, 306에서의 제1 레코드는 FID=1001 및 버전=V2의 복합 키를 갖고, 306에서의 제2 레코드는 FID=1001 및 버전=V3의 복합 키를 가지므로, 복합 키들은 여전히 고유하고 서로 구별가능하다.

[0031] 도 4b로 계속하면, 306에서의 데이터베이스 테이블(230a)의 수정에 후속하여, 308에서 V3을 통해 값 "Pie"를 참조하는 질의 Q(Pie)가 수신된다. 310에서, 값 "Pie" 및 "V3 또는 V1" 둘다를 참조하도록 질의가 수정되어 Q(Pie, V3 또는 V1)로 되는데, 그 이유는, 위의 논의와 유사하게, 가상 클론들 V3 및 V1이 데이터베이스(230)로부터 유래하고/하거나 시작하는 공통 조상 라인을 공유하기 때문이다. 그 다음, 수정된 질의가 데이터베이스 테이블(230a)에 대해 실행된다. "Pie" 및 "V3 또는 V1" 둘다와 매칭되는 데이터베이스 테이블(230)에서의 유일한 레코드는 FID 1001 및 버전 V3을 갖는 레코드이기 때문에, (312에 도시된 바와 같이) 질의는 그 레코드를 반환한다. 따라서, V3으로부터 시작된 질의 때문에, V1 및 V3과 연관된 레코드들은 검색되지만, V3과 동일한 조상 라인에 있지 않은 V2와 연관된 레코드들은 검색되지 않는다.

[0032] 도 4b에서의 310에서 충돌 제약을 준수하지 않는 구현들에서, 질의는 수정되지 않을 것이라는 점에 유의한다. 대신에, 프로세서는 먼저 값들 "Pie" 및 "V3"과 매칭되는 레코드를 검색할 것이다. (도 4b의 310의 예에서 그러할) 그러한 레코드가 발견되면, 그 레코드(FID=1001; 항목=Pie; 파일=Pear)는 반환되고, 질의가 종료된다. 그러나, 버전 트리에 기초하여, 값들 "Pie" 및 "V3"과 매칭되는 레코드가 데이터베이스(230)에서 발견되지 않으면, 프로세서는 다음으로 값들 "Pie" 및 "V1"과 매칭되는 레코드들에 대해 데이터베이스(230)에 질의할 것이다 (즉, 버전 트리 위로 올라간다). 그러한 복합 키를 갖는 레코드가 데이터베이스(230)에서 발견되면, 그 레코드는 반환되고 질의가 종료된다. 그렇지 않으면, 프로세서는 "널" 또는 "응답 없음"의 응답을 반환한다.

[0033] 도 4c로 계속하면, 추가 질의 Q(Pie)가 314에서 V2를 통해 수신되어, 속성 값 "Pie"를 참조한다. 다시, 값 "Pie" 및 "V2 또는 V1" 둘다를 참조하도록 질의가 수정되어 Q(Pie, V2 또는 V1)로 되는데, 그 이유는, 가상 클론들 V2 및 V1이 데이터베이스(230)로부터 유래하고/하거나 시작하는 공통 조상 라인을 공유하기 때문이다. 다음, 316에서, 수정된 질의가 데이터베이스 테이블(230a)에 대해 실행된다. "Pie" 및 "V2 또는 V1" 둘다와 매칭되는 데이터베이스 테이블(230a)에서의 유일한 레코드는 FID 1001 및 버전 V2를 갖는 레코드이기 때문에, (318에 도시된 바와 같이) 질의는 그 레코드를 반환한다. V2로부터 시작된 질의 때문에, V1 및 V2와 연관된 레코드들은 검색되지만, V2와 동일한 조상 라인에 있지 않은 V3과 연관된 레코드들은 검색되지 않는다. 따라서, V2 및 V3은 둘다 데이터베이스 테이블(230a)에서 동일한 레코드 1001의 상이한 버전들을 가질 수 있다.

[0034] 도 4d로 계속하면, 추가 질의 Q(Pear)가 320에서 다시 V2를 통해 수신되지만, 이번에는 속성 값 "Pear"를 참조한다. 다시, 질의는 값 "Pear" 및 "V2 또는 V1" 둘다를 참조하도록 수정되어 Q(Pear, V2 또는 V1)로 되는데, 그 이유는, 가상 클론들 V2 및 V1이 데이터베이스(230)로부터 유래하는 공통 조상 라인을 공유하기 때문이다. 그 다음, 322에서, 수정된 질의가 데이터베이스 테이블(230a)에 대해 실행된다. "Pear" 및 "V2 또는 V1" 둘다와 매칭되는 레코드가 데이터베이스 테이블(230a)에 없기 때문에, 324에서 결과들이 반환되지 않는다(예를 들어, 질의와 매칭되는 레코드들이 발견될 수 없다는 신호 및/또는 통지가 반환된다). 즉, V3을 통해 행해진 데이터베이스 테이블 데이터에 대한 수정은 V2를 통해 액세스가능하지 않고, V2의 질의가능 데이터에 영향을 미치지 않는다.

[0035] 도 4e의 326으로 계속하면, 326에서, V2를 통해, 데이터베이스 테이블(230a)로부터 파일 속성에 대해 Apple 및 항목 속성에 대해 Pie를 갖는 레코드들을 삭제하고, 파일 속성에 대해 Banana 및 항목 속성에 대해 Bread를 갖는 레코드를 추가하기 위한 요청이 수신된다. 328에서, 그리고 326에서 V2를 통해 수신된 요청에 응답하여, 속성 값들 "Apple" 및 "Pie"를 참조하는 레코드가 식별되고, 다음 중 하나 이상이 수행된다: 속성 값들 "Apple" 및 "Pie"를 참조하는 레코드가 데이터베이스 테이블(230a)로부터 삭제되고, 그 버전 ID가 제거(그것이 후속하여 V2를 통해 액세스가능하지 않을 것을 보장함)되거나, 또는 "삭제된" 플래그를 나타내는 속성이 데이터베이스 테

이블(230a)에 추가되고, 그 레코드에 대해 "참(ture)"으로 설정된다. 또한 328에서, 1002의 FID, "Bread"의 항목 속성 값, "Banana"의 과일 속성 값, 및 V2의 버전 ID를 갖는 새로운 레코드가 데이터베이스 테이블(230a)에 추가된다. 후속하여, 330에서, Bread의 값을 갖는 항목 속성을 갖는 레코드를 검색하는 질의 Q(Bread)가 V3을 통해 수신된다. 위의 논의와 유사하게, 질의는 먼저 Bread의 항목 속성 값 및 버전 ID "V3 또는 V1" 둘다를 참조하도록 수정되어 Q(Bread, V3 또는 V1)로 된다. "Bread" 및 "V3 또는 V1" 둘다와 매칭되는 레코드가 데이터베이스 테이블(230a)에 없기 때문에, 332에서 결과들이 반환되지 않는다(예를 들어, 질의와 매칭되는 레코드들이 발견될 수 없다는 신호 및/또는 통지가 반환된다).

[0036] 도 4e에서 설명된 바와 같이, 일부 구현들에서, 데이터베이스 테이블(230a)로부터의 레코드의 삭제는 레코드로부터 버전 속성을 제거하는 것에 의해 달성된다(즉, 데이터베이스 테이블(230a)과 연관된 임의의 가상 클론을 통해 행해진, 어떠한 후속 질의들도 레코드 또는 그의 임의의 부분을 반환하지 않을 것이다). 삭제들은, 예를 들어, 삭제되는 레코드가 삭제를 요청하는 가상 클론이 속하는 조상 경로에 존재하는 버전 ID를 갖는 경우, 이러한 방식으로 달성될 수 있다. 그러나, 삭제에 대한 요청이 요청하는 가상 클론의 조상 경로 및 삭제를 요청하지 않는 다른 경로에 의해 공유되는 버전 ID를 갖는 레코드를 참조하면, 삭제는 데이터베이스 테이블(230a)로부터 참조된 레코드를 삭제하고, 하나 이상의 새로운 레코드를 데이터베이스(230)에 추가함으로써 달성될 수 있고, 하나 이상의 새로운 레코드는 삭제를 요청하지 않는 다른 경로(또는 경로들)를 참조하는 연관된 버전 ID들을 갖는다(즉, 연관된 데이터가 다른 경로(들)를 따라 가상 클론들에 대해 액세스 가능한 채로 남아 있게 된다).

[0037] 다른 구현들에서, 데이터베이스 테이블(230a)에서의 레코드의 삭제를 달성하기 위해, (1) 삭제를 요청하는 가상 클론의 조상 경로 상에만 존재하는 버전 ID에 대한 레코드를 삭제하는 것이, "삭제된" 속성을 그 레코드에 대해 "참"으로 변경함으로써 달성되고, (2) 삭제를 요청하는 가상 클론의 조상 경로 및 다른 경로에 의해 공유되는 버전 ID를 갖는 레코드를 삭제하는 것이, (i) 삭제를 요청하는 가상 클론의 버전 ID를 갖는 새로운 레코드를 데이터베이스 테이블(230a)에 정의하고, 새롭게 생성된 레코드의 "삭제된" 속성을 "참"으로 설정하는 것, 및 (ii) 다른 경로 상의 가상 클론의 버전 ID를 갖는 새로운 레코드를 데이터베이스 테이블(230a)에 정의하고, 새롭게 생성된 레코드의 "삭제된" 속성을 "거짓(false)"으로 설정하는 것에 의해 달성되도록, (참 또는 거짓의 값들을 취하는) "삭제된" 속성이 데이터베이스 테이블(230a)의 각각의 레코드에 추가될 수 있다.

[0038] 도 4f를 참조하면, 334에서 V2를 통해 요청 Q(Banana)가 수신되어, Banana의 값을 갖는 과일 속성을 갖는 레코드를 검색한다. 단계 336에서, 질의는 Banana의 항목 속성 값 및 버전 ID "V2 또는 V1" 둘다를 참조하도록 수정되어 Q(Banana, V2 또는 V1)로 되고, 수정된 질의가 데이터베이스 테이블(230a)에 대해 실행된다. "Banana" 및 "V2 또는 V1" 둘다와 매칭되는 데이터베이스 테이블(230a)에서의 유일한 레코드는 FID 1002 및 버전 V2를 갖는 레코드이기 때문에, (338에 도시된 바와 같이) 질의가 그 레코드를 반환한다.

[0039] 도 4g는 가상 클론들 V1, V2 및 V3 각각에 대한 레코드들을 포함하는 (그리고 레코드 FID=1001 및 버전=V1이 삭제되지 않는 것으로 도시된), 340에서의 데이터베이스 테이블(230a)을 갖는 도 4f의 버전 트리를 도시한다. 340에서의 데이터베이스 테이블(230a)은, 예를 들어, 데이터베이스 테이블(230a)이 충돌 제약으로 구현되지 않았고, 도 4a의 304에서 시작하는 경우, (1) "Pie"를 "Apple"로부터 "Pear"로 변경하라는 요청이 FID=1001, "Pear"의 "과일" 속성에 대한 값 및 V3의 "버전" 속성에 대한 값(306으로 도시됨)을 갖는 새로운 레코드의 추가, 및 FID=1001, "Apple"의 "과일" 속성에 대한 값 및 V1의 "버전" 속성에 대한 값을 갖는 레코드의 보유(304로 도시됨)를 초래하고, (2) Banana Bread를 추가하라는 요청이 V2를 통해 행해지도록(FID=1002, "Bread"의 "항목" 속성에 대한 값, "Banana"의 "과일" 속성에 대한 값 및 V2의 "버전" 속성에 대한 값의 데이터베이스 테이블(230a)로의 추가를 야기함), 초래할 수 있다.

[0040] 340에서의 데이터베이스 테이블(230a)의 구현이 충돌 제약을 준수하지 않는다는 가정으로 계속하면, "Pie"의 "항목" 속성에 대한 값을 참조하는 V2에서 수신된 질의는 동시에 V2 및 V1 둘다를 검색하기 위해 프로세서에 의해 수정되지 않을 것이다. 대신에, 프로세서는 먼저 값들 "Pie" 및 "V2"와 매칭되는 레코드를 검색할 것이다. 그러한 레코드는 340에서 데이터베이스 테이블(230a)에 존재하지 않기 때문에, 프로세서는 다음으로 값들 "Pie" 및 "V1"과 매칭되는 레코드들에 대해 데이터베이스(230)에 질의할 것이다(즉, 버전 트리 위로 올라간다). 단일 레코드(FID=1001; 항목=Pie; 과일=Apple; 버전=V1)가 데이터베이스 테이블(230a)에서 발견되고, 그 레코드는 반환되고 질의가 종료된다. V3에 대해 행해진 유사한 질의는, V3이 먼저 검색될 것이기 때문에 레코드(FID=1001; 항목=Pie; 과일=Pear; 버전=V3)를 반환할 것이고, 레코드가 식별되었기 때문에, 질의는 종료될 것이고, 조상 경로를 따라 V1을 검색하는 것을 계속하지 않을 것이다. 대신에, 질의가 "모든 Pie들"에 대해 V3에서 수신되었다면, 프로세서는 먼저 값들 "Pie" 및 "V3"과 매칭되는 레코드에 대해 데이터베이스 테이블(230a)을 검색할 것이

고, 그에 의해 제1 레코드(FID=1001; 항목=Pie; 과일=Pear; 버전=V3)를 식별할 것이다. 그 다음, 프로세서는 "항목" 속성에 대해 "Pie"의 값을 포함하는 임의의 추가 레코드들에 대해 (조상 경로를 따라) V1을 검색함으로써 계속할 것이고, 제2 레코드(FID=1001; 항목=Pie; 과일=Apple; 버전=V1)를 식별할 것이다. 그러나, (예를 들어, 레코드 FID들의 비교에 기초하여 프로세서에 의해 검출된 바와 같이) 제1 레코드 및 제2 레코드의 FID들이 매칭되므로, V3의 검색은 FID=1001을 갖는 레코드를 이미 식별했기 때문에, 버전 V1을 갖는 레코드는 질의 결과들로부터 배제될 것이다. 대신에, 추가 레코드들이 값 "Pie"와 매칭되지만 1001과 상이한 FID를 갖는 V1을 검색시에 식별되었다면, 이들 레코드들은 질의 응답의 일부로서 반환될 것이다.

[0041] 가비지 수집(Garbage Collection)

[0042] 일부 구현들에서, 계층적 트리(본 명세서에서 "버전 트리" 라고도 지칭됨) 및/또는 계층적 트리의 표현은, 메모리에 동작가능하게 및 통신가능하게 결합되는 (예를 들어, 도 1의 고속 카피어(115)를 실행하는 프로세서(111)와 같은) 프로세서에 의해 수행되는 증분적 가비지 수집(incremental garbage collection)의 형태를 통해, 감소된 및/또는 보다 최적의 형태로, 비일시적 메모리에 유지 및/또는 저장된다. 그러한 구현들에서, 계층적 트리의 각각의 노드는 가상 클론, 및 다수의 정수 시퀀스 ID들의 문자열의 엔드포인트들을 나타내는 2개의 시퀀스 ID들과 연관된다. 예로서, 일련의 시퀀스 ID들 10, 11, 12 및 13은 시퀀스 ID 쌍 (10, 13)에 의해 표현될 수 있다. 시퀀스 ID 42에만 연관된 노드는 시퀀스 ID 쌍 (42, 42)로 지정될 것이다. 본 명세서에서 이용된 바와 같이, 시퀀스 ID는 버전 ID와는 상이하게 정의된다. 시퀀스 ID는 계층적 트리의 가상 클론들이 생성되었던 순차적 순서와 관련된다. 그와 같이, 특정 가상 클론에 대한 시퀀스 ID 및 버전 ID는 동일한 값을 가질 수 있거나, 상이할 수 있다. 일부 그러한 구현들에서, 도 5의 도면들 (a) 내지 (g)에 도시된 바와 같이, (a)에서 트리 로 확장하기 전에 처음에 버전 트리는 단일 노드 (1, 1)만을 포함한다. 고속 카피 이벤트가 버전 트리에서의 새로운 노드의 생성 및/또는 정의를 트리거할 때마다, 그 새로운 노드는 순차적 순서로 다음 정수의 시퀀스 ID를 할당받는다. 그와 같이, 버전 트리에 추가된 제2 노드는 시퀀스 ID (2, 2)를 할당받고, N번째 노드는 시퀀스 ID (N, N)을 할당받는다. 본 명세서에서 이용된 바와 같이, X가 시퀀스 ID 쌍 (A, B)로 지정되고 Y가 시퀀스 ID 쌍 (B+1, C)로 지정되면, 노드 X는 "후속자(successor)" 노드 Y를 갖는다. 즉, Y는 X의 시퀀스 ID 범위 직후에 오는 시퀀스 ID 범위를 나타낸다. 메모리에서의 (및 데이터베이스에서의) 데이터 구조는 버전 트리의 각각의 노드에 대해, 그의 후속자인 노드(들)에 대한 참조를 저장할 수 있다. 후속자 관계들은 도 5에서 파선들로서 표현된다. 본 명세서에서 이용된 바와 같은 "후속(succession)" 및 "자손(descendancy)"은 상이한 (상호 배타적이지 않은) 개념들이다. 예를 들어, 이전 노드의 후속자인 노드는 반드시 그 이전 노드의 자손일 필요는 없고(예를 들어, 도 5의 (a)에서, 노드 (6, 6)은 노드 (5, 5)의 후속자이지만, 노드 (5, 5)의 조상 경로 상의 자손이 아님), 이전 노드의 자손 노드는 반드시 그 이전 노드의 직속 후속자일 필요는 없다(예를 들어, 도 5의 (a)에서, 노드 (6, 6)은 노드 (3, 3)의 자손이지만, 노드 (3, 3)의 직속 후속자는 아님).

[0043] 일부 구현들에서, (예를 들어, 버전 트리로부터 중복성을 제거하고, 성능 및 효율성을 증가시키고, 스토리지 및/또는 리소스 소비를 감소시키는 등등을 위해) 메모리에 저장된 (예를 들어, 노드 X 및 그의 후속자 Y 노드를 수반하는) 노드들에 대한 "시퀀스 업데이트"는 고속 카피 소프트웨어(예를 들어, 도 1의 고속 카피어(115, 125))를 실행하는 프로세서에 의해 수행되고, 프로세서는 버전 트리를 저장하는 메모리와 동작가능하게 통신한다. 시퀀스 업데이트는 다음과 같이 달성된다: 첫째로, (A, B)의 시퀀스 ID 범위를 갖는 노드 X가 (B+1, C)의 시퀀스 ID 범위를 갖는 후속자 Y를 가지며, 노드들 X 및 Y를 포함하는 노드들의 시퀀스가 업데이트될 것이라고 가정한다. (A, C)의 시퀀스 ID 범위를 갖는(즉, 노드 X의 제1 엔드포인트 및 노드 Y의 제2 엔드포인트를 갖는) 새로운 노드 Z가 정의될 수 있다. 다음, 노드 X 또는 노드 Y에 대한 버전 트리에서의 다른 노드들의 임의의 관계들은 (예를 들어, 도 5에서 실선 및 파선으로 표현되는 바와 같이) 그들이 노드 X 또는 Y에 더 이상 접속하고/하거나 이들을 지칭하지 않고, 그 대신에 노드 Z에 접속하고/하거나 그것을 지칭하도록 업데이트된다. 그 다음, 노드들 X 및 Y는 버전 트리로부터 제거/삭제될 수 있고(그에 대응하여, 연관된 데이터베이스 테이블로부터 삭제되거나, 연관된 데이터베이스 테이블에서 삭제된 것으로서 주석이 달리게 됨), 그에 의해 시퀀스 업데이트 이벤트를 완료하게 된다. 일부 구현들에서, 시퀀스 업데이트가, 또한 그 노드의 후속자인 단일 자식 노드만을 갖는 노드를 초래하는 경우, 전술한 것과 유사한 방식으로, 그 노드 및 그의 단일 자식 노드를 수반하는 다른 시퀀스 업데이트가 (예를 들어, 노드가 또한 그 노드의 후속자인 단일 자식 노드만을 갖는 조건을 검출할 때, 자동으로) 수행될 수 있다.

[0044] 일부 구현들에서, 버전 트리로부터의 노드 X의 삭제는 다음과 같이 달성된다: 첫째, 노드 X가 버전 트리에 존재하는 부모 노드를 갖는다면, 노드 X와 그것의 부모 노드 사이의 (예를 들어, 도 5에 도시된 바와 같이, 버전 트리에서의 실선에 의해 표현된) 관계는 제거된다. 다음, 노드 X가 후속자 노드를 갖는다면, 그 후속자 노드는

새로운 시퀀스 ID들을 할당받고, 노드 X는 삭제된다. 대안적으로, 노드 X가 후속자 노드를 갖지 않으면, 노드 X는 단지 삭제된다. 노드 X의 이전의 부모 노드가 이제 그로부터 유래하는 자식 노드들을 갖지 않으면, 이전의 부모 노드는 삭제된다. 후속자로서 노드 X를 갖는 노드 W가 있었고, 노드 W가 이제 하나의 자식 노드만을 갖고, 그 자식이 그의 후속자인 경우, 노드 W의 자식 노드는 새로운 시퀀스 ID들을 할당받는다.

[0045] 전술한 임의의 삭제 또는 시퀀스 업데이트 이벤트는 새롭게 자식들을 갖지 않는 노드를 초래할 수 있고, 따라서 다른 삭제를 트리거할 수 있다. 대안적으로 또는 추가적으로, 전술한 임의의 삭제 또는 시퀀스 업데이트 이벤트는 그의 후속자인 단일 자식 노드를 갖는 새로운 노드를 초래할 수 있고, 따라서 다른 시퀀스 업데이트 이벤트를 트리거할 수 있다. 따라서, 단일 시퀀스 업데이트 또는 삭제에 의해 야기되는 시퀀스 업데이트들 및 삭제들의 캐스케이드가 존재할 수 있다. 일부 구현들에서, 본 개시내용의 시스템은 캐스케이딩 타입 가비지 수집(cascading-type garbage collection)을 수행하도록 구성되고, 다른 형태의 가비지 수집은 버전 트리에 대해 수행되지 않는다. 본 명세서에서 설명되는 (그리고 프로세서에 의해 실행되는) 증분적 가비지 수집 방법들은 (예를 들어, 실질적으로 연속적으로 및/또는 실질적으로 실시간으로) 만들어질 버전 트리에서의 가능한/허용가능한 단순화들(또는 그것의 미리 정의된 서브세트)을 자동으로 인식하고, 실행/구현/야기할 수 있다. 이어서, 수정된/단순화된 버전 트리의 표현(예를 들어, 도 5의 (d)의 버전 트리를 대체할 도 5의 (e)의 버전 트리)은, 이후 액세스를 위해 메모리에 저장될 수 있다.

[0046] 일부 구현들에서, 삭제 이벤트는 증분적 가비지 수집 프로세스를 트리거할 수 있다. 예를 들어, 버전 트리도 5의 (a)에 도시된 바와 같이 표현되는 경우, 파선 화살표들은 후속자 노드들에 대한 참조들을 나타낸다. 고속 카피가능 데이터베이스의 일부 구현들에서, 리프 노드들만이 삭제될 수 있다. 즉, 고속 카피 소프트웨어는, 그 버전이 "리프" 노드와 연관되는 경우에만 "버전"을 삭제하도록 구성될 수 있다. "리프 노드"는 자식들을 갖지 않는 노드로서 정의된다(여기서, 자식 노드들은 도 5에서 실선으로 표현된다). 파선들은 리프 노드들의 식별 동안 무시될 것이다.

[0047] 사용자가 도 5의 (a)에서 음영 회색(shaded gray)의 리프 노드 (5, 5)의 삭제를 요청한다고 가정한다. 리프 노드 (5, 5)의 삭제 후의 버전 트리는 도 5의 (b)에 도시된 바와 같이 나타난다. (b)의 버전 트리는 노드 (6, 6)이 시퀀스 ID 쌍 (5, 6)을 할당받는 시퀀스 업데이트의 결과이고, 그 다음에 (노드 (4, 4)를 노드 (5, 6)과 접속하는 파선에 의해 표현되는 바와 같이) 노드 (4, 4)가 그의 후속자로서의 노드 (5, 6)을 참조하도록 업데이트된다. 이전의 노드 (5, 5)의 부모 노드인 (2, 2)는 자식 노드(노드 (4, 4))를 여전히 가지며, 그의 후속자 노드(노드 (3, 3))를 그의 유일한 자식으로서 갖지 않고, 따라서 삭제 프로세스는 도 5의 (b)에 도시된 버전 트리로 정지한다는 점에 유의한다.

[0048] 다음, 사용자가 (도 5의 (b)에서 음영 회색인) 노드 (4, 4)의 삭제를 요청한다고 가정한다. 노드 (4, 4)를 삭제하기 위한 이러한 요청은 여러 동작들을 트리거할 수 있다. 먼저, 노드 (4, 4)가 부모 노드 (2, 2)를 갖기 때문에, 도 5의 (b)에서 실선으로 표현된, 노드 (2, 2)와 노드 (4, 4) 사이의 관계가 제거된다. 또한, 도 5의 (c)에 도시된 바와 같이, 노드 (5, 6)이 시퀀스 ID 쌍 (4, 6)을 할당받도록 시퀀스 업데이트가 수행되어, 이제 노드 (3, 3)과 노드 (4, 6) 사이의 업데이트된 후속자 관계를 갖게 된다(노드 (4, 6)은 노드 (3, 3)의 후속자임). 전술한 동작들의 결과로서, 노드 (2, 2)는 (도 5의 (c)에 도시된 바와 같이) 어떠한 자식 노드들도 더 이상 갖지 않는다. 그와 같이 및/또는 이러한 조건에 응답하여, 그리고 (그들의 제거 이전에, 이전에 생성된 노드 (2, 2)의 자손 노드들의 결과로서) 노드 (2, 2)를 통해 질의들이 더 이상 행해질 수 없다는 사실로 인해, 삭제 프로세스는 노드 (2, 2)의 삭제로 자동으로 계속함으로써 "캐스케이드" 할 수 있고, 이는 도 5의 (d)에 도시된 바와 같은 버전 트리로 이어질 수 있다. 이 단계에서, 노드 (1, 1)은 단지 하나의 자식(노드 (2, 3))만을 가지며, 그 자식은 그의 후속자 노드이다. 따라서, 위에서 논의된 바와 같이, 자식 노드 (2, 3)이 새로운 시퀀스 ID 쌍 (1, 3)을 할당받도록 시퀀스 업데이트가 수행될 수 있고, 도 5의 (e)에 도시된 바와 같은 버전 트리를 초래하게 되며, 그 시점에 캐스케이딩이 종료된다. 즉, 도 5의 (e)에 도시된 바와 같은 버전 트리는 노드 (4, 4)를 삭제하라는 사용자의 요청의 최종 결과이다.

[0049] 이제, 사용자가 (도 5의 (e)에서 음영 회색인) 리프 노드 (4, 6)을 삭제하기로 선택한다고 가정한다. 이러한 삭제 이벤트는 그 자체가 최종 시퀀스 업데이트를 자동으로 트리거하는 도 5의 (f)에 도시된 바와 같이 나타나도록 버전 트리의 변환을 초래할 것이고, 따라서 노드 (4, 7)(노드 (1, 3)의 자식 노드)은 새로운 시퀀스 ID 쌍 (1, 7)을 할당받고, 이는 도 5의 (g)에 도시된 최종 결과(유일한 노드 (1, 7))를 제공한다. 가비지 수집이 진행됨에 따라, 연관된 데이터베이스 테이블에 대한 대응하는 변경들은, 예를 들어, 도 5의 (g)에서 모든 레코드들이 동일한 버전 속성(예를 들어, "7")을 포함하도록 행해질 수 있다. (예를 들어, 도 5의 (g)에서의) 단지 하나의 노드가 남아 있는(즉, 버전 트리가 "가비지 수집된" 그리고 크기가 감소된) 단계 이전에, (예를 들어,

도 4a 내지 도 4g의 데이터베이스 테이블(230a)과 같은) 데이터베이스 테이블의 레코드들은 잔존 노드(surviving node)를 지칭하도록 수정될 수 있거나, 또는 (예를 들어, 그 레코드와 고유하게 연관되는 노드의 삭제에 응답하여) 삭제될 수 있다.

[0050] 도 6은 실시예에 따른, 고속 카피 데이터베이스를 위한 프로세서에 의해 구현된 가비지 수집(또는 캐스케이딩 삭제) 방법(400)에 대한 프로세스 흐름도이다. 도 6에 도시된 바와 같이, 방법(400)은 430에서 가상 클론들의 계층적 트리의 생성으로 시작한다. 이어서, 432에서, 가상 클론들의 계층적 트리로부터 제2 가상 클론을 삭제하라는 명령어가 수신된다. 434에서, 제2 가상 클론을 삭제하라는 명령어에 응답하여, 제3 가상 클론의 후속자 관계가 수정된다. 또한, 436에서, 제2 가상 클론을 삭제하라는 명령어에 응답하여, 제2 가상 클론이 삭제된다. 438에서, 프로세서는 제2 가상 클론의 삭제 후에 가상 클론들의 계층적 트리의 제1 가상 클론이 남아 있는 자식 가상 클론을 갖는지를 평가한다. 제2 가상 클론의 삭제 후에 가상 클론들의 계층적 트리의 제1 가상 클론이 남아 있는 자식 가상 클론을 갖지만, 그 자식이 그의 후속자가 아닌 경우, 프로세서는 종료된다. 제2 가상 클론의 삭제 후에 가상 클론들의 계층적 트리의 제1 가상 클론이 남아 있는 자식 가상 클론을 갖고, 그 자식이 그의 후속자인 경우, 위에서 논의된 바와 같이, 시퀀스 업데이트가 수행될 수 있다. 제2 가상 클론의 삭제 후에 가상 클론들의 계층적 트리의 제1 가상 클론이 남아 있는 자식 가상 클론을 갖지 않는 경우, 440에서, 가상 클론들의 계층적 트리로부터 제1 가상 클론이 삭제된다.

[0051] 도 7은 실시예에 따른, 고속 카피 데이터베이스에 대한 추가적인 프로세서에 의해 구현된 가비지 수집(또는 캐스케이딩 삭제) 방법(500)에 대한 프로세스 흐름도이다. 도 7에 도시된 바와 같이, 방법(500)은 530에서 가상 클론들의 계층적 트리의 생성으로 시작한다. 이어서, 532에서, 가상 클론들의 계층적 트리로부터 제1 가상 클론을 삭제하라는 명령어가 수신된다. 제1 가상 클론(예를 들어, 도 5의 노드 (4, 4))를 삭제하라는 명령어에 응답하여, 그리고 534에서, 제2의 후속자 가상 클론(예를 들어, 도 5의 노드 (5, 6))의 후속자 관계가 수정된다. 536에서, 제1 가상 클론이 삭제된다. 이어서, 538에서, 제4 가상 클론(예를 들어, 도 5의 노드 (3, 3))의 후속자 관계가 수정되고, 여기서 제4 가상 클론은 제1 가상 클론의 조상인 제3 가상 클론(예를 들어, 도 5의 노드 (2, 2))의 후속자이다. 이어서, 540에서 제3 가상 클론이 삭제된다.

[0052] 도 8a 내지 도 8d는 실시예에 따른, 고속 카피 데이터베이스 시스템의 예시적인 구현을 도시하는 도면들이다. 도 8a에 도시된 바와 같이, 800에서 (그리고 초기 시간 $t=0$ 에서), 노드 "Jane"에서의 데이터베이스의 제1 인스턴스(830A)는 데이터베이스 인스턴스(830A)에 따라, 3개의 가상 클론들 V1, V2 및 V3을 포함하는 계층적 트리의 부분을 형성한다. 노드 Jane에서의 데이터베이스 인스턴스(830A)는 속성들 RID, 이름, 랭크 및 버전을 갖는 데이터베이스 테이블을 포함한다. 800에서, 데이터베이스 테이블에 2개의 채워진 레코드들이 있는데, 하나는 이름=Smith를 갖고, 하나는 이름=Jones를 갖는다. 800에서의 가상 클론 V1은 이전의 "제1" 합의 이벤트와 연관될 수 있다. 800에서의 가상 클론 V2는 (예를 들어, 데이터베이스 테이블은 트랜잭션 데이터와 같은 새로운 데이터가 수신됨에 따라 진행 중인 집계들 또는 상태들을 반영하도록) 데이터베이스(830A)의 데이터베이스 테이블에 실시간 또는 실질적으로 실시간 편집들을 행하기 위해 이용될 수 있는 반면, 800에서의 가상 클론 V3은 데이터베이스(830A)의 데이터베이스 테이블에 대한 합의후 편집들을 행하기 위해 예약될 수 있다. 또한, 도 8a에 도시된 바와 같이, 802에서 (그리고 초기 시간 $t=0$ 에서), 또한 각각 데이터베이스 인스턴스(830B)의 직접적인 의존인 2개의 가상 클론들 V1 및 V2를 포함하는 다른 계층적 트리의 일부를 형성하는 노드 "Bob"(803B)에서의 데이터베이스의 제2 인스턴스가 도시된다. 노드 Bob에서의 데이터베이스 인스턴스(830B)는 또한 속성들 RID, 이름, 랭크 및 버전을 갖는 데이터베이스 테이블을 포함한다. 802에서, 데이터베이스 테이블(830B)에 2개의 채워진 레코드들이 있는데, 하나는 이름=Smith를 갖는 것이고, 하나는 이름=Jones를 갖는 것이다. 즉, 제1 시간 $t=0$ 에서, 노드들 Jane(830A) 및 Bob(830B)에서의 데이터베이스 인스턴스들의 데이터베이스 테이블들은 동일한 레코드들을 포함한다. 노드 Jane(830A) 및 노드 Bob(830B)에서의 데이터베이스 테이블들에서의 레코드들 각각은 V1의 버전 속성에 대한 값을 포함한다는 점에 유의한다. 802에서의 가상 클론 V1은 새로운 자식 가상 클론들 V2 및 V3의 생성을 트리거한 이전의 "제1" 합의 이벤트와 연관될 수 있다. 802에서의 가상 클론 V2는 데이터베이스(830B)의 데이터베이스 테이블에 대한 실시간 또는 실질적으로 실시간 편집들을 행하기 위해 이용될 수 있는 반면, 802에서의 가상 클론 V3은 데이터베이스(830B)의 데이터베이스 테이블에 대한 합의후 편집들을 행하기 위해 예약될 수 있다.

[0053] 도 8b에서, 동기화 이벤트가 발생하는데, 이 동안에, 노드 Jane(830A) 및/또는 노드 Bob(830B)과의 네트워크 통신에서, 노드 Jane(830A), 노드 Bob(830B), 및 선택적으로 하나 이상의 추가적인 노드 중 둘 이상의 노드 사이에서 데이터(본 명세서에서 "이벤트"라고도 지칭되며, 예를 들어, 실시간으로 또는 실질적으로 실시간으로, 그들이 속하는 트랜잭션들과 동시에 수신될 수 있는 트랜잭션 데이터)가 교환된다. 동기화 동안, 노드

Jane(830A)은 제1 시간에, "Smith"의 속성 "이름"에 대한 값을 갖는 레코드에 대해 속성 "랭크"에 대한 값이 1만큼 증분될 것임을 명시하는 제1 이벤트를 수신한다. 제1 이벤트를 수신하는 것에 응답하여, 노드 Jane(830A)에서의 프로세서는 "Smith"에 대한 새로운 레코드를 생성하고, 그 레코드에서, V2의 속성 "버전"에 대한 값, 1001의 속성 "RID"에 대한 값, 및 $2+1=3$ 의 속성 "랭크"에 대한 값(도시되지 않음)을 포함한다. 또한, 제1 이벤트를 수신하는 것에 응답하여, 노드 Jane(830A)에서의 프로세서는 버전 속성에 대한 값이 V1 대신에 V3이도록 데이터베이스 테이블에서 제1 레코드를 수정한다. 또한, 동기화 동안(동일한 동기화 이벤트 동안 또는 별개의 동기화 이벤트 동안 중 어느 하나 동안), 노드 Jane(830A)은 제1 시간 이후의 제2 시간에, "Smith"의 속성 "이름"에 대한 값을 갖는 레코드에 대한 속성 "랭크"에 대한 값이 10으로 변경될 것임을 명시하는 제2 이벤트를 수신한다. 제2 이벤트를 수신하는 것에 응답하여, 노드 Jane(830A)에서의 프로세서는 "Smith"의 이름 속성에 대한 값 및 "V2"의 버전 속성에 대한 값을 갖는 레코드를 수정하며, 따라서 레코드는 (도 8b의 804에 도시된 바와 같이) 10의 속성 "랭크"에 대한 값을 갖는다.

[0054] 동기화 동안(노드 Jane(830A)과 동일하거나 또는 별개의 동기화 이벤트 동안), 노드 Bob(830B)은 제1 및 제2 이벤트들을, Jane(830A)이 이벤트를 수신한 것과는 반대 순서로 수신한다. 보다 구체적으로, 노드 Bob(830B)은 제1 시간에, "Smith"의 속성 "이름"에 대한 값을 갖는 레코드에 대한 속성 "랭크"의 값이 10으로 변경될 것임을 명시하는 제1 이벤트를 수신한다. 제1 이벤트를 수신하는 것에 응답하여, 노드 Bob(830B)에서의 프로세서는 "Smith"에 대한 새로운 레코드를 생성하고, 그 레코드에서, V2의 속성 "버전"에 대한 값, 1001의 속성 "RID"에 대한 값, 및 10의 속성 "랭크"에 대한 값(도시되지 않음)을 포함한다. 또한, 제1 이벤트를 수신하는 것에 응답하여, 노드 Jane(830A)에서의 프로세서는 버전 속성에 대한 값이 V1 대신에 V3이도록 데이터베이스 테이블에서 제1 레코드를 수정한다. 동기화 동안(동일한 동기화 이벤트 동안 또는 별개의 동기화 이벤트 동안 중 어느 하나 동안), 노드 Bob(830B)은 제1 시간 이후의 제2 시간에, "Smith"의 속성 "이름"에 대한 값을 갖는 레코드에 대한 속성 "랭크"에 대한 값이 1만큼 증분될 것임을 명시하는 제2 이벤트를 수신한다. 제2 이벤트를 수신하는 것에 응답하여, 노드 Bob(830B)에서의 프로세서는 "Smith"의 이름 속성에 대한 값 및 "V2"의 버전 속성에 대한 값을 갖는 레코드를 수정하여, (도 8b의 806에 도시된 바와 같이) 레코드가 11의 속성 "랭크"에 대한 값을 갖도록 한다.

[0055] 노드 Jane 및 노드 Bob에서 V2를 통해 수신되는 이벤트들/데이터(예를 들어, 트랜잭션 데이터)는 노드 Jane 및 노드 Bob의 연관된 메모리들에 각각, 예를 들어, 분산 비순환 그래프(distributed acyclic graph)(DAG), 해시 그래프, 데이터 체인, 블록체인, 또는 임의의 다른 적절한 데이터 구조 또는 포맷으로 저장될 수 있다. 호환 가능한 데이터 구조들을 포함하는 추가적인 구현 상제들은, 그 전체 내용이 모든 목적들을 위해 본 명세서에 참조로 포함되는, 미국 특허 제9,646,029호에 개시되어 있다.

[0056] 동기화 이벤트(들) 이후에, 그리고 도 8c의 808에 도시된 바와 같이, 노드들 Jane(830A) 및 Bob(830B)의 프로세서들은 (예를 들어, 도 1의 합의 명령어들(116 및 126)과 같은 합의 명령어들의 구현의 결과로서) 합의에 도달한다. 즉, (그 노드의 메모리에 저장된 고속 카피 소프트웨어를 실행하는) 노드 Jane(830A)의 프로세서에 의해 구현되는 합의 알고리즘 및 (예를 들어, 그 노드의 메모리에 저장된 고속 카피 소프트웨어를 실행하는) 노드 Bob(830B)의 프로세서에 의해 구현되는 합의 알고리즘은 합의에 도달하였다는 것을 검출 및/또는 결정한다. 비제한적인 예로서, 다음과 같은 조건들 중 하나 이상 하에서 합의에 도달할 수 있다: 합의 알고리즘에 의해 정의된 "라운드"가 완료될 때, 계층적 트리가 개시된 이후에 미리 결정된 양의 시간이 경과했을 때, 이전의 합의 이벤트 이후에 미리 결정된 양의 시간이 경과했을 때, 해당 노드의 데이터베이스 테이블의 주어진 레코드에 대해 다른 노드들로부터 미리 결정된 수의 투표들(votes)이 수신되었을 때, 미리 결정된 수의 동기화 이벤트가 발생했을 때 등. 도 8c에 도시된 바와 같이, 합의의 결과는, 수신된 이벤트들의 정확한 순서화가 Smith에 대한 랭크가 먼저 증분되었고, 그 다음 10의 값으로 변경된다는 것이었다. 그와 같이, Smith의 이름 속성에 대한 값 및 V2의 버전 속성에 대한 값을 갖는 레코드에 대한 랭크 속성에 대한 정확한 현재 값은 "10"이다. 유사하게 말하면, 808에 도시된 바와 같이, 이 레코드에 대한 랭크 값은 V3을 통해 "10"이 되도록 업데이트된다. 합의에 도달한 결과로서, 가상 클론 V3에 매달리는/그로부터 내려가는 가상 클론들 V4 및 V5의 새로운 쌍이 생성된다. 808에서, 새롭게 생성된 가상 클론 V4는 이어서 노드 Jane(830A)에서의 데이터베이스 인스턴스의 데이터베이스 테이블에 실시간 또는 실질적으로 실시간 편집들을 행하기 위해 이용될 수 있는 반면, 808에서 가상 클론 V5는 노드 Jane(830A)에서의 데이터베이스 인스턴스의 데이터베이스 테이블에 대한 합의후 편집들을 행하기 위해 예약될 수 있다. 즉, 새로운 가상 클론들 V4 및 V5는 각각 가상 클론들 V2 및 V3에 의해 이전에 수행된 기능들을 수행할 수 있다. 일부 구현들에서, 일단 가상 클론들 V4 및 V5가 생성되었다면, (가상 클론들 V4 및 V5가 후속하여 삭제되는 경우에도) 가상 클론 V3을 통해 데이터베이스 테이블에 대한 편집들이 더 이상 행해질 수 없다는

것에 유의한다.

[0057] 도 8c의 논의와 유사하게, 도 8d에서, 노드 Bob(830B)은 (예를 들어, 도 1의 합의 명령어들(116 및 126)과 같은 합의 명령어들의 구현의 결과로서) 합의에 도달한다. 즉, (예를 들어, 그 노드의 메모리에 저장된 고속 카피 소프트웨어를 실행하는) 노드 Jane(830A)의 프로세서에 의해 구현되는 합의 알고리즘 및 (예를 들어, 그 노드의 메모리에 저장된 고속 카피 소프트웨어를 실행하는) 노드 Bob(830B)의 프로세서에 의해 구현되는 합의 알고리즘은 합의에 도달하였다는 것을 검출 및/또는 결정한다. 도 8d에 도시된 바와 같이, 노드 Bob(830B)에서의 데이터베이스 인스턴스의 데이터베이스 테이블은 이제 Smith의 이름 속성에 대한 값 및 V2의 버전 속성에 대한 값을 갖는 레코드에 대한 랭크 속성에 대한 정확한 현재 값을 "10"으로서 반영한다. 유사하게 말하면, 810에 도시된 바와 같이, 이 레코드에 대한 랭크 값은 V3을 통해 "10"이 되도록 업데이트된다. 도 8c의 논의와 유사하게, 그리고 제2 합의에 도달한 결과로서, 가상 클론 V3에 매달리는/그로부터 내려가는 가상 클론들 V4 및 V5의 새로운 쌍이 생성된다. 808에서, 새롭게 생성된 가상 클론 V4는 이어서 노드 Bob(830B)에서의 데이터베이스 인스턴스의 데이터베이스 테이블에 실시간 또는 실질적으로 실시간 편집들을 행하기 위해 이용될 수 있는 반면, 808에서 가상 클론 V5는 노드 Bob(830B)에서의 데이터베이스 인스턴스의 데이터베이스 테이블에 대한 합의후 편집을 행하기 위해 예약될 수 있다. 즉, 새로운 가상 클론들 V4 및 V5는 각각 가상 클론들 V2 및 V3에 의해 이전에 수행된 기능들을 수행할 수 있다. 일부 구현들에서, 일단 가상 클론들 V4 및 V5가 생성되었다면, (가상 클론들 V4 및 V5가 후속하여 삭제되는 경우에도) 가상 클론 V3을 통해 데이터베이스 테이블에 대한 편집들이 더 이상 행해질 수 없다는 것에 유의한다.

[0058] 도 8a 내지 도 8d는 데이터베이스 테이블에서의 레코드의 실행 "잔고(balance)", "집계(tally)", 또는 "상태(state)"의 정확도를 보장하기 위해 이벤트들(예를 들어, 트랜잭션들)의 정확한 순서화를 결정하는 것으로서 합의의 실시하고 설명하지만, 대안적으로 또는 추가적으로, 합의는, 자산의 현재 소유자를 결정하는 것, 데이터베이스 테이블 내의 레코드들의 정확한 순서화를 결정하는 것, 상품의 체인 오브 커스터디(chain-of-custody) 또는 체인 오브 타이틀(chain-of-title) 타이틀을 결정하는 것 등 중에서의 하나 이상의 목적을 위해 이벤트들의 정확한 순서화를 결정하기 위해 이용될 수 있다.

[0059] 본 명세서에 개시된 일부 실시예들에서, 장치(예를 들어, 도 1의 컴퓨팅 디바이스(110) 및/또는 컴퓨팅 디바이스(들)(120))는 프로세서, 및 프로세서와 전자 통신하는 메모리를 포함한다. 메모리(예를 들어, 도 1의 메모리(112) 및/또는 메모리(122))는 다수의 레코드들을 포함하는 데이터베이스 테이블을 저장하고, 다수의 레코드들로부터의 각각의 레코드는 버전 식별자를 갖는 복합 키를 포함한다. 메모리는 프로세서로 하여금, 데이터베이스 테이블의 제1 가상 클론 및 데이터베이스 테이블의 제2 가상 클론을 생성하게 하는 실행가능한 명령어들을 포함한다. 메모리는 또한 프로세서로 하여금, 제1 가상 클론에서 질의를 수신하고, 제1 가상 클론에서 질의를 수신하는 것에 기초하여, 제1 가상 클론과 연관된 버전 식별자를 포함하는 수정된 질의를 정의하기 위해 질의를 수정하도록 실행가능한 명령어들을 포함한다. 메모리는 또한 프로세서로 하여금, 수정된 질의를 이용하여 데이터베이스 테이블에 질의하여, 다수의 레코드들로부터의 레코드들의 세트를 포함하는 응답을, (1) 질의를 충족시키는 레코드들의 세트로부터의 각각의 레코드, 및 (2) 제1 가상 클론과 연관된 버전 식별자의 조상 경로와 연관된 버전 식별자를 갖는 레코드들의 세트로부터의 각각의 레코드에 기초하여, 생성하게 하도록 실행가능한 명령어들을 포함한다. 메모리는 프로세서로 하여금, 레코드들의 세트를 포함하는 응답을 질의에 대한 응답으로서 전송하게 하도록 실행가능한 명령어들을 또한 포함한다. 일부 그러한 실시예들에서, 다수의 레코드들로부터의 각각의 레코드에 대한 복합 키는 그 복합 키에 대한 버전 식별자 및 데이터 부분을 포함한다. 다수의 레코드들은, (1) 데이터 부분의 값 및 제1 가상 클론과 연관된 버전 식별자를 포함하는 복합 키; 또는 (2) 데이터 부분의 값 및 제2 가상 클론과 연관된 버전 식별자를 포함하는 복합 키 중 하나만이 임의의 주어진 시간에 데이터베이스 테이블에 존재하도록 구성될 수 있다.

[0060] 일부 실시예들에서, 메모리는 프로세서로 하여금, 제1 가상 클론을 통해 수신된 삽입 명령어에 응답하여, 데이터베이스 테이블에 레코드를 추가하게 하도록 실행가능한 명령어들을 또한 포함한다. 레코드는, (1) 제1 가상 클론에서 후속하여 수신되고 데이터를 참조하는 질의가 레코드를 반환하고, (2) 데이터에 대해 제3 가상 클론에서 후속하여 수신되는 질의가 레코드를 반환하지 않도록, 제1 가상 클론과 연관된 버전 식별자를 갖는 복합 키 및 데이터를 포함한다.

[0061] 일부 실시예들에서, 메모리는 프로세서로 하여금, 데이터베이스 테이블의 제1 레코드의 필드의 값을 제1 값으로부터 제2 값으로 업데이트하라는, 제3 가상 클론을 통해 수신된 명령어에 응답하여, 제2 레코드를 데이터베이스 테이블에 추가하게 하도록 실행가능한 명령어들을 또한 포함한다. 제2 레코드는 제2 값 및 제3 가상 클론과 연관된 버전 식별자를 갖는 복합 키를 포함한다. 명령어들은 제1 레코드의 필드의 값을 업데이트하라는 명령어에

응답하여, 제1 레코드의 복합 키를, 제3 가상 클론에서 후속하여 수신되고 필드를 참조하는 질의가 제2 레코드를 반환하고, 제1 가상 클론에서 수신되고 필드를 참조하는 질의가 제1 레코드를 반환하도록, 제3 가상 클론과 연관된 버전 식별자를 포함하지 않도록 수정하게 하도록 더 실행가능할 수 있다.

[0062] 일부 실시예들에서, 메모리는 프로세서로 하여금, 다수의 레코드들로부터 레코드를 삭제하라는, 제3 가상 클론을 통해 수신된 명령어에 응답하여, 레코드의 복합 키의 버전 식별자를, 제3 가상 클론에서 후속하여 수신된, 레코드에 대한 질의가 레코드를 반환하지 않고, 제1 가상 클론에서 후속하여 수신된, 레코드에 대한 질의가 레코드를 반환하도록, 제3 가상 클론과 연관된 버전 식별자를 포함하지 않도록 수정하게 하게 하도록 실행가능한 명령어들을 또한 포함한다.

[0063] 일부 실시예들에서, 응답은 제1 응답이고, 수정된 질의는 제1 수정된 질의이다. 명령어들은 프로세서로 하여금, 제1 응답이 널인 경우 제2 가상 클론의 버전 식별자를 포함하는 제2 수정된 질의를 생성하게 하도록 더 실행가능할 수 있고, 제2 가상 클론은 제1 가상 클론에 대한 조상이다. 명령어들은 프로세서로 하여금, 제2 수정된 질의를 이용하여 데이터베이스 테이블에 질의하여, 다수의 레코드들로부터의 레코드들의 세트를 포함하는 제2 응답을, (1) 제2 수정된 질의를 충족시키는 제2 응답의 레코드들의 세트로부터의 각각의 레코드, 및 (2) 제2 가상 클론의 버전 식별자의 조상 경로와 연관된 버전 식별자를 갖는 제2 응답의 레코드들의 세트로부터의 각각의 레코드에 기초하여, 생성하게 하고, 레코드들의 세트를 포함하는 제2 응답을 질의에 대한 응답으로서 전송하게 하도록 더 실행가능할 수 있다.

[0064] 일부 실시예들에서, 명령어들은 프로세서로 하여금, 제1 가상 클론이 제2 가상 클론의 자손이라는 표시를 메모리에 저장하게 하도록 더 실행가능하다.

[0065] 일부 실시예들에서, 명령어들은 프로세서로 하여금, 데이터베이스 테이블의 제3 가상 클론을 생성하게 하도록 더 실행가능하고, 제3 가상 클론은 제1 가상 클론의 조상도 자손도 아니며, 다수의 레코드들은 데이터 부분의 값 및 제3 가상 클론과 연관된 버전 식별자를 포함하는 복합 키를 갖는 레코드를 포함한다.

[0066] 일부 실시예들에서, 장치는 프로세서, 및 프로세서와 전자 통신하는 메모리를 포함한다. 메모리는 데이터베이스 테이블 및 프로세서 실행가능 명령어들을 저장한다. 프로세서 실행가능 명령어들은 프로세서로 하여금, 데이터베이스 테이블의 다수의 가상 클론들의 표현들을 포함하는 계층적 트리를 생성하게 하는 명령어들을 포함하고, 다수의 가상 클론들은 제1 가상 클론(예를 들어, 도 5의 노드 (2, 2)), 제1 가상 클론의 자식 가상 클론인 제2 가상 클론(도 5의 노드 (4, 4)), 및 제2 가상 클론의 후속자 가상 클론인 제3 가상 클론(예를 들어, 도 5의 노드 (5, 6))을 포함한다. 프로세서 실행가능 명령어들은 프로세서로 하여금, 제2 가상 클론을 삭제하라는 명령어에 응답하여, 제3 가상 클론과 연관된 후속자 관계를 수정하게 하는 명령어들을 또한 포함한다. 프로세서 실행가능 명령어들은 프로세서로 하여금, 제2 가상 클론을 삭제하라는 명령어에 응답하여, 제2 가상 클론을 삭제하게 하는 명령어들을 또한 포함한다. 프로세서 실행가능 명령어들은 프로세서로 하여금, 제2 가상 클론을 삭제하라는 명령어에 응답하여, 제1 가상 클론이 제2 가상 클론의 삭제 후에 자식 가상 클론을 갖지 않을 때, 다수의 가상 클론들로부터 제1 가상 클론을 삭제하게 하는 명령어들을 또한 포함한다.

[0067] 일부 실시예들에서, 다수의 가상 클론은 제1 가상 클론의 후속자 가상 클론인 제4 가상 클론을 더 포함한다. 명령어들은 프로세서로 하여금, 제2 가상 클론을 삭제하라는 명령어; 및 제4 가상 클론이 제2 가상 클론의 삭제 후에 제1 가상 클론의 유일한 자손이라는 결정에 응답하여, 제4 가상 클론과 연관된 후속자 관계를 수정하게 하도록 더 실행가능하다.

[0068] 일부 실시예들에서, 명령어들은 프로세서로 하여금, 다수의 가상 클론들로부터의, 자식 가상 클론을 갖지 않는 제4 가상 클론을 자동으로 식별하고, 자동 식별에 기초하여 제4 가상 클론을 삭제하게 하도록 더 실행가능하다. 대안적으로, 명령어들은 프로세서로 하여금, 다수의 가상 클론들로부터의, 자식 가상 클론을 갖지 않는 제4 가상 클론을 자동으로 식별하고, 자동으로 식별하는 것에 응답하여, (1) 다수의 가상 클론들로부터의 제5 가상 클론과 연관된 후속자 관계를 수정하고, (2) 제4 가상 클론을 삭제하게 하도록 더 실행가능할 수 있다.

[0069] 일부 실시예들에서, 명령어들은 프로세서로 하여금, 제3 가상 클론과 연관된 후속자 관계를 수정하는 것에 응답하여, 다수의 가상 클론들로부터의 적어도 하나의 추가 가상 클론의 삭제를 트리거하게 하도록 더 실행가능하다.

[0070] 일부 실시예들에서, 후속자 관계는 제1 후속자 관계이고, 메모리는 제1 후속자 관계를 포함하는 다수의 후속자 관계들을 더 저장한다.

[0071] 일부 실시예들에서, 데이터베이스 테이블은 다수의 레코드들을 포함하고, 다수의 레코드들로부터의 각각의 레코

드는 복수의 가상 클론들로부터의 가상 클론과 연관된 버전 식별자를 포함하는 복합 키를 포함한다.

- [0072] 일부 실시예들에서, 장치는 프로세서, 및 프로세서와 전자 통신하는 메모리를 포함한다. 메모리는 데이터베이스 테이블, 및 프로세서로 하여금 데이터베이스 테이블의 다수의 가상 클론들의 표현들을 포함하는 계층적 트리를 생성하게 하도록 실행가능한 명령어들을 저장한다. 명령어들은 다수의 가상 클론들로부터 제1 가상 클론(예를 들어, 도 5의 노드 (4, 4))을 삭제하라는 명령어에 응답하여, 제2 가상 클론(예를 들어, 도 5의 노드 (5, 6))과 연관된 후속자 관계를 수정—제2 가상 클론은 제1 가상 클론의 후속자 가상 클론임—하고, 제1 가상 클론을 삭제하고, 제4 가상 클론(예를 들어, 도 5의 노드 (3, 3))과 연관된 후속자 관계를 수정—제4 가상 클론은 제3 가상 클론(예를 들어, 도 5의 노드 (2, 2))의 후속자 가상 클론이고, 제3 가상 클론은 제1 가상 클론의 조상 가상 클론임—하고, 제3 가상 클론을 삭제하게 하도록 또한 실행가능할 수 있다.
- [0073] 일부 실시예들에서, 명령어들은 프로세서로 하여금, 제2 가상 클론이 유일한 자손 가상 클론이고 제4 가상 클론의 후속자 가상 클론인 것을 검출하고, 제2 가상 클론이 유일한 자손 가상 클론이고 제4 가상 클론의 후속자 가상 클론인 것을 검출하는 것에 응답하여 제2 가상 클론을 삭제하게 하도록 더 실행가능하다.
- [0074] 일부 실시예들에서, 데이터베이스 테이블은 다수의 레코드들을 포함하고, 다수의 레코드들로부터의 각각의 레코드는 다수의 가상 클론들로부터의 가상 클론과 연관된 버전 식별자를 포함하는 복합 키를 포함한다.
- [0075] 일부 실시예들에서, 명령어들은 프로세서로 하여금, 제1 가상 클론을 삭제하라는 명령어에 응답하여, 다수의 가상 클론들로부터의 적어도 하나의 추가 가상 클론의 삭제를 트리거하게 하도록 더 실행가능하다.
- [0076] 일부 실시예들에서, 명령어들은 프로세서로 하여금, 다수의 가상 클론들로부터의, 자손 가상 클론을 갖지 않는 제5 가상 클론을 자동으로 식별하게 하도록 더 실행가능하다. 명령어들은 자동으로 식별하는 것에 응답하여, 제6 가상 클론과 연관된 후속자 관계를 수정하고, 제5 가상 클론을 삭제하도록 또한 실행가능하다.
- [0077] 일부 실시예들에서, 메모리는 제2 가상 클론과 연관된 후속자 관계 및 제4 가상 클론과 연관된 후속자 관계를 포함하는 다수의 후속자 관계들을 더 저장한다.
- [0078] 일부 구현들에서, 충돌 제약은 (1001, N)과 같은 주어진 복합 키를 갖는 레코드가 테이블에 존재하는 경우, 테이블이 (1001, M)의 복합 키를 갖는 임의의 레코드를 포함하지 않을 것임을 보장하며, 여기서 M은 버전 트리에서 N의 조상인 버전 번호이다. 예를 들어, 버전 트리가 도 8c에 도시된 바와 같은 경우, 그리고 키 (1001, 4)를 갖는 레코드가 존재하는 경우, 키 (1001, V3) 또는 키 (1001, V1)을 갖는 어떠한 레코드도 없는 것으로 보장된다. 그러나, V2는 트리에서 V4의 조상이 아니기 때문에, 키 (1001, 2)를 갖는 레코드가 존재할 수 있다.
- [0079] 일부 구현들에서, 데이터베이스 테이블의 각각의 레코드/행은 처음에 1의 버전 ID를 포함한다. 그 다음, 데이터베이스 테이블은 본 명세서에 설명된 바와 같이, 고속 카피, 추가, 삭제, 또는 수정을 수행하는 동작들의 결과로서 성장 및 축소될 수 있다.
- [0080] 일부 구현들에서, 삭제 동작은 다음과 같이 수행될 수 있다. 가상 클론(또는 "뷰")을 통해 수신된 SQL 질의는 삭제를 위한 하나 이상의 레코드를 참조할 수 있다. 각각의 그러한 레코드는 다음과 같이 삭제될 수 있다: 레코드가 버전 N을 갖는다면, 그 레코드가 삭제된다. 다음, 뷰로부터 계층적 트리의 "루트(root)"(즉, 데이터베이스 테이블 인스턴스)로의 직접적인 조상 경로를 따른 각각의 가상 클론에 대해, 직접적인 조상 경로 상에 있지 않은 그 가상 클론들의 자식들이 있는 경우, 그 버전이 M으로 변경된 삭제된 레코드의 카피가 그 자식들의 데이터베이스 테이블에 추가된다. 예를 들어, 도 4b의 계층적 트리가 주어지는 경우, 키 (Pie, V2)를 갖는 레코드가 V2를 통해 삭제되면, 레코드는 테이블로부터 삭제된다. 키 (Pie, V1)을 갖는 레코드가 V2를 통해 삭제되면, 레코드는 삭제되고, V3이 V1의 자식이고 데이터베이스(230)로부터 V2로의 직접적인 조상 경로 상에 있지 않기 때문에, 버전 V3을 갖는 레코드의 카피가 테이블에 추가된다. 이러한 삭제 동작은, V2를 통한 후속 판독이 가상 기본 키 Pie를 갖는 임의의 레코드를 반환하지 않을 것이고, V3을 통한 후속 판독들이 그 결과로부터 버전 번호가 제거되면 삭제 전과 동일한 결과를 반환할 것임을 보장한다는 점에 유의한다. 또한, 그러한 삭제 동작들이 충돌 제약을 충족시키는 테이블에 대해 수행될 때, 수정된 테이블은 충돌 제약을 계속 충족시킬 것이다.
- [0081] 일부 구현들에서, 삭제 명령어가 하나의 레코드의 삭제 및 상이한 버전들을 갖는 K개의 새로운 레코드들의 추가를 지시하면(K>0임), 이것은 레코드의 버전 번호를 새로운 버전 번호들 중 하나로 변경함으로써 구현될 수 있고, 이후 K-1개의 카피들(각각 상이한 버전 번호를 가짐)을 만들 수 있다. 이러한 최적화는 하나의 추가를 절약하는데, 이는 수정들을 실행하는 것이 기본 데이터베이스에 대한 추가들을 실행하는 것보다 빠른 경우에 유

익할 수 있다(예를 들어, 데이터베이스의 효율을 증가시킬 수 있다).

- [0082] 다른 구현들에서, "삭제된" 부울 속성(Boolean attribute)이 각각의 레코드에 추가될 수 있다. 예로서, 도 4b의 계층적 트리가 주어지면, V2를 통해 삭제된 레코드가 버전 V2를 갖는 경우, 해당 레코드에 대한 삭제된 속성은 거짓으로부터 참으로 변경될 수 있다. 그러나, V2를 통해 삭제된 레코드가 버전 1을 갖는 경우, 프로세서는 버전 V2 및 참으로 설정된 삭제된 속성을 갖는 레코드의 새로운 카피를 생성 및/또는 정의할 수 있다. 테이블이 충돌 제약을 충족시킬 필요가 없다면, 버전 V1을 갖는 레코드는 거짓으로 설정된 그 삭제된 속성으로 유지될 수 있다. 그러나, 테이블이 충돌 제약을 충족시킬 필요가 있다면, 버전 V3 및 거짓으로 설정된 삭제된 속성을 갖는 레코드의 다른 카피가 생성 및/또는 정의될 수 있고, 버전 V1을 갖는 레코드가 테이블로부터 제거될 수 있다.
- [0083] 일부 구현들에서, 삽입 동작(즉, 레코드를 데이터베이스 테이블 내로 삽입하는 것)은 먼저, 삽입될 레코드의 가상 기본 키와 매칭되는 레코드의 삭제를 수행하고, 그 다음에 삽입 동작을 트리거하는 삽입 명령어가 수신되도록 하는 가상 클론의 부모인 버전 번호를 갖는 새로운 레코드를 삽입함으로써 수행된다. 예를 들어, 도 4b의 계층적 트리가 주어지면, (예를 들어, "Blackberry"의 가상 기본 키 값을 갖는 새로운 레코드의 삽입을 지시하는) 삽입 명령어를 포함하는 SQL 질의가 V2를 통해 수신되는 경우, 질의는 V2를 통해 액세스가능한, 가상 기본 키 Blackberry를 갖는 임의의 레코드(즉, V2의 조상 경로에서의 가상 클론에 대응하는 버전 속성에 대한 값, 즉, V2 또는 V1을 갖는 임의의 레코드)를 먼저 찾아서 삭제하고, 이어서 복합 키 (Blackberry, V2)를 갖는 새로운 레코드를 추가하도록 수정될 수 있다. 이러한 추가 동작은 충돌 제약 준수를 지킨다는 점에 유의한다.
- [0084] 일부 구현들에서, 먼저 그 가상 기본 키를 이용하여 데이터베이스 테이블의 판독(또는 질의)을 수행하는 것에 의해 삽입 동작(즉, 레코드를 데이터베이스 테이블 내로 삽입하라는 요청에 응답한 프로세스)이 수행된다. 판독 동안에, 삽입될 레코드와 매칭되는 결과가 반환되는 경우, 레코드를 삽입하라는 요청에 응답하여 에러 메시지가 반환될 수 있다. 한편, 삽입될 레코드와 매칭되는 결과가 반환되지 않는 경우, 요청을 수신한 가상 클론의 부모인 버전 번호를 갖는 레코드가 데이터베이스 테이블 내로 삽입된다. 예를 들어, 도 4b의 계층적 트리가 주어지면, V2를 통한 SQL 삽입 동작이 가상 기본 키 Blackberry를 갖는 레코드를 삽입하는 경우, 이러한 삽입 동작은 먼저 전술한 판독 동작을 이용하여, 가상 기본 키 Blackberry 및 V2 또는 V1의 버전 속성에 대한 값을 갖는 임의의 레코드에 대한 판독을 수행하도록 수정된다. 질의가 판독에 대한 임의의 결과들을 반환하면, 프로세서는 삽입 동작에 응답하여 에러 메시지를 반환하는데, 이것은 공통 조상 경로와 연관된 중복 레코드를 추가하려는 시도이기 때문이다. 질의가 판독에 대한 결과를 반환하지 않으면, 프로세서는 복합 키 (Blackberry, V2)를 갖는 레코드를 추가한다. 이러한 삽입 동작은 충돌 제약 준수를 지킨다는 점에 유의한다.
- [0085] 일부 구현들에서, 레코드 X에 대한 수정 동작은 전술한 바와 같이, X의 이전 버전(old version)을 먼저 삭제한 다음, 전술한 바와 같이 X의 수정된 버전을 추가/삽입함으로써 구현될 수 있다. 삭제 및 삽입 모두가 충돌 제약 준수를 지키기 때문에, 수정 동작 또한 충돌 제약 준수를 지킨다.
- [0086] 일부 구현들에서, 도 4b의 계층적 트리와 같은 버전 트리는 데이터베이스에 추가되거나 그와 연관되는 별도의 테이블에서 기술될 수 있다. 고속 카피 데이터베이스 시스템은 또한, 예를 들어, 질의들이 신속하게 번역될 수 있도록, 메모리에서의 버전 트리 정보를 캐싱할 수 있다.
- [0087] 버전 트리가 최하부에 작은 트리를 갖는 긴 체인인 경우, 버전 비교들이 단순화될 수 있다. 예를 들어, 버전을 수용가능 버전들의 긴 리스트와 비교하는 대신에, 예를 들어, 모두 80보다 작은 버전 번호들의 긴 리스트가 아니라, "버전 < 80"을 명시함으로써, 그 리스트의 서브셋이 비교를 위해 이용될 수 있다. 대안적으로, 예를 들어, "버전=11 또는 버전=12 또는 버전=13 또는 버전=14"를 명시적으로 검색하는 대신에, 선택 "10 < 버전 및 버전 < 15"와 같은 범위들이 이용될 수 있다. 전술한 것은, 예를 들어, 원래의 SQL 질의를 나타내는 문자열(string)을 파싱(parsing)하고, 결과적인 파싱 트리를 수정하고, 이어서 문자열을 실제 데이터베이스로 패스하기 전에 문자열로 다시 변환함으로써 수행될 수 있다.
- [0088] 대안적으로, 다른 경우들에서, 기본 데이터베이스는 뷰들 및 저장된 절차들을 지원할 수 있다. 만약 그렇다면, 데이터베이스의 고속 카피가 생성될 때, 각 테이블에 대한 뷰가 정의될 수 있으며, 이는 테이블의 해당 버전으로부터의 선택들이 위에서 정의된 바와 같이 수행될 것임을 보장한다. 저장된 절차는 뷰에 대한 기입들이 전술한 레코드 카피(record copying)로 변환되도록 생성될 수 있다.
- [0089] 일부 구현들에서, (예를 들어, 자바(Java)로부터 액세스되는) 고속 카피가능 관계형 데이터베이스들을 위한 시스템은 SQL 질의들을 고속 카피가능 데이터베이스 플랫폼(database platform)(예를 들어, 자바로 또한 기입됨)

에 전송하는 사용자 프로그램(예를 들어, 자바로 기입됨)을 포함하고, 고속 카피가능 데이터베이스 플랫폼은 그것들을 실제 데이터베이스 상으로 전송하기 전에 그것들을 번역한다. 예를 들어, 관계형 데이터베이스가 "사람들(people)"이라고 명명된 다음의 테이블을 갖는다고 가정한다:

사람들:

id	이름	나이	주
123	Alice	20	AZ
456	Bob	30	OH
789	Carol	40	CO

[0090]

[0091]

가상 기본 키는 id_이고, 3개의 다른 속성들: 이름(name), 나이(age), 및 주(state)가 있다. (예를 들어, 프로세서를 통해 고속 카피가능 데이터베이스 프로세스를 실행하는) 고속 카피가능 데이터베이스 시스템은, 하나의 숨겨진 속성인 버전_을 추가하고, 다른 속성들 및 테이블 이름에 밑줄 표시(underscore)를 추가함으로써, 이것을 고속 카피가능 테이블로 변환할 수 있다.

사람들_:

id_	이름_	나이_	주_	버전_
123	Alice	20	AZ	1
456	Bob	30	OH	1
789	Carol	40	CO	1

[0092]

[0093]

이 새로운 테이블은 복합 키 (id_, 버전_)을 갖고, id_는 전술한 바와 같이 가상 기본 키라고 지칭될 수 있다. 따라서, 2개의 레코드가, 그들의 버전_이 상이한 한, 동일한 id_를 갖는 것이 이제 가능하다. 상기의 예는 충돌 제약을 준수하고, 이 예를 갖는 동작들은 그러한 준수를 지킬 수 있다. 일부 다른 테이블이 id의 외래키(foreign key)를 갖는 경우, 외래키가 id 대신에 id_로 명명된 속성과 매칭될 것이라는 점을 제외하고는, 외래키가 변하지 않은 채로 유지될 것이다. 원래, 외래키는 가상 기본 키 사람들.id에 링크되고, 이것은 외래키가 사람들 테이블에서의 기껏해야 단일 레코드에 링크됨을 의미한다. 그러나 이제, 외래키가 질의에서 이용될 때, 외래키는 여러 레코드들의 id_와 매칭될 수 있고, 따라서 외래키는 질의를 수행하기 위해 이용되는 뷰의 조상인 버전을 갖는 그 레코드들의 세트로부터의 단일 레코드에 링크하는 것으로서 해석될 것이다. 충돌 제약을 준수하는 경우들에서, 2개의 그러한 레코드들이 존재할 수 있는데, 그 이유는 이것이 충돌 제약을 위반할 것이기 때문이다. 그와 같이, 외래키들의 링크는 고속 카피가능 데이터베이스와 호환가능할 수 있다.

[0094]

일부 구현들에서, 사용자에게 의해 생성된 테이블들 및 속성들의 이름들에 대한 제한이 시행될 수 있다. 예를 들어, 이름들 및/또는 속성들이 기껏해야 기본 데이터베이스의 전형적인 테이블/속성 크기보다 1 문자 짧은 길이(예를 들어, 80 문자들 대신 79 문자들의 최대 크기, 또는 64 문자들 대신 63 문자들의 최대 크기)를 갖는다는 것을 명시하는 규칙이 구현될 수 있다. 대안적으로 또는 추가적으로, 이름들 및/또는 속성들이 밑줄 표시로 종료될 수 없다는 것을 명시하는 규칙이 구현될 수 있다. 이것은 테이블들 및 속성들이 재명명될 수 있게 한다. 이것은 또한 특정 구현을 위해 유용한 경우, 원래의 테이블 이름이 저장된 절차들을 갖는 뷰로서 이용되는 것을 허용한다.

[0095]

이제, 예를 들어, 시스템(또는 프로세서)이 버전 1에 대해 고속 카피 동작을 수행하여, (아래에 도시된 바와 같이) 버전들 2 및 3을 생성하는 것으로 가정한다. (프로세서에 의해 실행되는) 사용자 코드는 판독들 및 기입들 둘다를 위해 버전 번호에 각각의 SQL 질의를 제공할 수 있고, 그 버전 번호는 버전 트리에서의 리프의 버전일 것이다. 이 시점에서, (프로세서에 의해 실행되는) 사용자 코드는 더 이상 버전 1을 직접 판독 또는 기입하지 않을 수 있는데, 그 이유는 판독들 및 기입들이 버전 트리의 리프에서 행해지기 때문이다. 시스템은 이제 아래와 같이 보이며, 여기서 동작들은 리프 가상 클론들(원들)을 통해 발생하는 것으로 설명된다.

사람들_:

id_	이름_	나이_	주_	버전_
123	Alice	20	AZ	1
456	Bob	30	OH	1
789	Carol	40	CO	1



[0096]

[0097]

고속 카피 동작은 테이블을 전혀 변경하지 않았다. 그것은 단지 버전 트리를 변경하였으며, 이것은 우측에 도시되어 있고, 특별한 테이블에 저장되며, 일부 구현들에서는, 여기서는 도시되지 않지만, 2개의 속성, 즉, 부모 및 자식을 포함하는 (예를 들어, 끝에 2개의 밑줄 표시들을 갖는) 버전들__로 명명될 수 있다.

[0098]

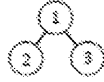
일부 구현들에서, 사용자 프로그램은 데이터베이스의 하나의 특정 버전에 전송될 SQL 질의인 문자열을 생성한다. 시스템은 그 문자열을 파싱하고, 결과적인 파싱 트리를 수정하고, 이후 그것을 다시 문자열로 변환

하고, 그 수정된 문자열을 실제 데이터베이스에 SQL 질의로서 전송할 것이다. 그와 같이, 사용자의 관점에서, 그것들이 실제로 고속 카피들을 나타내는 단일의 결합된 데이터베이스에 판독/기입할 때, 그것들이 다수의 데이터베이스 카피들 중 하나에 대해 판독/기입하는 것처럼 나타난다.

(예를 들어, 프로세서를 이용하는) 사용자가 가상 클론 3을 통해 Dave에 대한 새로운 레코드를 추가하는 것으로 가정한다. 이것은 다음의 데이터베이스 테이블을 초래할 것이다:

사람들:

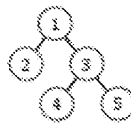
id	이름	나이	주	버전
123	Alice	20	AZ	1
456	Bob	30	OH	1
789	Carol	40	CO	1
890	Dave	50	DE	3



이제, 버전 2를 통한 질의들은 Dave에 대한 새롭게 생성된 레코드를 무시할 것이지만(즉, 검색하지 않을 것이지만), 버전 3을 통한 질의들은 Dave를 참조할 것이다. (예를 들어, 프로세서를 이용하는) 사용자가 이제 버전 3의 고속 카피를 수행하여, 다음의 데이터베이스 테이블을 제공한다고 가정한다:

사람들:

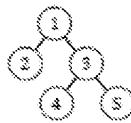
id	이름	나이	주	버전
123	Alice	20	AZ	1
456	Bob	30	OH	1
789	Carol	40	CO	1
890	Dave	50	DE	3



Dave 레코드는 3의 속성 "버전"에 대한 값을 포함하고, 따라서 가상 클론들 4 및 5를 통해 행해진 질의들에 의해 반환될 것이지만, 가상 클론 2에 대한 질의들에 의해서는 아니다. 그 다음, (예를 들어, 프로세서를 이용하는) 사용자가 가상 클론 2를 통해 Bob을 Bobbie로 변경하는 것으로 가정한다. 결과적인 데이터베이스 테이블은 다음과 같다:

사람들:

id	이름	나이	주	버전
123	Alice	20	AZ	1
456	Bob	30	OH	3
789	Carol	40	CO	1
890	Dave	50	DE	3
456	Bobbie	30	OH	2

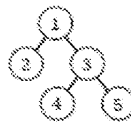


이러한 "변경"은 삭제 및 그 이후의 2개의 추가로 구현된다는 점에 유의한다. 원래의 레코드(Bob, 1)가 삭제되고, 3의 버전을 갖는 레코드의 카피가 추가되는데, 그 이유는 3은 2로부터 1까지의 경로 상의 노드의 자식이지만, 그 자체는 경로 상에 없기 때문이다. 그 후, 레코드가 가상 클론 2를 통해 수정되고 있기 때문에, 2의 버전을 갖는 변경을 갖는 새로운 레코드가 추가된다.

이제 id=456을 갖는 2개의 레코드가 있다. 이전 레코드의 카피는 이제 버전 3이고, 따라서 그것이 동작 전에 있었던 것처럼 여전히 4 및 5로 보여진다. 새로운 레코드는 버전 2이고, 버전 2를 통해 변경이 행해졌기 때문에, 가상 클론 2를 통해서만 보여진다. 이제, (예를 들어, 프로세서를 이용하는) 사용자가 가상 클론 4를 통해 Carol을 삭제하는 것으로 가정한다. 결과적인 데이터베이스 테이블은 다음과 같다:

사람들:

id	이름	나이	주	버전
123	Alice	20	AZ	1
456	Bob	30	OH	1
789	Carol	40	CO	2
890	Dave	50	DE	3
456	Bobbie	30	OH	2
789	Carol	40	CO	5



이것은 원래의 (Carol, 1) 레코드를 삭제한 다음, 버전들 2 및 5를 갖는 레코드의 2개의 카피들을 추가하였는데, 그 이유는 2 및 5가 4로부터 1까지의 경로 상의 노드들의 자식들이지만, 그들 자체는 그 경로 상에 없기 때문이다.

요약하면, 일부 경우들에서, 가상 클론 V를 통해 수행되는 동작들은 다음과 같이 수행된다:

- [0110] · 레코드 삭제: 버전 N을 갖는 원래의 레코드를 삭제하지만, 그 자신이 그 경로 상의 있지 않은 가상 클론 V로부터 가상 클론 N까지의 경로 상의 노드의 모든 자식에 대해 원래의 레코드의 카피들을 삽입한다.
- [0111] · 레코드 추가: 중복 레코드들이 동일한 조상 경로와 연관되는 것을 피하기 위해, 추가될 레코드에 공통이고 동일한 조상 경로 내에 있는 적어도 하나의 값(예를 들어, 가상 기본 키)을 포함하는 이전의 레코드가 먼저 삭제될 수 있고, 이어서 버전 V를 갖는 레코드의 추가가 뒤따를 수 있다.
- [0112] · 레코드 추가(선택적인 변형): 먼저 판독이 그 가상 기본 키를 갖는 임의의 레코드를 반환하지 않는 것을 보장하고, 이후 그것이 그렇지 않으면 버전 V를 갖는 레코드를 추가할 에러를 반환한다.
- [0113] · 레코드 수정: 이전 레코드를 삭제한 다음, 레코드의 수정된 버전을 추가한다.
- [0114] · 질의들, 조인들(joins) 등: 조인들과 같은 외래키들을 포함하는 연산들에 대해, 그것들을 정상적으로 수행하지만, 이들 레코드들이 동일한 조상 경로 내에 있을 버전 번호들을 갖는 2개의 테이블들로부터의 레코드들만을 조인하고, 따라서 그들은 가상 클론 V를 통해 액세스가능하다.
- [0115] · 테이블 생성/삭제: 사용자가 테이블의 법적 이름(마지막 문자가 밑줄 표시가 아닌, 충분히 짧은 것)을 선택하고, 이어서 테이블 이름에 밑줄 표시를 추가하도록 보장한다. version_ 속성을 추가한다.
- [0116] · 속성 생성/삭제: 속성 이름이 충분히 짧고 밑줄 표시로 끝나지 않는 것을 보장하고, 이어서 밑줄 표시를 추가한다.
- [0117] 데이터베이스 테이블의 질의들은 버전 번호를 포함한다. 일부 경우들에서, 가상 클론을 고속 카피하는 (예를 들어, 프로세서에 의해 실행되는) 방법이 존재한다. 일부 경우들에서, 가상 클론을 삭제하는 (예를 들어, 프로세서에 의해 실행되는) 방법이 또한 존재한다. 일부 구현들에서, 사용자는 그것이 리프인 경우에만 가상 클론을 삭제할 수 있다. 가상 클론의 삭제가 어떠한 자식들도 없는 내부 가상 클론을 남기면, 그 가상 클론 또한 삭제될 수 있다. 그러한 경우에, 비-리프(non-leaf) 가상 클론은 리프가 되지 않는다. 그것은 그의 자식들이 삭제되자마자 사라진다. 시간이 지남에 따라, 이것은 트리가 루트로부터 깊은 레벨로 내려가는 단일 자식 가상 클론들의 체인으로서 나타나게 할 수 있고, 그 후에 단지 몇 개의 분기들 및 몇 개의 노드들만을 갖는 작은 트리가 존재한다. 이들 단일 자식 가상 클론들은 기여하는 것 없이, 공간 및 시간을 낭비할 수 있다.
- [0118] 일부 경우들에서, 주기적으로(또는 산발적으로), 시스템은 그러한 단일 자식 버전들을 제거하기 위해 (예를 들어, 프로세서에 의해 실행되는) 압축 동작을 실행할 수 있다. 예를 들어, 가상 클론 42가 가상 클론 99인 단일 자식만을 갖는다면, 이후 (예를 들어, 프로세서에 의해 실행되는) 동작이 테이블들에 대해 수행될 수 있고, 42의 각각의 버전__ 속성 값을 99로 변경하고, 트리에서의 2개의 가상 클론들을 수반하는 시퀀스 업데이트가 수행될 수 있고, 결과는 버전 99를 갖는 단일 가상 클론으로서 표현된다.
- [0119] 데이터베이스가 몇 개의 가상 클론들에 의해 액세스될 수 있지만, 예를 들어, 하나의 가상 클론의 질의만이 한 번에 처리되고/되거나 압축 동작들 동안 질의들이 처리되지 않는다는 것을 명시하는 단일 규칙으로 전체 데이터베이스를 보호하는 것이 수용가능하다. (예를 들어, 프로세서에 의해) 압축 동작이 수행되기 전에, 질의들은 트리 내의 긴 체인들을 반영하도록 최적화될 수 있다. 예를 들어, 버전 V=99에 대해 질의가 수행되고, 버전 V=99의 조상들이 버전들 {1, 10, 12, 20, 23, 32, 42}인 경우, 질의는 그 자신을 그 7개의 버전들 중 임의의 버전을 갖는 레코드들로 제한하도록 (예를 들면, 프로세서에 의해) 수정될 수 있다. 그러나, 23 미만의 번호들을 갖는 트리에 다른 정점들(vertices)이 없다면, 질의는 {≤ 23, 32, 42}를 요청하기 위해 (예를 들어, 프로세서에 의해) 단순화될 수 있다. 즉, 버전 42 또는 버전 32 또는 23 이하의 임의의 버전 번호를 갖는 레코드들을 요청할 것이다. 유사하게, 버전 트리에서의 적용가능한 정점들은 {1, 10, 12, 20, 23, 32, 42}이지만, 10과 32 사이의 트리에서의 다른 정점들이 존재하지 않으면, 질의는 {1, 10...32, 42}에 대한 것일 수 있다. 즉, 이것은 1과 동일한, 42와 동일한, 또는 10 내지 32(경계 포함)의 임의의 번호인 버전들과 매칭될 것이다. 이러한 최적화들은 많은 버전들이 생성 및 삭제될 때 특히 유용하다.
- [0120] 전체 SQL 언어는 복잡하고, 악의적인 사용자가 질의하고 있는 가상 클론을 통해 액세스가능한 것을 넘어서는 정보에 액세스하기 위해 이용할 수 있는 많은 강력한 명령어들이 있다. 일부 구현들에서, 이러한 취약성(vulnerability)은, 전체 SQL의 작은 서브세트만이 사용자에게 의해 이용되도록 하고, 그 서브세트 밖의 어떤 것이 사용자에게 의해 수신될 때에 노드가 예외를 발생시키도록 허용함으로써 감소될 수 있다. 예를 들어, 일부 경우들에서, 시스템은, 그 이름이 문자열 조작에 의해 형성되는 테이블 또는 속성에 대한 질의들을 불허하도록 구성될 수 있는데, 그 이유는 사용자가 그것을 이용하여 사람들__ 또는 버전__ 또는 버전들__에 직접적으로 액세스

할 수 있기 때문이다. 시스템은 실행 시간에 컴파일링되는 transact-sql 문들(statements)과 같은 동적 SQL을 불허할 수 있다. 일부 경우들에서, 이러한 제한들은 일시적일 수 있다. 질의가 테이블에서의 속성들의 이름들 또는 모든 테이블들의 이름들을 요청하는 경우, 질의는 속성 버전_ 및 테이블 버전들_을 제거하도록 수정될 수 있고, 다른 테이블 이름들로부터의 후미 밑줄 표시를 제거할 수 있다.

[0121] 상기의 설명은 테이블들 및 속성들이 가상 클론 1에서 생성되었고, 후속하여 수정되지 않는다고 가정한다. 그러나, 사용자는 고속 카피들(본 명세서에서 "가상 클론들"이라고도 지칭됨)을 생성한 이후 데이터베이스 시스템의 스키마(schema)를 변경하여, 상이한 버전들이 상이한 스키마들을 갖도록 원할 수 있다. 일부 경우들에서, 테이블들_ 및 속성들_(둘다 2개의 밑줄 표시를 가짐)로 명명된 2개의 새로운 테이블들이 정의될 수 있다(예를 들어, 프로세서에 의해, 메모리에 저장될 수 있다). 테이블들_ 테이블은 버전들에서의 테이블들의 이름들을 레코딩할 수 있다. 속성들_ 테이블은 각각의 버전에서의 각각의 테이블에서의 속성 이름들을 레코딩할 수 있다.

[0122] 일부 경우들에서, 임의의 2개의 가상 클론이 동일한 이름 X를 갖는 테이블을 갖는 경우, 둘 모두는 X_로 명명된 동일한 물리 테이블에 저장될 수 있다. X_ 테이블은 X 테이블들 중 어느 하나에 나타나는 속성들을 포함할 수 있다. 이들 2개의 X 테이블들 각각이 속성 Y를 포함하지만, 이들이 상이한 타입의 엔트리들을 갖는 경우, X_ 테이블은 상이한 이름들을 갖는 2개의 상이한 속성들을 포함할 수 있고, 이들은 속성들_ 테이블에 의해 대응하는 Y 속성들에 맵핑된다.

[0123] 2개의 상이한 가상 클론들과 연관된, X로 명명된 2개의 상이한 테이블들이 그들의 가상 기본 키로서 상이한 속성(또는 속성들의 세트)을 갖는 경우, 2개의 테이블들은 개별적으로 저장될 수 있고, 테이블들_ 테이블은 별개의 테이블들에 할당된 이름들을 저장할 수 있다.

[0124] 일부 경우들에서, 시스템은 단일 가상 클론을 직렬화(serialize) 및 역직렬화(deserialize)하고, 단일 가상 클론의 해시를 획득하기 위한 (예를 들어, 프로세서에 의해 실행되는) 방법을 구현할 수 있다. 직렬화는 그 가상 클론에 대해 가시적이고/이거나 적용가능한 테이블들 및 데이터를 바이트들의 스트림으로 변환할 수 있고, 이는 그 가상 클론에 대한 스키마 및 데이터를 인코딩한다. 역직렬화는 그 스트림을 완전한 데이터베이스로 다시 돌려 놓을 수 있다. 일부 구현들에서, 직렬화 및 역직렬화 둘다는, 사용자 코드에 의해서가 아니라 신뢰된 코드(trusted code)에 의해 행해질 수 있다(예를 들어, 프로세서에 의해 실행됨). 일부 경우들에서, 역직렬화는 가상 클론 1이 생성되는 순간에만 수행될 수 있다. 통상적으로, 가상 클론 1은 사용자 코드에 대해 가시적인 테이블들이 없는 빈(empty) 데이터베이스이다. 가상 클론을 역직렬화하는 것은 직렬화 이전에 가시적이었던 테이블들 및 데이터를 재생한다.

[0125] 일부 경우들에서, 데이터베이스의 해시는 일부 정규적 순서(canonical order)로 생성되고 있다고 가정하여, 그 직렬화 스트림의 SHA2-384 해시일 수 있다. 유사하게 언급하면, 데이터베이스가 SQL 질의들의 특정 시퀀스에 의해 생성된다면, 그 데이터베이스는 어떤 컴퓨터가 그 질의들을 실행하는지에 관계없이 동일한 해시를 제공한다. 따라서, 예를 들어, 시스템은 그들이 생성된 순서로 테이블들을 직렬화하고, 그들이 생성되었던 순서로 테이블 내의 속성들을 직렬화할 수 있다. 대안적으로, 시스템은 테이블들을 알파벳 순서로 직렬화하고, 테이블 내의 필드들을 알파벳 순서로 직렬화할 수 있다. 다른 구현들에서, 이것은 각각의 테이블 및 각각의 레코드의 해시들을 캐싱함으로써 최적화될 수 있다.

[0126] 고속 카피들 대 스냅샷들(Fast-Copies vs Snapshots)

[0127] 일부 구현들에서, 데이터베이스 또는 데이터 구조는 3개의 상이한 방식들, 즉, 보통의 카피를 정의 및/또는 생성하는 것, 또는 "고속 카피"(가상 클론)를 정의 및/또는 생성하는 것, 또는 "스냅샷"을 정의 및/또는 생성하는 것에 의해 카피될 수 있다. 스냅샷은 변경불가능하다는 점에 의해, 보통의 카피 및 고속 카피와는 상이하다. 보통의 카피 및 고속 카피는 둘다 변경가능하지만, 저장된 정보의 양에 있어서 상이하다.

[0128] 도 9에서, 처음에 (좌측 트리, (a)) 데이터 1이 "뷰" X를 통해 액세스될 수 있다. 뷰 X를 통해 데이터에 대해 관독 및 기입하는 것 둘다가 가능하다. 이후에, 스냅샷 Y가 ((b)에서) X로부터 생성되고, 이것은 데이터의 카피 2를 생성하며, 그 후에 1은 변경불가능하게 된다(회색). 이후, 2에서의 데이터는 X를 통해 수정되거나 관독될 수 있고, Y는 1에서 데이터를 관독할 수 있지만, 그것을 수정할 수는 없다. 다음, (c)에서, 스냅샷 Z가 X로부터 생성되고, 이는 변경가능한 카피 3을 만들고, 2는 변경불가능하게 된다. 이 스냅샷들은 1의 완전한 카피인 2를 만드는 것에 의해 구현될 수 있다. 대안적으로, 스냅샷은 스냅샷이 생성된 후에 발생하는 변경들만을 포함하는 2를 만드는 것에 의해 구현될 수 있다. 어느 방식으로든, 뷰들 Y 및 Z는 단지 데이터의 관독만을 허

용하고, 데이터에 기입하는 것은 허용하지 않는다.

[0129] 스냅샷 동작은 보통의 카피 동작과 상이하다. 보통의 카피 동작은 도 9에 대하여 설명될 수 있다. 처음에, 도 9의 (a)에서, 데이터 1은 뷰 X에 의해 판독 또는 수정될 수 있다. 이후 (b)에서, 보통의 카피가 수행되고, 이는 뷰 Y를 생성하고, 또한 1에서 데이터의 2개의 카피 2 및 3을 생성한다. 그 시점에서, 1이 삭제될 수 있다. 그 후, 2는 X를 통해 2에 액세스함으로써 판독 또는 수정될 수 있고, 3은 Y를 통해 3에 액세스함으로써 판독 또는 수정될 수 있다.

[0130] 이러한 보통의 카피 동작은, 도 9에 대하여 설명될 수 있는, 다음의 방식으로 고속 카피 동작과 상이하다. 처음에 (a)에서, 데이터 1은 뷰 X에 의해 판독 또는 수정될 수 있다. 이후 (b)에서, 고속 카피 동작이 수행되고, 이는 뷰("고속 카피"라고도 지칭됨) Y를 생성한다. 이러한 고속 카피 동작은, 각각 초기에 비어있는 2 및 3을 생성한다. 1에서의 데이터는 삭제되지 않지만, 변경불가능하게 된다. 판독이 X 또는 Y를 통해 즉시 수행되는 경우, 그것은 1로부터 패스 스루(pass through) 및 판독될 것이다. X를 통해 기입이 수행되면, 변경이 2에 레코딩되고, 1은 변경되지 않는다. 기입이 Y를 통해 수행되면, 변경은 3에 레코딩되고, 1은 변경되지 않는다. 판독이 X에 대해 수행되고, 요청된 정보가 2에 레코딩된 변경의 일부라면, 2로부터의 그 정보가 반환된다. 그러나, 2가 그 데이터의 조각에 대한 어떠한 변경들도 포함하지 않는 경우, 판독은 1로 패스 스루된다. 유사하게, Y를 통한 판독은 먼저 3을 체크할 것이고, 답변이 3에서 발견되지 않는 경우에만 1로 계속된다. 고속 카피의 순간에서의 데이터는 1로서 도시되고, 그것이 변경불가능하기 때문에 회색이다. 나중에 여전히 (c)에서 뷰 X는 고속 카피되어 Z를 제공하고, 이는 새로운 4 및 5를 초래하고, 2에서는 변경불가능하게 된다. 마지막으로, 뷰 Y가 삭제되면, 그 결과는 도 9의 (d)에 도시된 바와 같다. 도 9의 (d)에서, 3은 존재하지 않는다. 3에 도달할 수 있는 어떠한 뷰들도 더 이상 존재하지 않기 때문에 그것은 삭제된다. 도 9에서, X, Y 및 Z 각각은 "뷰"로 지칭되고, 또한 "고속 카피"라고도 지칭된다.

[0131] 일부 구현들에서, 가비지 수집의 형태는 결국 도 9의 (d)에 도시된 데이터 구조에 대해 수행될 수 있다. 1에서의 데이터를 취하고, 2에서의 변경들을 적용함으로써, 1 및 2를 병합하는 것이 가능하다. 이어서, 이 병합된 데이터로 2를 대체하고, 1을 완전히 삭제하는 것이 가능하다. 이러한 가비지 수집은 X 및 Z로부터의 미래의 판독들의 속도를 증가시킬 수 있다. X로부터의 판독은 4, 2 및 1을 체크해야 하기보다는, 최악의 경우에 4 및 2를 체크하기만 하면 되기 때문에 그러한 판독들은 보다 고속일 수 있다. 유사하게, Z로부터의 판독들은 보다 고속일 것이며, 그 이유는 이들이 5, 2, 및 1이 아니라, 최악의 경우에 5 및 2를 체크하기만 하면 되기 때문이다.

[0132] 고속 카피가능 맵들(Fast Copyable Maps)

[0133] 키-값 쌍들의 세트(a set of key-value pairs)는 해시 테이블과 같은 다수의 방식으로 구현될 수 있다. 그러한 시스템(또는 프로세서)은 $O(1)$ 시간에서의 키와 연관된 값을 추가, 삭제, 또는 수정할 수 있다. 이것은 또한 $O(1)$ 쌍당 시간(time per pair)에서의 정의되지 않은 순서로 키-값 쌍들을 통해 반복할 수 있다. 그러한 데이터 구조는 또한, 예를 들어, 각각의 쌍에서의 값으로서 빈 문자열을 가짐으로써 세트들을 구현할 수 있다. 이것은 요소에 대한 세트 멤버십을 체크하기 위한 $O(1)$ 시간을 제공할 것이지만, 교집합(intersection) 및 합집합(union)과 같은 연산들은 더 느릴 것이다.

[0134] 고속 카피가능 맵은 몇 가지의 방식으로 구현될 수 있다. (도 9를 예로서 이용하는) 한가지 방식은 각각의 원(또는 가상 클론)에 대한 객체를 생성하고—각각의 객체는, 예를 들어, 자바 해시 맵(Java HashMap)을 포함함—, 뷰를 표현하는 각각의 정사각형에 대한 객체를 생성하는 것이다. 처음에, X를 통한 판독들 및 기입들은 1 내의 기본 맵에 직접 패스 스루된다.

[0135] Y 고속 카피가 행해진 후, 빈 맵들 2 및 3이 생성되고, 1은 변경불가능한 것으로서 표시된다. 키-값 쌍이 X에 기입될 때, 이것은 2에 대한 기입을 초래할 것이다. X에 대한 판독은 먼저 2에 대한 판독을 초래할 것이다. 키가 2에서 발견되지 않으면, 1에 대한 판독이 수행된다. 유사하게, Z가 고속 카피된 후에, Z에 대한 판독은 5에 대한 판독을 초래한다. 만일 실패한다면, 이것은 2를 판독한다. 그리고, 이것이 실패하면, 이것은 1을 판독한다. 따라서, 주어진 정사각형에 대한 기입은 인접한 원에 대한 변경을 야기하고, 키가 발견되거나, 루트가 질의될 때까지 판독은 트리를 따라 올라갈 것이다. 도 9의 (b)에서, X에 대한 기입 동작이 주어진 키-값 쌍을 삭제하는 경우, 프로세서는 키-값 쌍이 존재하지 않는다는 것을 나타내는 특수한 값인 "값"으로 1로부터 2로 그 키를 카피할 수 있다. 따라서, 해당 키-값 쌍에 대한 X에 대한 미래 판독들은 2에서 중단될 것이고, 1로 계속하여 그 키에 대한 원래의 값을 반환하는 대신에, 그 키에 대한 값이 없다는 것을 보고할 것이다.

- [0136] Z에서의 쌍들에 대한 반복은 1에서, 그 다음 2에서의, 그 다음 5에서의 쌍들을 통해 반복함으로써 행해질 수 있다. 1에서의 키-값 쌍은, 예를 들어, 질의가 그것의 키가 2 또는 5 어느 것에서도 없다는 것을 나타내는 경우에만 이용된다. 유사하게, 2에서의 키-값 쌍은, 예를 들어, 질의가 그것의 키가 5에 없다는 것을 나타내는 경우에만 이용된다. 5에서의 키-값 쌍은, 예를 들어, 임의의 다른 맵을 체크하지 않고 이용되는데, 그 이유는 5가 원들의 트리의 리프이기 때문이다.
- [0137] 기본 맵이 임의의 순서로 반복되면, 이 방안은 또한 임의의 순서로 반복될 것이다. 기본 맵이 그들이 마지막으로 수정된 순서로 키-값 쌍들을 통해 반복하는 경우, 이 방안은 동일하게 될 것이다. 이들 경우들 모두에서, 고정 길이의 트리에 대해 요소당 시간은 $O(1)$ 이거나, 트리가 d 레벨들 깊이이면 $O(\log d)$ 이다.
- [0138] 기본 맵이 키들을 알파벳 순으로 정렬하는 것과 같이 소정의 다른 순서로 반복하는 경우, 이 방안은 자동적으로 동일한 것을 행하지 않을 것이다. 대신에, 시스템은 먼저 키들을 버퍼에 카피하고, 이어서 그들을 정렬하고, 그 후 그 버퍼를 통해 반복할 수 있다. 그것은, n개의 항목들에 대해 $O(1)$ 로부터 $O(\log n)$ 으로의 분할 상환된(amortized) 요소당 시간을 증가시키는 값비싼 동작이다. 그것은 일부 응용들에 대해 여전히 수용가능할 수 있다.
- [0139] 실행 시간에 추가 최적화들이 가능하다. 예를 들어, 도 9의 (c)에서, 2에서의 맵은 2 및 1을 병합하는 새로운 맵으로 대체될 수 있다. 그 다음, 새로운 2는 그 자신의 트리의 루트일 것이고, 1은 그 자식을 잃을 것이다. 1이 어떠한 자식들도 더 이상 갖지 않을 때, 1은 삭제될 수 있다. 유사하게, 도 9의 (c)에서, 5는 1, 2, 및 5의 병합으로 대체될 수 있고, 그 자신의 루트가 된다. 1 및 2와 같은 2개의 변경불가능한 테이블들을 병합하는 것은 시간이 좀 걸릴 수 있지만, 배경 스레드에서 행해질 수 있다. 추가적으로, 일단 마무리되면, 병합된 맵은 이전 맵과 교체될 수 있고, 결과적인 트리들은 더 알아지고, 판독들 및 반복들에 대한 속도들을 증가시킨다.
- [0140] 또한, 뷰들이 삭제되는 경우, 원들 중 일부는 액세스 불가능하게 되고, 이어서 메모리로부터 삭제될 수 있고, 이들 병합에 대한 임의의 작업이 중단될 수 있다. 이것은 시스템이 도 9의 (d)를 반영하게 한다. 이것은 기본 언어에서의 가비지 수집 프로세스에 의해 트리거될 수 있다. 대안적으로, 시스템은 뷰를 명시적으로 삭제하기 위한 방법을 구현할 수 있고, 참조 카운트들을 유지하여 원이 홀로 되는 때를 결정할 수 있고, 이 시점에서 그것은 삭제된다.
- [0141] 키-값 쌍들의 삭제는 키-값 쌍을 실제로 삭제함으로써 루트 맵에서 구현될 수 있다. 루트 이외의 맵들에서, 그러한 삭제는 값을 특별한 "삭제된" 객체로 변경함으로써 행해질 수 있다. 맵들을 병합하는 것이 새로운 맵이 루트가 되게 하면, "삭제된" 객체를 갖는 키-값 쌍들을 제거하기 위해 그의 키-값 쌍들을 통한 반복이 행해질 수 있다.
- [0142] 다른 방안은 해시 테이블들의 트리를 갖는 대신에, 트라이(trie)와 같은 단일의 데이터 구조를 이용하는 것이다. 트라이에 대해, 각각의 키는, 예를 들어, 16, 32 또는 64 비트와 같은 미리 정의된 크기의 고정 길이 해시일 수 있다. 맵은 이진 트리에 의해 표현되고, 그 n번째 레벨은 키의 n번째 비트를 나타내고, 16 (또는 32 또는 64) 레벨들을 따라 내려가는 것에 의해 키가 록업된다. 그 다음, 값은 최하부 레벨에서 리프에 저장된다. 키 해시 충돌들을 처리하기 위해, 리프는 키-값 쌍들의 리스트를 저장하지만, 그 리스트는, 특히 과도한 충돌을 야기하는 키들을 선택하려고 시도할 수 있는 공격자에게 해시 함수가 알려지지 않으면, 평균적으로 매우 짧은 것으로 예상될 것이다. 2진 트리보다는 n항(n-ary) 트리를 이용함으로써 속도 및 메모리 요건들 둘다를 개선하는 것도 가능하다. 예를 들어, 32항 트리는 $32=2^5$ 이기 때문에, 많은 레벨들로서 5번째 레벨만을 가질 것이다.
- [0143] 전술한 것은 표준 트라이를 설명한다. 표준 트라이를 고속 카피가능 트라이로 변환하기 위해, 트리에서의 각각의 예지는 어느 데이터 버전들이 적용되는지에 관한 정보로 라벨링될 수 있다. 따라서, 도 9의 (a)에서, 주어진 키-값 쌍이 추가될 수 있고, 그것의 예지들은 1로 표시될 수 있다. 이어서, Y 및 Z가 고속 카피된 후에, 키-값 쌍이 Z에 추가될 수 있다. 그것은 더 많은 예지들을 트리에 추가하는 것을 수반할 수 있으며, 이는 각각 5로 표시될 수 있다. 키가 뷰 Z로부터 판독될 때, 시스템은 루트에서 시작할 수 있고, 키의 비트들과 매칭되는 예지들을 따르지만, 1, 2, 또는 5로 표시되는 예지들만을 따른다. 물론, 이것은 도 9의 (c)의 그래프가 메모리에 명시적으로 저장되거나, 적어도 Z가 도 9의 (c)의 그래프의 루트로의 경로가 1, 2, 및 5로 표시된 원들을 패스스루하는 것을 (예를 들어, Z와의 연관을 저장함으로써) 식별할 수 있는 것을 의미한다.
- [0144] 보다 고속 카피들이 행해짐에 따라, 예지들에 대한 조건부 체크(conditional check)가 더 복잡해질 수 있다. 예를 들어, 질의는 번호들 {1, 2, 5, 17, 32, 65, 82, 97, 99} 중 하나로 표시되는 예지들만을 따라 트라이를 결국 따라 내려갈 수 있다. 그러나, 일부 응용들에서, 도 9의 그래프가 장기간으로 루트로부터 최하부 근처로

의 길고 얇은 체인으로 진화할 것으로 예상되는 경우가 있을 수 있고, 이것은 그 후 최하부 근처의 많은 분기들로 분기한다. (도면들에서의 경우와 같이) 원들에서의 번호들이 숫자 순서로 할당되는 경우, 에지가 긴 리스트 {1, 2, 5, 17, 32, 65, 82, 97, 99}에서의 임의의 숫자로 표시되는지를 체크하는 대신에, 그 번호가 리스트 {<80, 82, 97, 99}에 있는지를 체크할 수 있고, 여기서 "<80" 엔트리는 80 미만의 임의의 수와 매칭된다. 즉, 트리가 최하부에서 작은 트리를 갖는 체인인 한, 도 9에서의 트리가 시간에 걸쳐 더 높이 성장하더라도, 부울 조건은 시간에 걸쳐 복잡성에 있어서 성장할 필요가 없다. 키-값 쌍을 삭제하기 위해, 전술한 바와 같이, 특별한 "삭제된" 객체가 이용될 수 있다. 또는, 그러한 삭제는 에지들을 특정 버전들에서 삭제되는 것으로 표시함으로써 행해질 수 있다. 그 후, 버전은 키를 삭제할 수 있으며, 이는 트라이에서의 일부 에지들을 그 버전에 대해 삭제된 것으로서 표시한다. 그 경우에, 도 9의 (c)에 대해, 그 에지에 대해 언급되는 {1, 2, 5} 중 가장 높은 번호가, 에지가 삭제되는 것이 아니라 추가되는 것을 나타내는 경우, 에지는 뷰 Z에 대해 트라이에 있는 것으로 카운트된다.

[0145] **고속 카피가능 어레이들**

[0146] 고속 카피가능 맵이 생성되면, 다른 데이터 구조들의 고속 카피가능 버전들이 생성될 수 있다. 예를 들어, 고속 카피가능 어레이는 이전 섹션에서 설명된 고속 카피가능 맵과 같은 고속 카피가능 맵과 실제 어레이를 결합함으로써 형성될 수 있다. 처음에, 단지 하나의 가상 클론 또는 뷰만이 존재할 때, 판독들 및 기입들은 어레이로 직접 진행한다. 고속 카피 후에, 기입들은 맵으로 진행하고, 각각의 키-값 쌍은 키로서 어레이 인덱스를 이용하고, 값으로서 어레이 요소를 이용한다. 판독은 먼저 맵을 체크하고, 그것이 인덱스를 발견하지 못하면, 어레이로부터 판독한다. 다수의 고속 카피 동작들은 맵들의 트리를 초래할 수 있다. 하나의 가상 클론 또는 뷰로부터의 판독은 그 키를 갖는 맵을 찾을 때까지, 또는 실제 어레이인 트리의 루트에 도달할 때까지, 트리를 따라 올라갈 것이다.

[0147] 이전과 같이, 버전 트리를 때때로 압축하는 것이 가능하다. 예를 들어, 도 9의 (c)에서, 1, 2, 및 5를 반영하는 새로운 어레이를 생성하는 것이 가능할 것이다. 새로운 어레이가 생성되면, 뷰 Z는 빈 맵과 연관된 이 새로운 어레이에 접속할 수 있다. 그러한 압축 후에, 트리를 다시 성장시키는 추가의 고속 카피들이 있을 때까지, 판독들 및 기입들 둘다 더 고속일 것이다. 이 프로세스는 반복(repeated)/반복(iterated)될 수 있다. 동일한 방안이 단지 어레이들이 아니라, 거의 모든 수집에 적용된다. 임의의 수집은 고속 카피가능 맵을 포함하는 객체에 래핑(wrapped)되고, 판독들 및 기입들은 수집을 체크하기 전에 맵을 체크한다. 판독들 및 기입들은 다수의 고속 카피 동작들이 버전 트리를 성장시킬 때 더 저속이 될 수 있고, 그 후 트리가 압축될 때 더 고속이 될 수 있다.

[0148] **고속 카피가능 파일시스템**

[0149] 고속 카피가능 파일시스템에서, 가상 클론 또는 뷰 객체는 전체 파일시스템을 나타낼 수 있다. 가상 클론 또는 뷰 객체는, 사용자에게, 파일들을 포함하는 디렉토리들의 트리를 포함하는 별도의 하드 드라이브인 것으로 보인다. 고속 카피 동작은, 사용자에게, 원본(original)의 정확한 카피를 갖는 새로운 하드 드라이브를 생성하는 것으로 나타난다. 그와 같이, 두 드라이브들은 독립적으로 수정될 수 있고, 각각에 대한 변경들은 다른 것에 영향을 미치지 않을 것이다. 이것은 모든 버전들로부터의 파일들을 포함하는 하나의 실제 디렉토리를 갖는 것에 의해 구현될 수 있다. 본 명세서에 설명된 바와 같이, 고속 카피가능 데이터베이스는 각각의 버전에 대한 디렉토리 트리, 및 각각의 디렉토리에서의 파일들에 대한 포인터들을 저장한다. 2개의 버전들 모두가 동일한 파일을 포함하는 경우, 시스템은 실제 하드 드라이브 상에서 하나의 실제 카피만을 필요로 하고, 데이터베이스는 실제 카피에 대한 2개의 포인터들을 저장할 수 있다. 파일에 대한 마지막 포인터가 삭제될 때, 파일을 볼 수 있는 마지막 뷰로부터 파일을 삭제함으로써, 시스템은 물리적 하드 드라이브 상에서의 물리적 파일을 삭제한다. 이러한 방안은 O(1) 동작을 카피하는 것을 행한다. 이것은 또한 중복 파일들의 단일 카피만을 저장함으로써 하드 드라이브 공간을 절약한다.

[0150] 시스템은 파일들을, 예를 들어, 2KB 청크들(chunks)과 같은 작은 청크들로 분할함으로써 더 최적화될 수 있다. 그 후, 데이터베이스는 해당 파일을 구성하는 청크들을 가리키는, 각 파일에 대한 포인터들의 리스트를 저장한다. 그러한 방식으로, 2개의 파일이 처음 2KB에 대해 동일한 콘텐츠를 갖는 경우, 그 청크의 하나의 카피만이 물리적으로 저장되고, 이는 공간을 더 절약한다. 이러한 방식으로, 동일한 파일에 대한 액세스를 갖는 2개의 가상 클론들 또는 뷰들의 경우 뿐만 아니라, 단일 가상 클론 또는 뷰가 2개의 상이한 디렉토리들에서 동일한 파일들을 갖는 디렉토리 트리를 나타내는 경우에 대해 중복이 감소된다.

[0151] 파일들 또는 전체 디렉토리 서브트리들의 해시들을 신속하게 계산할 필요가 있는 경우, 그 정보는 또한 데이터

베이스에 저장될 수 있다. 각각의 청크의 해시는 청크가 수정될 때 저장되고 수정될 수 있다. 실제로, 일부 경우들에서 기본 파일시스템에서의 청크의 파일명은 16 진법에서의 파일의 해시인 것으로 정의될 수 있다. 주어진 청크 X에 대해 하나의 포인터만이 존재하는 경우, 청크의 수정은 파일명에 대한 변경을 포함할 수 있다. 주어진 청크 Y에 대해 다수의 포인터들이 있을 때, 청크 Y의 수정은 청크의 카피를 만들고, 카피를 수정하고, 그 다음에 카피를 그의 해시로 재명명함으로써 구현될 수 있다.

[0152] 그러한 경우에, 파일의 해시는 그 청크들의 연쇄의 해시와 같은 증분 해시(incremental hash)일 수 있다. 대안적으로, 파일에서의 해시는 파일 내의 청크들에 대한 (위치, 데이터) 쌍들의 해시들의 합으로서 정의될 수 있다. 디렉토리의 해시는 디렉토리에서의 파일들의 해시들의 합일 수 있다. 파일이 변경될 때, 데이터베이스는 영향을 받은 해시들을, 예를 들어, 배경 스레드에서 업데이트할 수 있다. 대안적으로, 해시들은 방법에 대한 호출이 해시를 요청할 때까지 계산되지 않을 수 있다. 그러한 경우들에서, 해시는 요구에 따라 계산될 수 있다. 이 아키텍처는 통상적인 파일 시스템들에서는 보통 존재하지 않는 추가적인 특징의 구현을 용이하게 한다. 파일시스템 "트랜잭션들"은 데이터베이스 의미로 구현될 수 있다. 시스템이 (예를 들어, 프로세서를 통해) 파일시스템 방법들을 호출하여 파일시스템을 조작할 때, 시스템은 트랜잭션을 개시하기 위해 (예를 들어, 프로세서에 의해 실행되는) 방법을 호출하고, 이어서 파일들을 추가, 제거, 또는 수정하기 위해 다수의 호출을 행할 수 있고, 이어서 트랜잭션을 커밋(commit)하기 위해 (예를 들어, 프로세서에 의해 실행되는) 방법을 호출할 수 있다. 일부 그러한 구현들에서, 시스템이 언제든지 파워 다운되었더라도, 파일시스템은 결국 그 동작들 모두를 반영하거나 어느 것도 반영하지 않을 것이다. 이 능력은 일반적으로 통상적인 파일시스템들에 추가되지 않는데, 그 이유는 시스템을 느리게 하는 방식으로 통상적인 데이터베이스에 대한 호출을 수반할 것이기 때문이다. 그러나, 본 명세서에 개시된 고속 카피가능 파일시스템은 다른 이유들로 계산 비용을 이미 지불하고 있으므로, 트랜잭션들을 추가하는 것은 실제로는 매우 적은 추가/증분 비용을 갖는다.

[0153] 일부 구현들에서, 파일시스템의 레벨에서의 "트랜잭션"은 데이터베이스의 레벨에서의 "트랜잭션"으로서 구현될 수 있다. 동작의 중간 동안 고장이 있다면, (예를 들어, 프로세서에 의해 실행되는) 데이터베이스 복구 방법은 데이터베이스를 데이터베이스 트랜잭션의 시작으로 롤백(rolling back)하는 것을 포함할 수 있다. 그 다음, 데이터베이스 복구 방법은 저장된 청크들을 패스 스루하여, 데이터베이스에 의해 더 이상 참조되지 않는 것이 있는지 알아보고, 임의의 그러한 식별된 청크들을 삭제할 수 있다. 이것은 정확한 상태의 빠른 복구를 용이하게 하고, 그 후 낭비된 공간을 복구하기 위해 (예를 들어, 배경 스레드에서의) 더 긴 프로세스가 뒤따른다. 이 방안에서, 청크 파일들은 커밋이 데이터베이스에 전송된 이후까지 삭제되지 않고, 데이터베이스가 업데이트된다.

[0154] 다음은 (예를 들어, 자바로부터 액세스된) 고속 카피가능 파일시스템을 위한 시스템의 상세한 논의를 포함한다. 사용자 프로그램(예를 들어, 자바로 기입되고 프로세서에 의해 실행됨)은 공통의 파일시스템 동작들(common filesystem operations)에 더하여 몇 가지의 비공통의 것들을 지원하는 고속 카피가능 파일시스템 플랫폼(예를 들어, 또한 자바로 기입되고, 프로세서에 의해 실행됨)의 방법들을 호출한다.

[0155] 고속 카피가능 파일시스템은 (프로세서에 의해 실행되는) 사용자 코드 및/또는 프로세스가 볼륨들, 디렉토리들 및 파일들인 것으로 보이는 것에 액세스하게 할 수 있다. 알려진 파일시스템들과 달리, 본 명세서에 설명된 고속 카피가능 파일시스템들은 다음을 용이하게 할 수 있다:

- [0156] - 파일, 디렉토리 트리, 또는 볼륨의 고속 카피
- [0157] - 파일, 디렉토리 트리 또는 볼륨의 해시의 빠른 계산
- [0158] - 파일, 디렉토리 트리, 또는 볼륨의 직렬화 및 역직렬화
- [0159] - (차동 카피들(differential copies)에 대해) 그들의 서브세트의 직렬화 및 역직렬화
- [0160] - 기입 속도에 비해 판독 속도에 대한 강조
- [0161] - 중복 디렉토리들, 파일들, 및 파일들의 부분들의 자동 중복제거
- [0162] - 그의 해시만이 주어지면 파일 또는 디렉토리를 찾는 능력(즉, 해시들은 포인터들임)
- [0163] - 파일시스템에 대한 메타데이터의 데이터베이스에 대한 판독 전용 SQL 질의들
- [0164] 일부 구현들에서, 파일시스템은 다음을 구현한다:
- [0165] **볼륨** - 그의 루트 디렉토리를 갖는/그의 루트 디렉토리로서의 가상 하드 드라이브

- [0166] **디렉토리** - 일부 메타데이터 + 0 이상의(<263) 파일들 및 디렉토리들의 정렬된 리스트
- [0167] **파일** - 일부 메타데이터 + 0 이상의(<263) 바이트들의 시퀀스
- [0168] **액세스 제어 리스트**(Access Control List) - 아이덴티티들의 그룹을 포함하는 파일 또는 디렉토리
- [0169] **아이덴티티** - 파일시스템에 액세스할 수 있는 누군가를 나타내는 (ACL에 저장된) 공개 키
- [0170] **엔티티** - 파일 또는 디렉토리(또한 ACL일 수도 있고, ACL이 아닐 수도 있음)
- [0171] **체크** - 0 내지 `CHUNK_SIZE` 바이트들의 시퀀스(자바 파일로서 물리적으로 저장됨)
- [0172] **체크 트리** - 1개의 엔티티를 표현하기 위해 트리로 배열된 1개 이상의 체크들의 세트
- [0173] **`CHUNK_SIZE`** = 4096 (또는 다른 상수)
- [0174] **메타데이터** - 이름, 크기(계산된 다수의 방식들), 액세스 허가, 사용자 정의 메타데이터 등을 포함하는, 하나의 엔티티에 관한 정보
- [0175] **허가들** - 엔티티는 모든 사람에 대한 `READ` 허가들을 갖거나, 그것은 `READ` 허가들을 갖는 그들의 명시된 ACL 그룹을 갖는다. `WRITE`에 대해서도 마찬가지이다.
- [0176] 일부 구현들에서, 파일시스템은 다음을 구현하지 않는다:
- [0177] **링크들** - 이 파일시스템은 링크들, 하드 링크들, 소프트 링크들, 심링크들(symlinks), 에일리어스들(alises) 등을 갖지 않는다. 그러나, 고속 클로닝(fast cloning)은 유사한 것을 허용한다: 디렉토리에서의 라이브러리예 링크를 넣는 대신에, 당신은 간단히 전체 라이브러리를 당신의 디렉토리에 카피할 수 있고, 이것은 당신이 링크를 이용하는 것처럼 적은 스토리지를 이용할 것이다.
- [0178] **디렉토리들** . 및 .. - 데이터 구조들은 . 또는 .. 디렉토리들의 존재를 반영하지 않는다. 그러나, 인터페이스는 "/여기/./저기"와 같은 것들을 포함하는 사용자로부터의 경로 문자열들을 올바르게 해석할 수 있다.
- [0179] **압축된/암호화된 디렉토리들** - Windows 또는 OSX에서와 같이, 압축된 또는 암호화된 디렉토리의 등가물이 없다. 그러나, 보다 약한 형태의 압축(체크들을 중복제거하는 것)이 보편적으로 구현된다. 더욱이, 디렉토리 트리에서 그 파일들 및 서브디렉토리들 각각에 대해 참(true)으로 설정된 `readAcl` 플래그를 갖는다면, 데이터, 파일명들 및 디렉토리 이름들은 암호화될 것이지만, 디렉토리 트리의 토폴로지 및 파일 크기들은 여전히 가시적일 것이다.
- [0180] **Rsync 및 시간 머신** - 일부 경우들에서 차동 카피 및/또는 차동 백업들(differential backups)이 파일시스템 상에서 구현될 수 있다.
- [0181] **유닉스-스타일 허가들** - 고속 카피가능 파일시스템의 성질 때문에, 파일 또는 디렉토리에 대한 소유자, 그룹, 또는 사용자들의 고유 개념이 없다. 판독, 기입, 및 실행 동작들 중에서, 처음 2개만이 의미가 있다. 따라서, 고속 카피가능 파일시스템은 각각의 파일 또는 폴더가 `READ`를 위한 선택적 액세스 제어 리스트(ACL), 및 `WRITE`를 위한 다른 ACL을 허용할 수 있다. 사용자의 애플리케이션이 그 이상을 필요로 하는 경우, 사용자는 사용자 메타데이터의 일부로서 그것을 추가할 수 있다. 또한, 여기서 `READ` 및 `WRITE`의 정확한 정의들은 유닉스 및 다른 시스템들에서의 것들과 약간 상이하다.
- [0182] 사용자에게, 파일시스템은 다음과 같이 기능하는 것처럼 보인다: 볼륨은 루트 디렉토리를 갖는/루트 디렉토리로서의 외부 하드 드라이브와 같다. 볼륨은 디렉토리 트리를 포함한다. 각각의 디렉토리는 디렉토리들 및 파일들의 세트를 포함한다. 이것은 2개의 디렉토리들 . 및 .. 또는 어떤 등가물도 포함하지 않는다(그렇지만 일부 경우들에서 사용자 인터페이스가 그들의 존재를 시뮬레이션할 수 있다). 일부 경우들에서, 사용자는 하드 링크들, 소프트 링크들, 심링크들, 에일리어스들 등을 생성하거나 또는 볼 수 없다. 가시적 디렉토리 트리는 DAG 또는 순환 그래프보다는 트리인 것으로 (사용자에게) 나타난다. 사용자는 고속 카피가능 파일시스템을 보통의 파일시스템으로서 보지만, 기입들은 보통보다 느리고, 이하의 동작들은 보통보다 더 빠르다:
 - [0183] - 파일, 디렉토리, 또는 볼륨을 고속 카피
 - [0184] - 파일 또는 전체 디렉토리 트리의 해시를 획득
 - [0185] - 파일, 디렉토리, 또는 전체 디렉토리 트리의 바이트 단위의 크기를 획득

- [0186] - 파일 또는 디렉토리 트리를 스트림(예를 들어, 자바 스트림)으로 직렬화/역직렬화
- [0187] - 청크들의 세트를 스트림(예를 들어, 자바 스트림)으로 직렬화/역직렬화
- [0188] - (rsync와 같은) 차동 카피들을 정의하기 쉽게 하는 정보 획득
- [0189] - 주어진 볼륨에서의 파일들에 대한 메타데이터의 데이터베이스 상의 관독 전용 SQL 질의들
- [0190] (예를 들어, 프로세서를 이용하는) 시스템은 파일들 및 디렉토리들의 고속 클로닝을 수행하고, 파일 또는 디렉토리의 해시를 신속하게 획득하고, 파일들 및 디렉토리 트리들을 직렬화/역직렬화할 수 있다. 일부 경우들에서, 분배 해시 테이블(Distributed Hash Table)(DHT)이 고속 카피가능 파일시스템 플랫폼에 추가되고, 사용자들은 그의 해시가 주어지면, 인터넷으로부터 파일 또는 디렉토리 트리를 찾고 다운로드할 수 있다. 그러한 경우들에서, 다운로드는 효율적일 수 있어서, 사용자가 아직 갖지 않는 데이터만을 전송한다.
- [0191] 사용자는 볼륨의 콘텐츠를 디렉토리들의 유한 트리인 것으로서 본다—DAG, 순환 그래프 또는 무한 트리로서 보지 않음—. 이들 디렉토리들 각각은 파일들 및 다른 디렉토리들을 포함할 수 있다. 파일들 및 디렉토리들 중 일부는 ACL들로서 표시될 수 있고, 통상적으로 숨겨지지만, 요청될 때 사용자에게 의해 보여질 수 있다.
- [0192] 일부 경우들에서, 액세스 허가 모델은 READ 및 WRITE를 포함하지만, EXECUTE를 포함하지 않으며, ID들의 계층적 ACL 그룹들에 구축된다. 사용자는 그것의 readAll이 참이거나, 또는 그것의 readAll이 거짓이고 그것의 readAcl이 그 사용자를 포함하는 경우에 파일 또는 폴더에 대한 READ 허가들을 갖는다. WRITE 허가들은(writeAll 및 writeAcl로) 유사하게 정의된다.
- [0193] 다음의 표에서, 사용자가 행의 시작에 열거된 허가들을 갖는다면, 그것들은 그것의 열에 대해 열거된 것들 상의 채색된 동작(적색=권한들 없음, 황색=관독 권한들만, 녹색=관독 및 수정 권한들)을 수행하도록 허용된다. 처음 4개의 열들은 메타데이터를 위한 것이다.

	readAll, readAcl	writeAll, writeAcl	이름, userMetadata	모든 다른 메타데이터	파일 콘텐츠	디렉토리 콘텐츠
없음	관독 (황색)	관독 (황색)	(적색)	관독 (황색)	(적색)	관독 (황색)
READ	관독 (황색)	관독 (황색)	관독 (황색)	관독 (황색)	관독 (황색)	관독 (황색)
WRITE	관독 (황색)	관독, 수정 (녹색)	(적색)	관독, 수정 (녹색)	(적색)	(적색)
READ, WRITE	관독, 수정 (녹색)	관독, 수정 (녹색)	관독, 수정 (녹색)	관독, 수정 (녹색)	관독, 수정 (녹색)	관독, 수정 (녹색)

- [0194]
- [0195] 관독 = 사용자는 데이터를 관독할 수 있다.
- [0196] 관독의 부재 = 사용자는 데이터의 암호화된 버전만을 관독하거나, 파일 또는 디렉토리를 다른 장소로 카피하거나, 또는 데이터를 직렬화/역직렬화할 수 있다.
- [0197] 수정 = 사용자는 데이터의 임의의 부분을 변경할 수 있다. 디렉토리의 경우, 디렉토리 자신의 이름을 변경하는 것 및 그 디렉토리 내의 파일들/디렉토리들을 추가/삭제하는 것을 포함하지만, 반드시 그 디렉토리 내의 것들의 파일명들의 관독 또는 수정을 허용하는 것은 아니다.
- [0198] 일부 경우들에서, 사용자는 그의 허가들에 관계없이, 파일 또는 디렉토리 트리를 카피, 직렬화, 또는 역직렬화할 수 있다. 사용자가 그렇게 할 때, 사용자는 선택적으로 파일에 대한 또는 디렉토리 트리에서의 모든 것에 대한 기입 허가들을 설정할 수 있다(즉, 그들 각각에 대해 writeAll 및 writeAcl을 설정할 수 있다).
- [0199] "엔티티"는 파일 또는 디렉토리일 수 있다. 각각의 엔티티는 그 자신의 메타데이터 및 콘텐츠를 갖는다. 파일의 "콘텐츠"는 그 파일 내의 바이트 시퀀스인 것으로 정의된다. 디렉토리의 콘텐츠는 그 디렉토리 내의 파일들 및 디렉토리들의 해시들의 정렬된 시퀀스인 것으로 정의된다. 일부 경우들에서, 각각의 엔티티는 아래에 정의

된 바와 같이 청크들로 이루어진 청크 트리에 물리적으로 저장된다.

- [0200] "청크"는 (예를 들어, Java에 의해 제공되는) 기본 파일시스템에 단일 파일로서 물리적으로 저장되는 바이트들의 시퀀스이다. 고속 카피가능 파일시스템에서의 디렉토리들 및 파일들 및 ACL들은 가상일 수 있다. 그러한 구현들에서, 청크들만이 사용자의 하드 드라이브 상에 물리적으로 존재한다. 사용자들은 청크들을 보지 않지만, 기본적인 물리적 파일시스템은 청크들을 동작시키고/거나 이용한다.
- [0201] 청크는 (예를 들어, 자바에 의해 제공되는) 하드 드라이브의 물리적 파일시스템에서의 파일명을 갖는다. 일부 경우들에서, 그 파일명은 소문자 16 진법에서의 그의 콘텐츠의 256-비트 해시일 수 있다. 따라서, 청크가 디스크 메모리에 기입되면, 청크는 수정되지 않지만, 삭제될 수 있다. 일부 경우들에서, 청크들은 메모리에서의 (예를 들어, 자바에 의해 제공되는) 물리적 파일시스템에서의 단일 디렉토리에 저장될 수 있다. 다른 경우들에서, 청크들이 그들의 파일명의 처음 몇 개의 문자들에 따라 분할되는 서브디렉토리들을 포함하는 디렉토리가 메모리에 청크들을 저장하는 데 이용될 수 있다.
- [0202] 일부 경우들에서, 청크는 CHUNK_SIZE 바이트들보다 크지 않다. 이러한 전역 상수는, 예를 들어, 4*1024인 것으로 정의될 수 있다. 다른 경우들에서, CHUNK_SIZE는 OSX 희소 번들(sparse bundle)들에 의해 이용되는 4*1024와 8*1024*1024 사이에 있다.
- [0203] 하나 이상의 청크가 결합되어 "청크 트리"를 형성할 수 있으며, 이는 단일 엔티티(파일 또는 디렉토리)를 저장하는데 이용된다. 제1 청크는 "깊이" 바이트로 시작하여, 트리의 깊이를 제공한다. 따라서, 0x01은 트리가 루트 및 리프 둘다인 단지 단일 청크임을 의미하고, 0x02는 루트의 자식들이 모두 리프들임을 의미하는 등으로 된다. 깊이 바이트(depth byte) 이후에 아래의 메타데이터 섹션에 정의된 바와 같이, 엔티티에 대한 "고정된" 메타데이터 요소들이 뒤따른다.
- [0204] 충분히 작은 엔티티에 대해, 이것 다음에 메타데이터의 나머지가 뒤따르고, 그 다음 파일 또는 디렉토리의 콘텐츠가 뒤따른다. (디렉토리의 "콘텐츠"는 디렉토리 내의 각 파일 및 서브디렉토리의 32-바이트 해시들의 정렬된 리스트이다). 예로서, 저장된 데이터가 최대(at most) CHUNK_SIZE 바이트들인 경우, 데이터는 단일 청크에 맞고, 그 청크는 1의 깊이(0x01)를 가질 것이고, 그 청크는 그 엔티티에 대한 전체 청크 트리를 포함한다.
- [0205] 엔티티가 하나의 청크에 맞추어지기에 너무 크면, 고정 메타데이터 다음에 자식 청크들의 해시들의 시퀀스가 뒤따른다. 자식 청크들 각각은, 트리에서의 내부 청크들에 대한, 그들의 자식들의 청크들의 해시들의 시퀀스 등을 포함한다.
- [0206] 리프 청크들은, 폭 우선으로(breadth-first order) 방문될 때, 메타데이터의 나머지, 이후 파일 또는 디렉토리의 콘텐츠를 포함한다. 일부 경우들에서, 콘텐츠의 시작은 메타데이터의 끝으로부터의 별개의 리프 청크에서이다.
- [0207] 일부 경우들에서, 트리 토폴로지는 다음과 같이 고유하게 정의된다. 트리가 단일 청크에 맞을 수 있다면, 그것은 단일 청크 내에 맞는다. 그렇지 않으면, 시스템은 트리를 정의하여, 노드들이 폭 우선으로 방문되는 경우, 어떠한 단일 리프 노드도 메타데이터 및 콘텐츠 둘다로부터의 바이트들을 포함할 수 없다는 제약에 따라, 이전 노드들이 가능한 한 크게 되도록 할 수 있다.
- [0208] 일부 경우들에서, 각각의 엔티티(파일 또는 디렉토리)는 다음의 메타데이터를 다음의 순서로 포함할 수 있고, 여기서 편의상 3개의 카테고리들로 나누어질 수 있다(고정(fixed), 가변(variable), 민감(sensitive)).
- [0209] **고정:**
- [0210] 1. **파일시스템** - 2바이트 - 마지막으로 수정될 때의 파일시스템 버전
- [0211] 2. **엔티티 타입** - 1바이트:
- [0212] 0x00 = ENTITY_FILE = 파일
- [0213] 0x01 = ENTITY_DIR = 디렉토리
- [0214] 0x80 = ACL_FILE = ACL 그룹을 나타내는 파일
- [0215] 0x81 = ACL_DIR = ACL 그룹을 나타내는 디렉토리
- [0216] 3. **크기** - 8바이트 - 어떠한 청크 중복제거(deduping)도 없는 경우, 이 파일 또는 단일 디렉토리에서의 바이트

들의 수

- [0217] 4. **sizeSmetadata** - 8바이트 - 민감 메타데이터의 바이트들의 수
- [0218] 5. **sizeVmetadata** - 8바이트 - 가변 메타데이터의 바이트들의 수
- [0219] 6. **sizeTree** - 8바이트 - 디렉토리들에 대해서만 존재함: 어떠한 중복제거도 없었다면 이 디렉토리 트리에서의 파일들 및 디렉토리들의 총 크기
- [0220] 7. **readAll** - 1바이트 - 모든 사람이 이 파일 또는 디렉토리 상의 READ 허가들을 갖는 경우 참(0x01). 그렇지 않으면 이들을 ACL로 제한하기 위해 거짓(0x00).
- [0221] 8. **writeAll** - 1바이트 - 모든 사람이 이 파일 또는 디렉토리에 대한 WRITE 허가들을 갖는 경우 참(0x01). 그렇지 않으면 이들을 ACL로 제한하기 위해 거짓(0x00).
- [0222] **민감:**
- [0223] 9. **이름** - 파일명 또는 디렉토리 이름은 String.length()≤255인 비어 있지 않은 문자열(예를 들어, 자바 문자열)이다. 유사하게 언급하면, 최대 255 UTF-16 코드 유닛들. 일부 경우들에서, 모든 유니코드 문자들이 허용되며, 이는 슬래시(slash) 및 콜론(colon)을 포함한다. 일부 경우들에서, 경로 길이들에 대한 제한은 없다.
- [0224] 10. **userMetadata** - 키-값 쌍들의 리스트, 여기서 키 및 값은 둘다 문자열들이고(예를 들어, 임의의 법적 자바 문자열이 ok임), 동일한 키에 대해 다수의 쌍들이 존재할 수 있다. 이것은 사용자가 쌍들을 추가하기 위한 방법을 호출하지 않는 한 비어 있다.
- [0225] **가변:**
- [0226] 11. **readAcl** - 32 바이트 - readAll이 거짓인 경우 존재한다 - ACL 파일 또는 ACL 디렉토리 트리의 해시는 READ 허가들을 갖는 ID들의 ACL 그룹을 나타낸다. 시스템은 디렉토리의 readAcl이 디렉토리 내에 저장된 ACL 파일 또는 ACL 디렉토리의 해시라는 것을 보장할 수 있다. 파일의 readAcl은 파일과 동일한 디렉토리에 저장된 ACL 파일 또는 ACL 디렉토리의 해시일 수 있다.
- [0227] 12. **writeAclHash** - 32 바이트 - writeAll이 거짓인 경우 존재한다 - 가능하게는 상이한 엔티티를 지시하지만, readAcl과 유사한 해시이다.
- [0228] 13. **encryptedKeys** - readAll이 거짓인 경우 존재한다. ACL 그룹에서의 공개 키당 하나의 암호화된 키의 리스트이다. 리스트는, ACL 그룹에서의 ID들의 정렬된 리스트에 대응하는 순서로, 8바이트 카운트로서 저장되고, 그후 암호화된 키들 각각이 뒤따른다.
- [0229] 일부 경우들에서, 저장되는 4개 타입의 크기에 더하여, 저장되지 않은 5번째 종류가 존재하지만, 필요에 따라(예를 들어, 프로세서에 의해) 재계산된다. 이것은 이 파일 또는 디렉토리 트리에서의 고유한 청크들의 총 크기인 "중복제거된 크기(deduped size)"이다. 이것은, 파일 또는 디렉토리 트리 밖의 청크들을 무시하면서, 중복제거가 이 파일 또는 디렉토리 트리 내에서 발생하는 경우 디스크 메모리 상의 크기일 것이다.
- [0230] 일부 경우들에서, readAll이 거짓인 엔티티는(예를 들어, 프로세서를 이용하여) 부분적으로 암호화될 수 있다. "민감" 메타데이터는 암호화되지만, "고정" 및 "가변" 메타데이터는 암호화되지 않는다. 파일의 콘텐츠는 암호화되지만, 디렉토리의 콘텐츠는 암호화되지 않는다.
- [0231] 일부 경우들에서, 암호화는 랜덤 키를 갖는 XTS(NIST SP800-38E)일 수 있다. XTS "데이터 유닛"은 데이터 유닛이 단지 민감 부분인, 민감 메타데이터 및 가변 메타데이터 둘다를 포함하는 임의의 청크를 제외하고는, 각각의 전체 청크이다. 민감 메타데이터의 암호화는 랜덤 키를 이용한 XTS 암호화일 수 있다. 파일의 콘텐츠의 암호화는 동일한 키를 이용하지만 그의 비트가 플립되는 별개의 XTS 암호화일 수 있다. 따라서, 민감 메타데이터가 그의 길이를 확장함으로써 수정되면, 메타데이터의 일부는 재암호화될 수 있지만, 콘텐츠가 새로운 청크의 시작에서 시작하기 때문에 콘텐츠는 재암호화되지 않는다.
- [0232] 랜덤 키는 그 후 ACL 그룹으로부터 공개 키들의 정렬된 리스트에서의 공개 키들 각각으로(예를 들어, 프로세서를 이용하여) 암호화될 수 있고, 결과들은 암호화 키들(encryptedKeys)로서 저장된다. ACL 파일에 대해, ACL 그룹은 ACL 파일에서의 ID들의 정렬된 리스트이다. ACL 디렉토리에 대해, ACL 그룹은 그 루트가 그 디렉토리인 디렉토리 트리 내의 ACL 파일들에서의 ID들의 정렬된 리스트이다.
- [0233] 일부 경우들에서, ACL ID 그룹이 ID를 추가함으로써 변경될 때, 새로운 암호화된 키가 암호화된 키들의 리스트

에 추가될 수 있다. ID가 삭제될 때, 새로운 랜덤 키가 선택되고, 암호화된 키들의 리스트가 재계산되고, 민감 메타데이터 및 콘텐츠 둘다 이전 키(old key)로 암호해독되고 새로운 키로 재암호화된다. 기입 캐싱은, 예를 들어, ACL에 대한 몇몇 변경들이 빠르게 연속하여 행해질 수 있게 하고, 그 후 변경들에 대해 재암호화를 한번 수행함으로써 이러한 프로세스를 최적화하는데 이용될 수 있다.

[0234] 사용자 메타데이터는 (키, 값) 쌍들의 정렬된 리스트일 수 있다. 각각의 키는 문자열(예를 들어, 자바 문자열)이고, 각각의 키는 고유하다. 각각의 값은 자바 리스트<byte[]>일 수 있고, 리스트에서의 순서는 다양한 값들이 사용자에게 의해 추가되는 순서이다. 사용자는 값들을 추가 및 삭제할 수 있다. 남아 있는 값들이 없을 때, 쌍은 삭제된다.

[0235] 이러한 키들은 사용자가 원하는 임의의 문자열일 수 있다. 일부 구현들에서, 빈 문자열이 허용가능하지만, 널(null)은 그렇지 않다. 예들은 "creatorApp" 또는 "openWith", 또는 "소스"를 포함하고, 그 값은 디렉토리 트리의 해시이다(여기서 처음 2개의 경우들은 최상위 디렉토리에서 실행가능한 단일의 것을 갖는다). 다른 예들은, 시간 및 날짜와 연관된 작은 jpg 또는 "createdDate" 또는 "modifiedDate"와 연관된 "썸네일(thumbnail)", 또는 이 파일이 계속 존재하기를 원하는 각각의 사용자의 ID와 연관된 "핀드(pinned)"를 포함한다. 고속 카피 가능 파일시스템 플랫폼은 사용자가 어떤 키들을 제공하는지에 따라, 이들의 일부 또는 전부를 이용할 수 있다.

[0236] 일부 구현들에서, 크기 필드는 이 엔티티에 대한 체크 트리에서의 모든 체크의 바이트들에서의 총 크기를 부여하는 서명된 64비트 정수일 수 있다. 동일 체크가 체크 트리에서 두 번 나타나면, 이것은 두 번 카운트된다. 따라서, 크기 필드는 어떠한 중복제거도 없는 경우에 엔티티에 대한 체크 트리가 차지할 물리적 디스크 메모리 공간의 양이다.

[0237] treeSize 필드는 파일 또는 ACL에 대해서가 아니라, 디렉토리에 대해 존재한다. treeSize 필드는 이 디렉토리에 대한 크기에, 이 디렉토리에서의 각 파일의 크기를 더하고, 이 디렉토리에서의 각 서브디렉토리의 treeSize를 더한 합일 수 있다. 이는 중복제거가 없는 경우에 이러한 디렉토리 트리가 메모리에서 차지하는 공간의 양이다.

[0238] 일부 경우들에서, 고속 카피 가능 파일시스템 플랫폼은 또한 각각의 사용자가 얼마나 많은 바이트들을 책임지는지를 추적할 수 있다. 파일에 대해, 파일을 핀(pin)하는 사용자들에게는 "크기" 바이트들이 부과될 수 있다. 디렉토리에 대해, 디렉토리를 핀하는 사용자들에게는 "treeSize" 바이트들이 부과될 수 있다. 파일이 파일을 핀하는 것을 갖지 않는 경우, 앱(app)은 메모리 공간을 절약하기 위해 파일을 삭제하기로 결정할 수 있다. 디렉토리 트리가 핀된 파일들 또는 디렉토리들을 포함하지 않을 때, 그 전체 디렉토리 트리는 메모리 공간을 절약하기 위해 (예를 들어, 프로세서를 이용하여) 삭제될 수 있다. 이것은 사용자들이 이용되게 하는 공간이 사용자들에게 부과되어, 2명의 사용자들이 동일한 것을 핀할 때 발생하는 중복제거를 무시한다는 것을 의미한다. 고속 카피 가능 파일시스템 플랫폼은 고속 카피 가능 파일시스템 플랫폼이 디렉토리 및 그 디렉토리 트리 내부의 엔티티 둘다를 핀하지 않는 것을 보장할 수 있다.

[0239] 전술한 파일시스템은 고속 관독들 및 저속 기입들을 갖는다. 그것은 또한 고속 해시 계산을 갖는다. 사용자가 파일 또는 디렉토리의 해시를 요청하는 경우, 답변은 단순히 그 체크 트리의 루트 체크의 물리적 파일명이다. 이것은 메타데이터와 함께 관계형 데이터베이스에 저장될 수 있고, 따라서 신속하게 반환될 수 있다. 일부 경우들에서, 파일시스템 전체에 걸쳐 SHA2-256 해시가 이용될 수 있다. 다른 경우들에서, SHA3-256 해시(또는 임의의 다른 적절한 해시)가 파일시스템 전체에 걸쳐 이용될 수 있다.

[0240] 일부 구현들에서, 파일시스템은 저속 기입들을 갖지만, 그것은 체크들 중 하나에서 동일 바이트들을 갖기 위해 발생하는 2개의 파일들이 생성될 때마다 그 체크의 단일 카피만이 메모리에 물리적으로 저장되고, 그에 의해 저장 공간의 소비를 감소시키고, 사용자가 그렇지 않으면 만들지 못했을 카피들을 만들 수 있게 하는 이점을 갖는다. 예를 들어, 라이브러리에 대한 디렉토리에서의 링크를 만드는 대신에, (예를 들어, 프로세서를 이용하는) 사용자는 단순히 전체 라이브러리를 그 디렉토리에 카피할 수 있고, 가상 카피는 실제 카피보다 적은 스토리지를 이용할 것이다.

[0241] 일부 경우들에서, 사용자가 파일 또는 디렉토리를 (심지어 하나의 볼륨으로부터 다른 볼륨으로) 카피하는 경우, 물리적으로 (예를 들어, 프로세서를 이용하는) 시스템은 실제 체크들 자체가 아니라, 엔티티의 루트 체크에 대한 포인터들만을 카피한다. 여기서, "포인터"는 다른 엔티티들의 메타데이터 및 데이터베이스에 저장되는 엔티티의 해시를 의미할 수 있다. 디렉토리들 /X/ 및 /Y/ 둘다가 파일명 N을 갖는 파일 A를 포함하고, 사용자가 /X/N을 수정한다면, 시스템은 B를 A의 카피로서 만들고, B에 대한 요청된 수정을 행하고, 포인터를 /X/N으로부터

터 B를 가리키게 변경하고, 원래의 /Y/N을 여전히 A를 가리키게 남겨둘 것이다.

- [0242] 처음에, 상기의 것은 설명된 바와 같이 코딩될 수 있다. 그것은 각각의 기입이 해시들을 재계산하고 새로운 물리적 파일들에 저장되는 새로운 청크들을 생성하는 것을 수반할 수 있음을 의미한다. 일부 경우들에서, 시스템은 RAM에서의 기입들을 캐싱하고, 지연 후에 디스크 메모리에만 그들을 기입하는 것에 의해 더 빠르게 만들어질 수 있어서, 동일한 파일에 대한 다수의 기입들이 해싱 및 청크 생성의 단일 단계 내로 합쳐질 것이다. 추가적으로, 일부 그러한 경우들에서, 단일 디렉토리에서 변경하는 다수의 파일들은 디렉토리 트리에서의 그 디렉토리 및 그 디렉토리 위의 것들에 대한 단일의 변경을 초래할 것이다. 그 경우에, 판독들 이전에, 판독된 값에 영향을 미칠 임의의 캐싱된 기입들이 현재 존재하는지를 알아보기 위해 데이터베이스에 질의하는 것이 선행된다.
- [0243] 일부 구현들에서, 파일을 구성하는 청크들은 사용자가 볼 수 없으며, 오히려, 사용자는 단지 파일 자체를 보게 된다. 그러나, 다른 구현들에서, 시스템은 청크 해시들과 관련된 (예를 들어, 프로세서에서 실행되는) 여러 방법들을 수행하도록 구성될 수 있다. 시스템은 사용자가 파일 또는 디렉토리의 루트 청크의 해시를 찾는 것을 허용할 수 있다. 이것은 그 파일 또는 디렉토리의 해시인 것으로 정의되고, 시스템 전체에 걸쳐 그 파일 또는 디렉토리에 대한 포인터로서 이용될 수 있다.
- [0244] 일부 구현들에서, 사용자는 바이트들의 스트림에 대해 파일 또는 디렉토리 트리 또는 청크들의 리스트(그들의 해시들에 의해 참조됨)를 직렬화하기 위한 (예를 들어, 프로세서에 의해 실행되는) 방법을 호출할 수 있다. 대안적으로 또는 추가적으로, 일부 구현들에서, 사용자는 그러한 스트림을 역직렬화하고, 사용자에게 의해 선택된 디렉토리 내의 새로운 파일 또는 디렉토리를 생성하기 위한 (예를 들어, 프로세서에 의해 실행되는) 방법을 호출할 수 있다. 역직렬화된 청크들은 파일시스템에 투명하게/비가시적으로 추가될 수 있고, 예를 들어, 전체 파일 또는 디렉토리가 존재하는 청크들을 가질 때 후속하여 가시적 파일들 및 디렉토리 트리들로 변환될 수 있다. 사용자가 그들이 역직렬화가 수행되었음을 나타낼 때, 시스템은 이용되지 않은 임의의 청크들을 가비지 수집할 수 있다. 그러한 가비지 수집은 또한, 주기적으로(예를 들어, 1분에 1회, 1시간에 1회, 1일에 1회 등), 산발적으로, 또는 임의의 다른 미리 결정된 시간 또는 시간 간격으로 발생할 수 있다.
- [0245] 사용자가 파일에 대해 (예를 들어, 프로세서에 의해 실행되는) 직렬화 방법을 호출하는 경우, 프로세서는 전체 파일(즉, 깊이 우선으로(depth-first order), 전위 순회로(pre-order traversal), 전체 청크 트리)을 직렬화하고, 각 청크의 콘텐츠를 출력할 수 있다. 일부 구현들에서, 청크 이름들 중 어느 것도 직렬화되지 않는다. 사용자가 디렉토리 트리를 직렬화하도록 요청하는 경우, 프로세서는 전체 디렉토리 트리(즉, 디렉토리 트리에서의 파일들 및 서브디렉토리들의 청크 트리들)를 직렬화하여, 그 디렉토리들의 깊이 우선 검색을 행할 수 있다. 두 경우들에서, 직렬화는 재귀적으로 작동하고, 루트 청크를 직렬화하고, 이후 그것이 포함하는 해시들 각각을 직렬화하는 등으로 된다. 시스템은 지금까지 직렬화된 청크들의 표시를 저장하고/하거나 이를 기억하므로, 프로세서는 동일한 청크를 2번 직렬화하지 않는다.
- [0246] 역직렬화시에, 시스템은 (예를 들어, 그 청크를 저장하기 위한 파일명을 결정하기 위해) 그의 콘텐츠로부터 각각의 청크의 해시를 계산하고, 프로세스가 완료된 후에, 완전한 엔티티의 유효한 부분인 것으로 되지 않은 임의의 청크들을 폐기할 수 있다.
- [0247] 시스템이 (그들의 해시들의 세트에서 패스함으로써) 청크들의 세트를 직렬화하는 경우, 시스템은 주어진 순서로 그 청크들에서의 바이트들만을 직렬화한다. 방법은 청크들의 세트에 의해 참조된 청크들의 세트를 반환한다. 사용자가 그러한 청크들의 세트를 역직렬화하는 경우, 청크들은 시스템에 추가되고, 시스템은 사용자의 컴퓨터 상에 이미 존재하지 않는 청크들의 세트에 의해 참조된 청크들의 세트를 반환한다.
- [0248] 시스템(또는 프로세서)은 또한 파일 또는 디렉토리 트리의 비-리프 청크들을 직렬화하기 위한 방법을 실행할 수 있다. 역직렬화시에, 시스템은 사용자의 시스템 상에 이미 존재하지 않는 리프 청크들의 해시들을 사용자에게 반환한다. 따라서, 예를 들어, (컴퓨팅 디바이스 Alice에서의 프로세서를 이용하는) Alice는 Bob으로(컴퓨팅 디바이스 Bob에서의 프로세서로) 리프들을, 그 다음 비-리프들을 직렬화할 수 있고, Bob은 Alice로부터 Bob으로 얼마나 많은 바이트들이 전송될 것인지를 정확하게 가능한 빨리 조기에 학습할 것이다. 또는, 대안적인 시스템에서, Alice는 Bob으로 비-리프 청크들을 직렬화할 수 있고, Bob은 Bob이 아직 수신하지 않은 리프들의 해시들을 Alice에게 통지할 수 있고, 그 다음 Alice는 Bob이 아직 수신하지 않은 리프들만을 직렬화할 수 있다.
- [0249] 청크 세트 직렬화의 하나의 이점은 Alice가 Bob에게 Bob이 이미 갖고 있는 많은 파일들을 포함하고 있는 디렉토리 트리를 효율적으로 전송하는 것을 허용한다. Alice는 먼저 루트 청크의 해시를 직렬화하고, 그것을 전송한다. Bob은 루트 청크의 해시를 그가 선택하는 디렉토리 내로 역직렬화한다. (예를 들어, Bob에서의 프로세서

에 의해 실행되는) 그 방법 호출이 해시들의 비어 있지 않은 리스트를 반환한다면, Bob은 해당 리스트를 Alice에게 전송하고, Alice는 그 청크들의 직렬화로 다시 전송한다. 그 다음, Bob은 청크들을 역직렬화하고, 그들이 그가 여전히 갖고 있지 않다고 참조하는 청크들의 리스트를 Alice에게 전송한다. 이것은 Bob이 전체 디렉토리 트리의 카피를 가질 때까지 계속된다. 이러한 방식으로, Alice는 Bob이 이미 갖고 있는 청크는 전송하지 않으면서, 전체 디렉토리 트리를 전송한다.

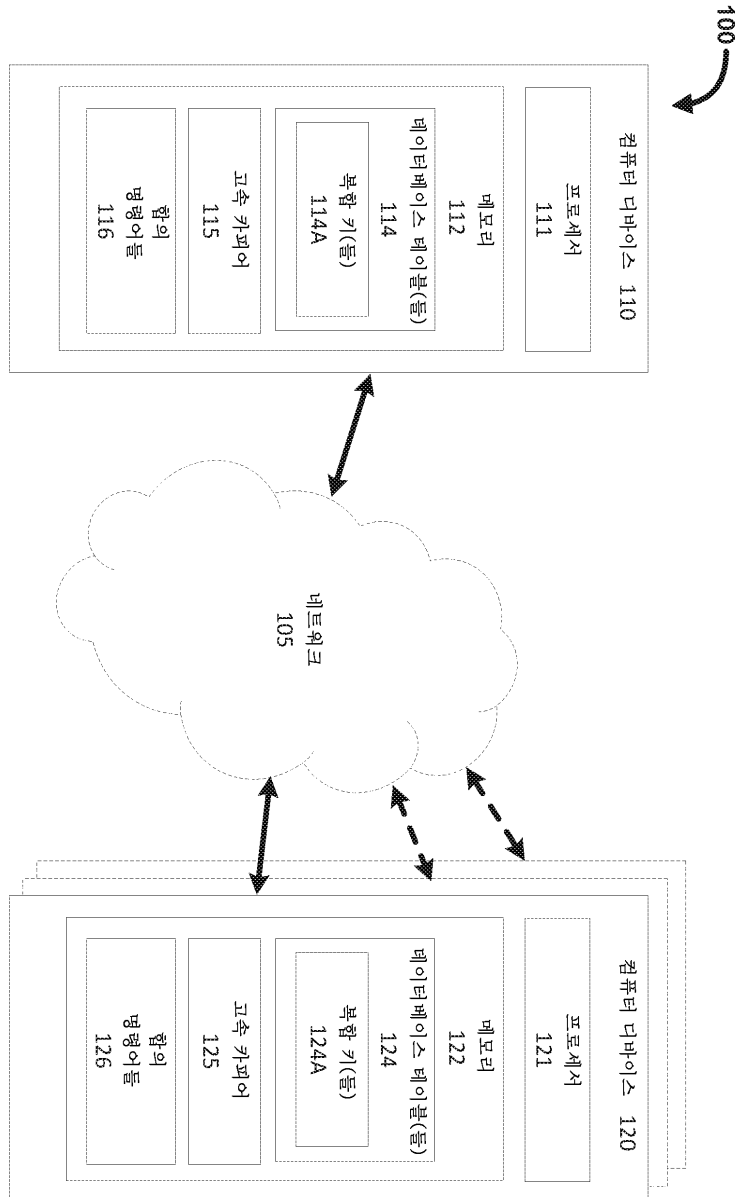
- [0250] 일부 경우들에서, 역직렬화 동안, 관계형 데이터베이스는 새로운 청크들, 파일들 및 디렉토리들의 메타데이터를 추적하고/하거나 저장할 수 있다. 일부 경우들에서, 임의의 주어진 시간에, 사용자는 일단 모든 그의 청크들이 (그리고 그의 서브디렉토리들의 청크들이 재귀적으로) 수신되었다면, 파일시스템에 존재하는 파일 또는 디렉토리를 볼 수 있다.
- [0251] 일부 경우들에서, 시스템(또는 프로세서)에 의해 실행되는 deserializeDone 방법은 사용자가 역직렬화를 수행하였음을 나타낸다. 그 시점에서, 시스템은 유효 엔티티의 일부로서 종료되지 않은 임의의 청크들을 폐기할 수 있다. 역직렬화시에, 사용자는 새로운 파일 또는 디렉토리 트리가 주어진 WRITE 허가를 갖도록 설정될 것을 요청할 수 있다. 이것은 또한 카피할 때 수행될 수 있다.
- [0252] 고속 카피가능 파일시스템 플랫폼은 다음의 단계들을 포함하는 동작을 주기적으로 수행할 수 있다:
- [0253] - 마지막 서명된 상태 이후에 현재 이벤트들의 리스트 및 파일시스템 데이터베이스의 스냅샷을 취하는 단계
- [0254] - 그 스냅샷을 디스크 메모리로 직렬화하는 단계
- [0255] - 이들 새로운 파일들이 이제 공식적인 백업이라는 것을 나타내기 위해 디스크 메모리 상의 바이트를 변경하는 단계
- [0256] 고장 후에(또는 심지어 정상적인 셧다운(shutdown) 및 리부트(reboot) 후에), 고속 카피가능 파일시스템 플랫폼은 다음을 수행할 수 있다:
- [0257] - 가장 최근의 서명된 상태를 로딩하는 것
- [0258] - 가장 최근의 공식적인 백업으로부터의 이벤트들을 역직렬화하는 것
- [0259] - 가장 최근의 공식적인 백업으로부터 데이터베이스를 역직렬화하는 것
- [0260] - 정상적인 고속 카피가능 파일시스템 플랫폼을, 동기화 등과 함께 실행하는 것을 시작하는 것
- [0261] 또한, 일부 구현들에서, 청크 가비지 수집을 위해 배경 스레드가 실행될 수 있다. 가비지 수집 스레드는 주기적으로 청크 디렉토리들을 검토하고, 그 파일명들을 데이터베이스에 대해 체크할 수 있다. 가비지 수집 스레드가 현재 볼륨들 중 임의의 것에 의해 참조되지 않은 청크를 식별하고, 가장 최근의 서명된 상태에 의해 참조되지 않으면, 가비지 수집 스레드는 하드 드라이브로부터 그 청크를 삭제해야 한다. 또한, 일부 경우들에서, 가비지 수집 스레드는 기존 청크들에서 주기적으로 관독하고, 각각이 그의 콘텐츠의 해시와 매칭되는 파일명을 갖는지를 체크할 수 있다. 그들이 매칭되지 않으면, 청크는 하드 드라이브로부터 삭제될 수 있다. 청크가 (데이터베이스에 따라) 여전히 필요한 경우, 청크를 DHT로부터 또는 다른 멤버로부터 재다운로드하기 위해 단계들이 취해질 수 있다.
- [0262] 본 명세서에서 설명된 일부 실시예들 및/또는 방법들은 (하드웨어 상에서 실행되는) 소프트웨어, 하드웨어, 또는 이들의 조합에 의해 수행될 수 있다. 하드웨어는, 예를 들어, 범용 프로세서, 필드 프로그래밍가능한 게이트 어레이(field programmable gate array)(FPGA), 및/또는 애플리케이션 특정 집적 회로(application specific integrated circuit)(ASIC)를 포함할 수 있다. (하드웨어 상에서 실행되는) 소프트웨어는 C, C++, Java™, Ruby, Visual Basic™ 및/또는 다른 객체-지향형, 절차적 또는 다른 프로그래밍 언어를 포함하는 다양한 소프트웨어 언어들(예를 들어, 컴퓨터 코드) 및 개발 툴들로 표현될 수 있다. 컴퓨터 코드의 예들은, 제한적인 것은 아니지만, 마이크로-코드 또는 마이크로-명령어들, 컴파일러(compiler)에 의해 생성된 것과 같은 머신 명령어들, 웹 서비스를 생성하기 위하여 이용된 코드, 및 인터프리터(interpreter)를 이용하여 컴퓨터에 의해 실행되는 하이-레벨 명령어들을 포함하는 파일들을 포함한다. 예를 들어, 실시예들은 명령적 프로그래밍 언어들(예를 들어, C, Fortran 등), 함수 프로그래밍 언어들(Haskell, Erlang 등), 논리 프로그래밍 언어들(예를 들어, Prolog), 객체-지향 프로그래밍 언어들(예를 들어, Java, C++ 등) 또는 다른 적절한 프로그래밍 언어들 및/또는 개발 툴들을 이용하여 구현될 수 있다. 컴퓨터 코드의 추가적인 예들은, 제한적인 것은 아니지만, 제어 신호들, 암호화된 코드 및 압축된 코드를 포함한다.

[0263]

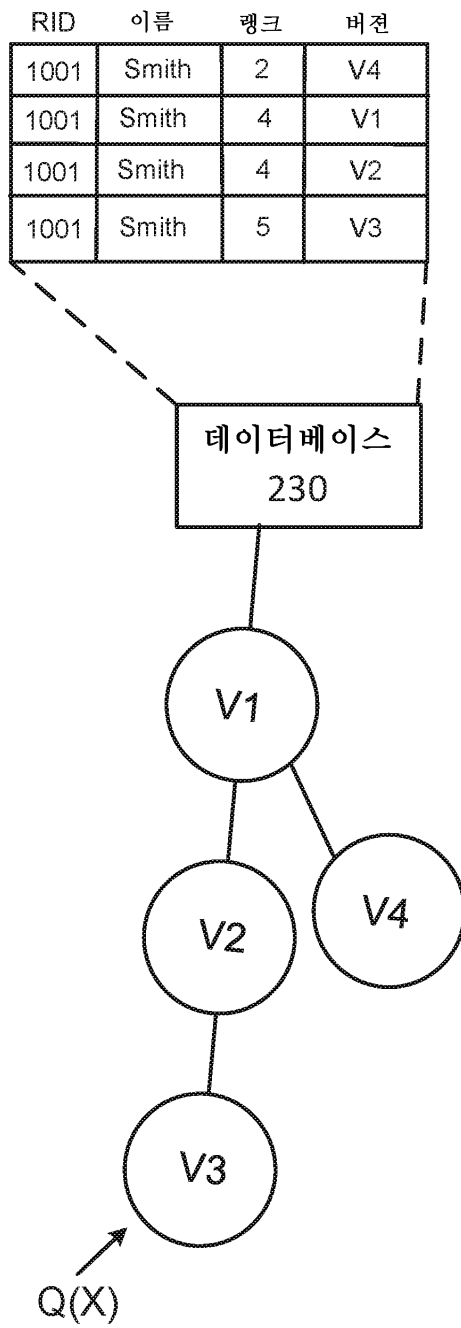
용어들 "명령어들" 및 "코드"는 임의의 타입의 컴퓨터 판독가능 문(들)을 포함하도록 넓게 해석되어야 한다. 예를 들어, 용어들 "명령어들" 및 "코드"는 하나 이상의 프로그램, 루틴, 서브루틴, 함수, 절차 등을 지칭하는 것일 수 있다. "명령어들" 및 "코드"는 단일의 컴퓨터 판독가능 문 또는 다수의 컴퓨터 판독가능 문들을 포함할 수 있다.

도면

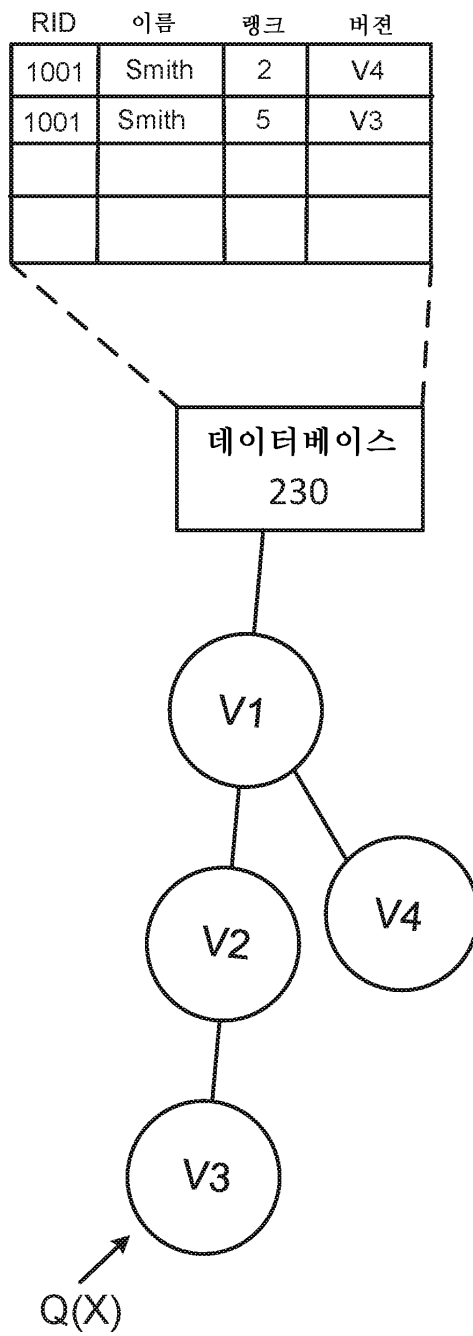
도면1



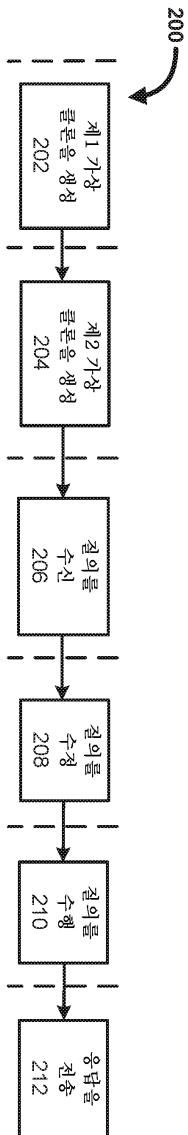
도면2a



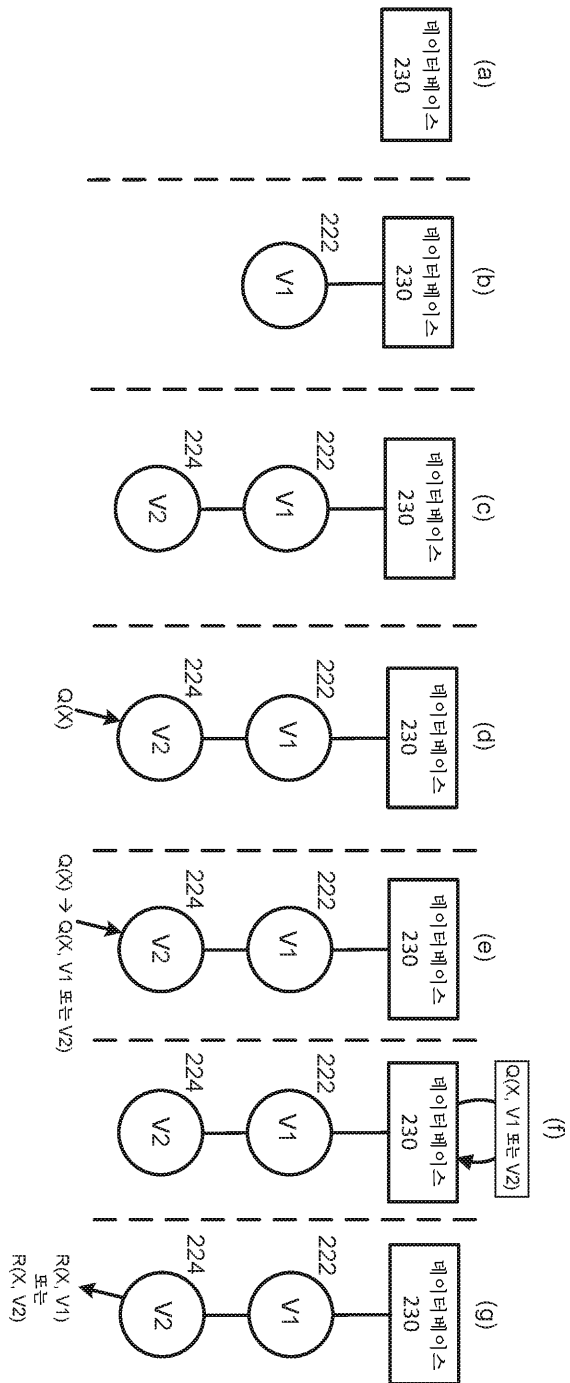
도면2b



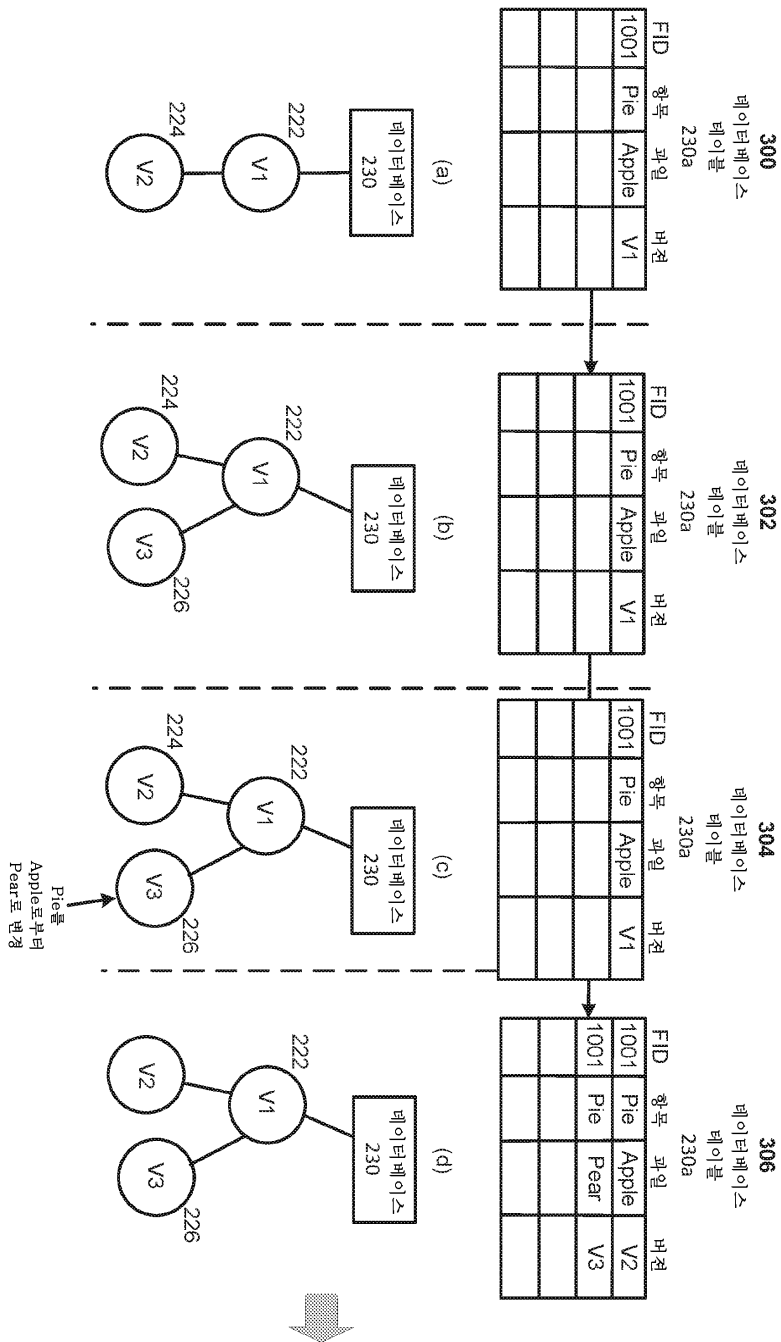
도면3a



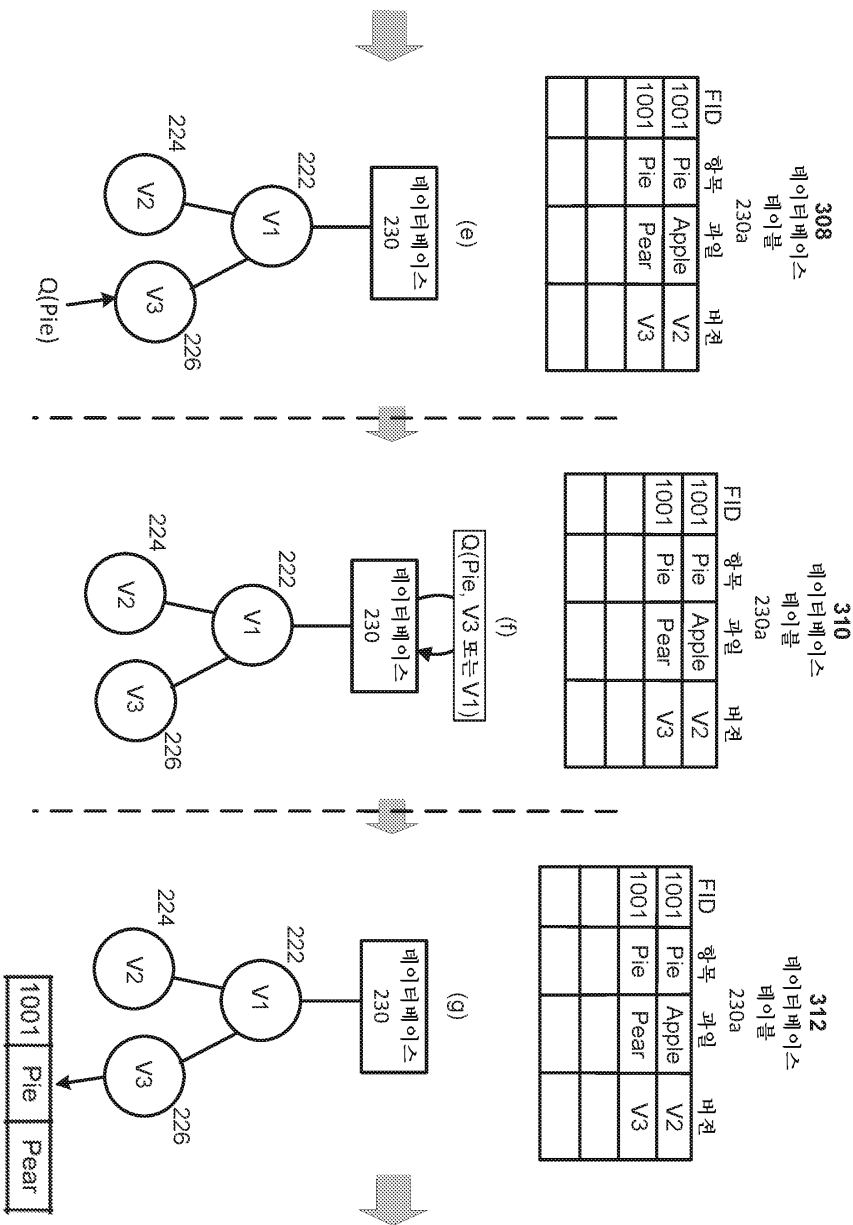
도면3b



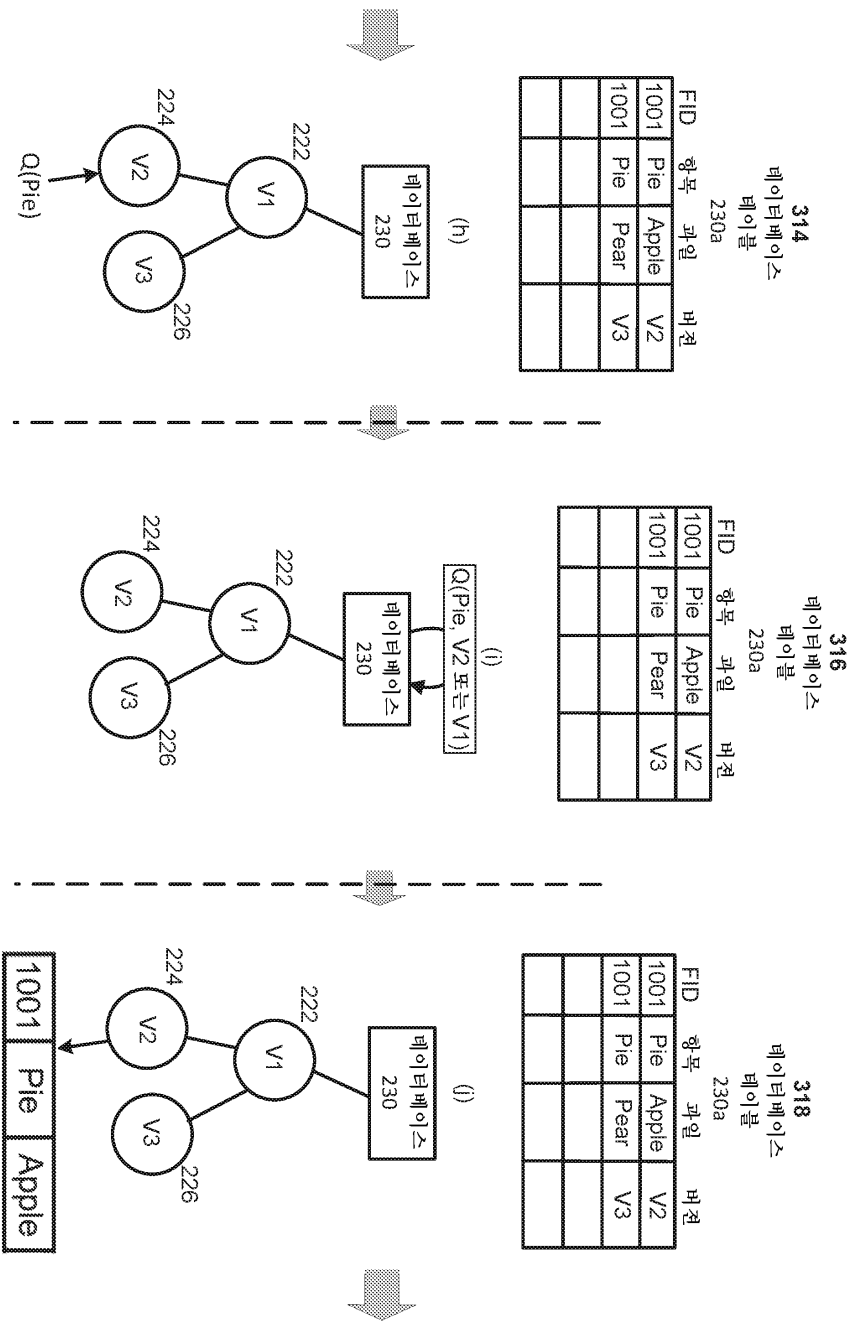
도면4a



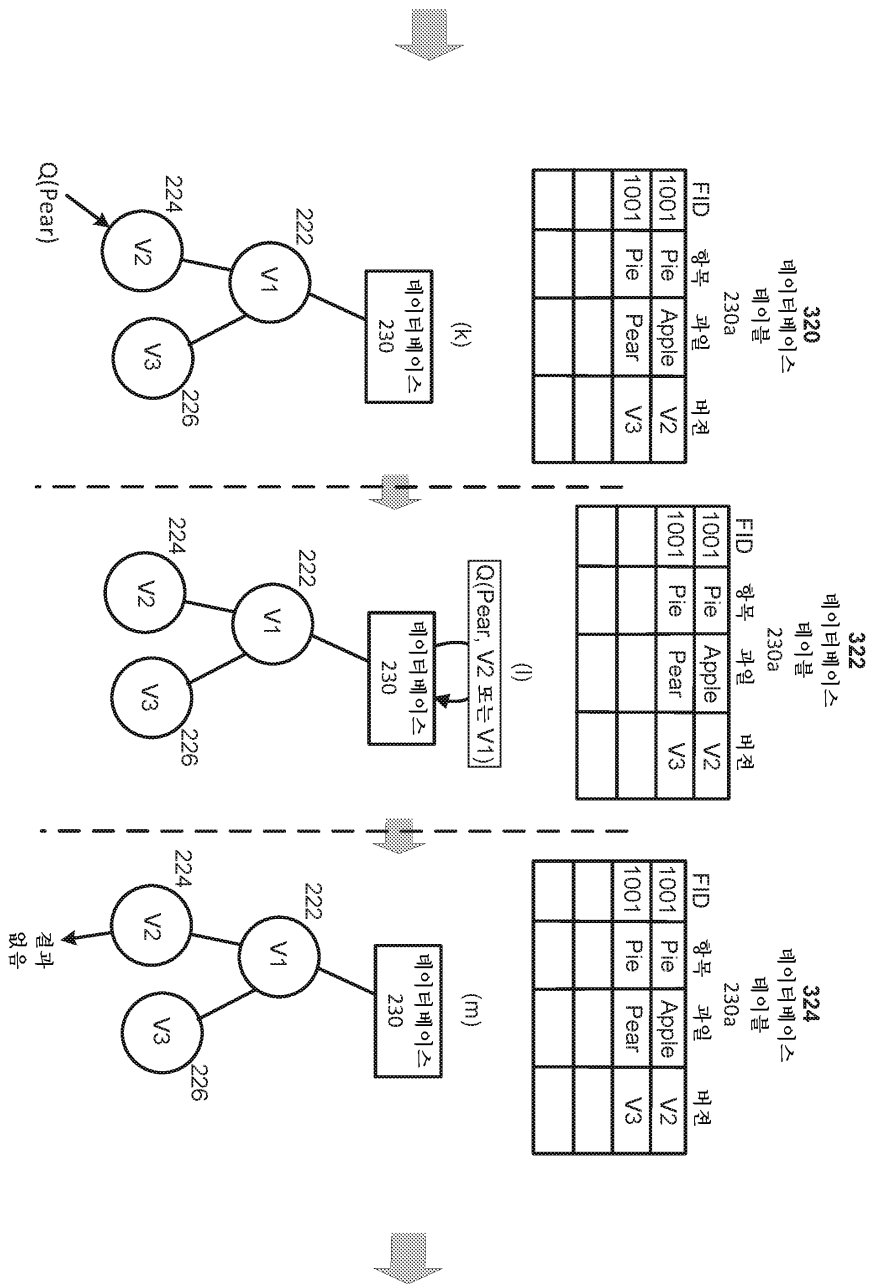
도면4b



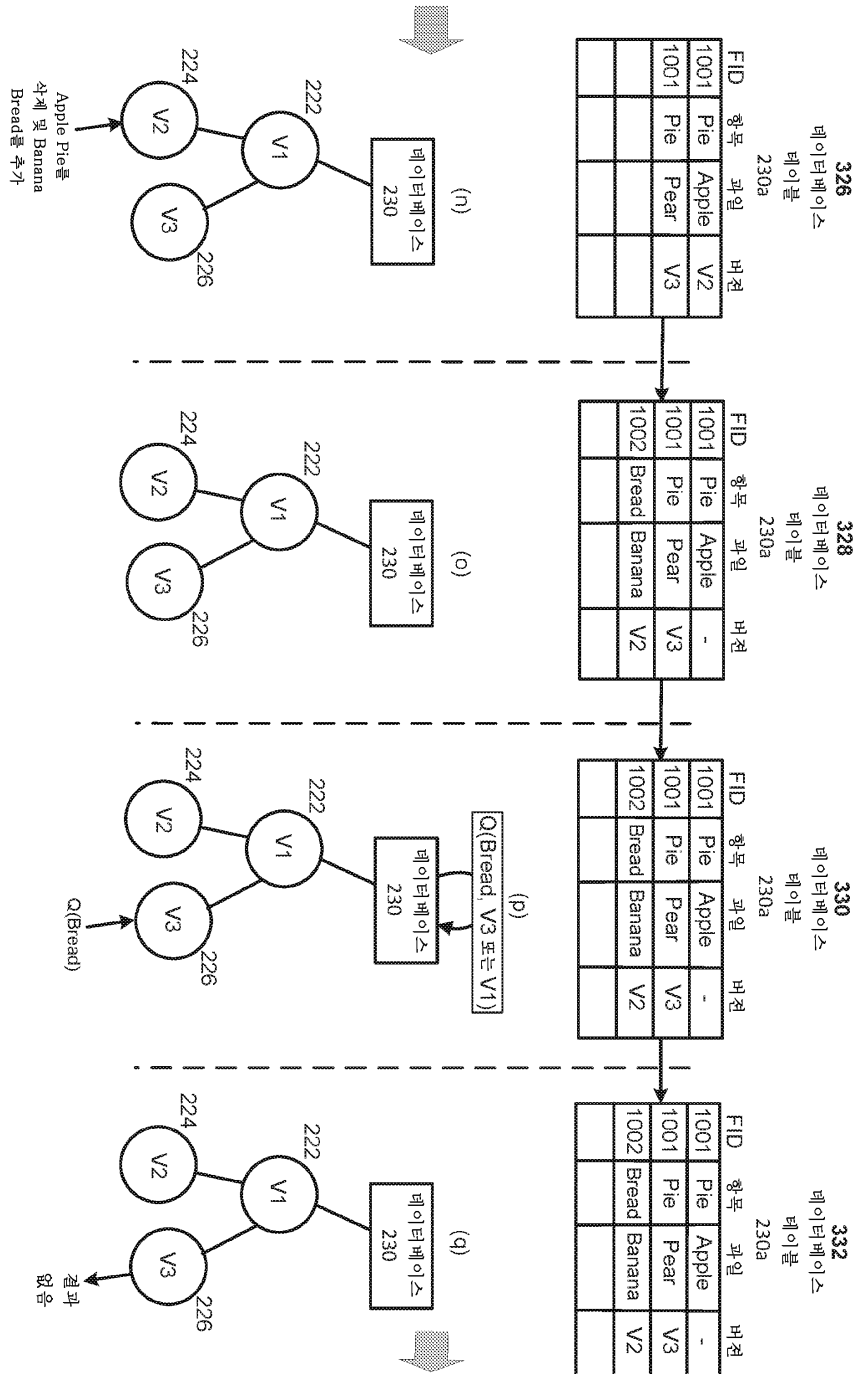
도면4c



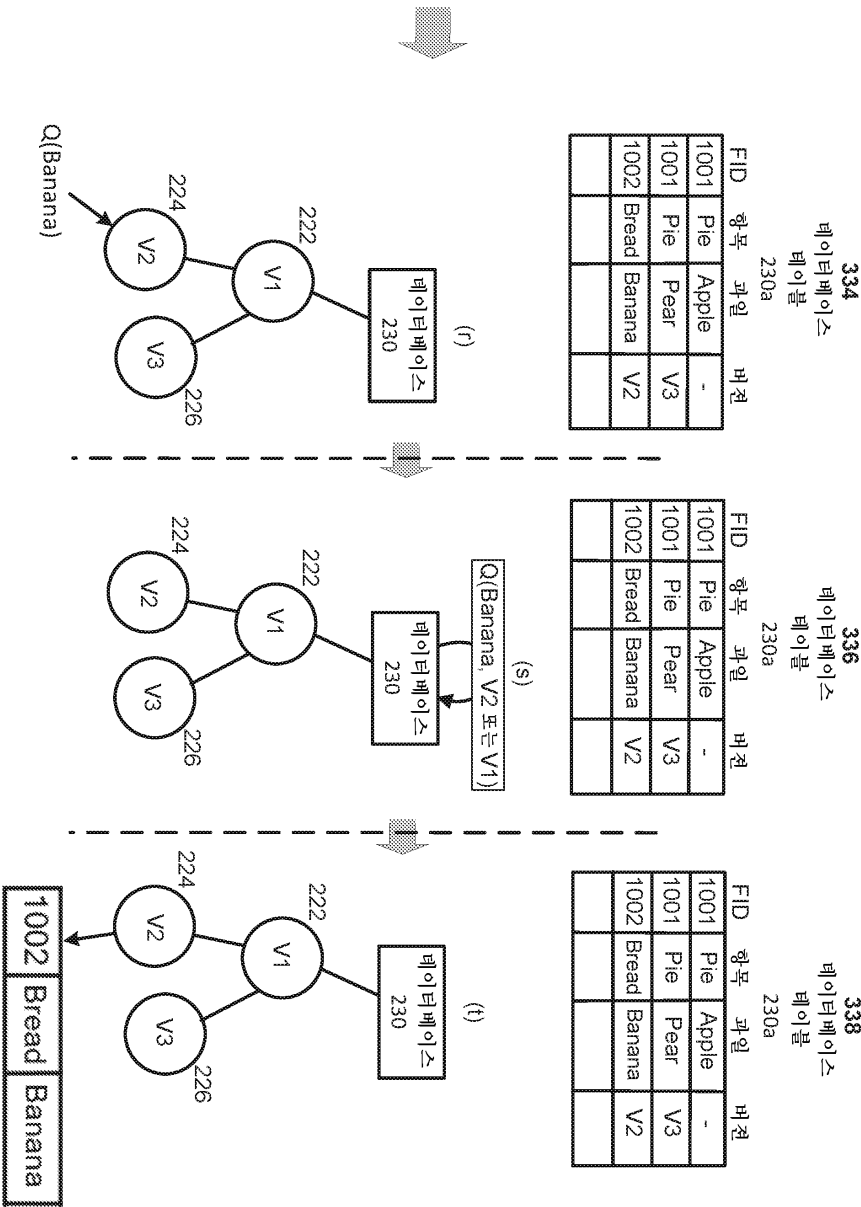
도면4d



도면4e



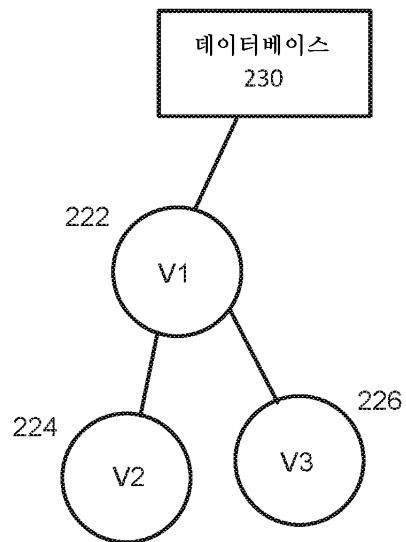
도면4f



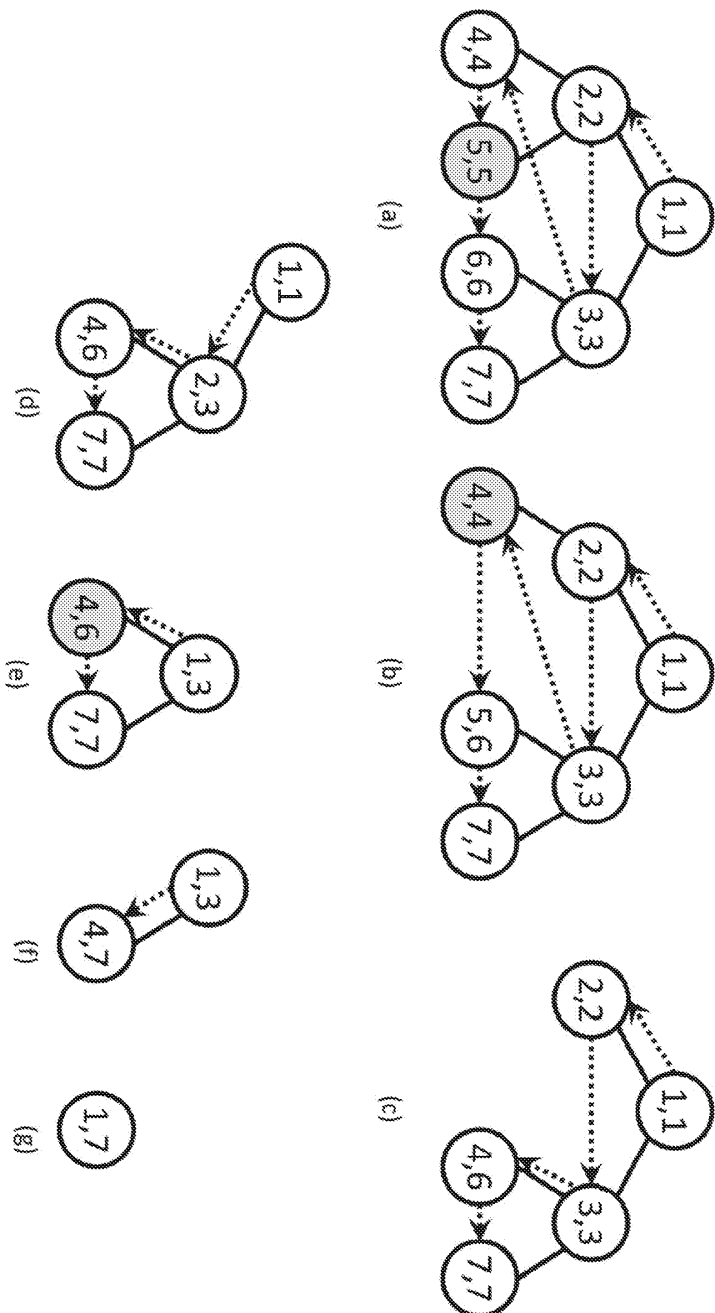
도면4g

340
데이터베이스 테이블
230a

FID	항목	과일	버전
1001	Pie	Apple	V1
1001	Pie	Pear	V3
1002	Bread	Banana	V2

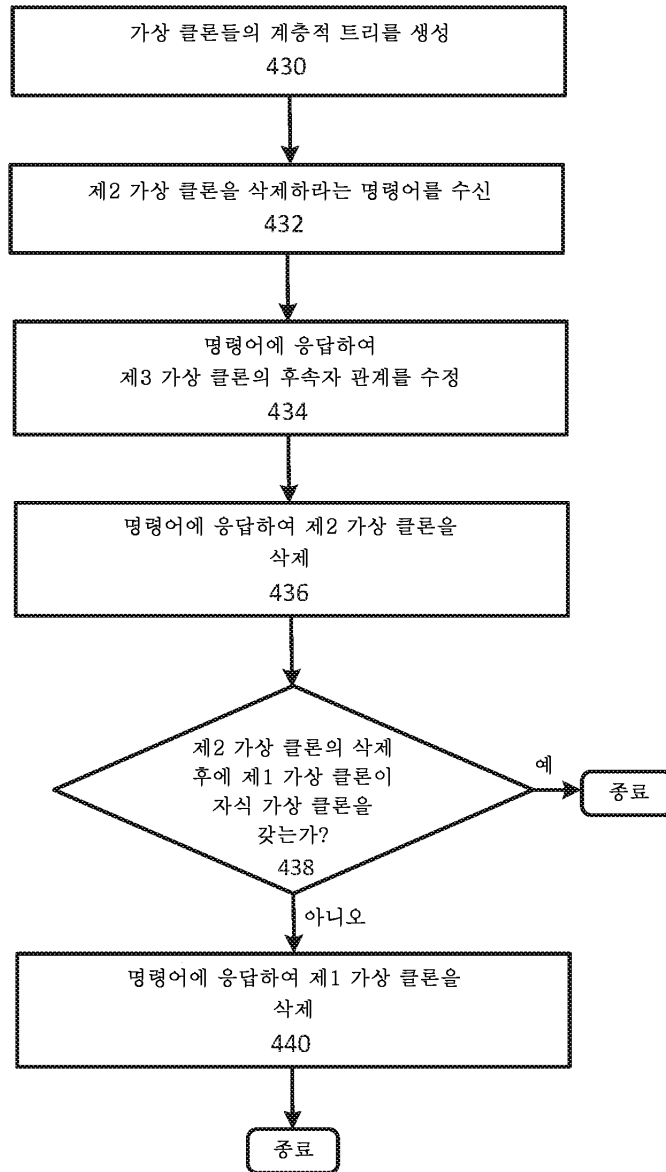


도면5



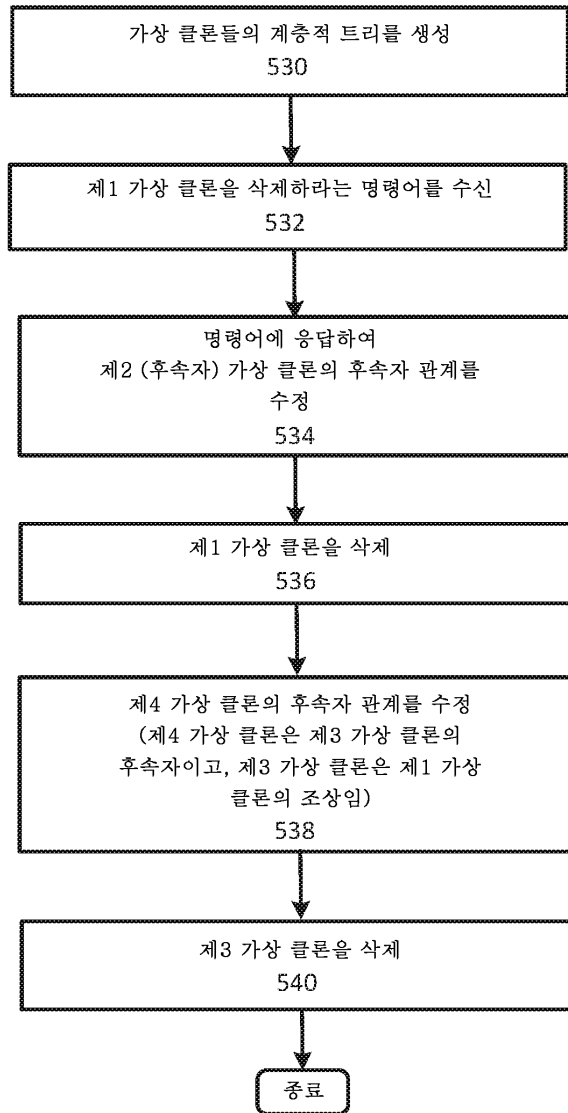
도면6

400

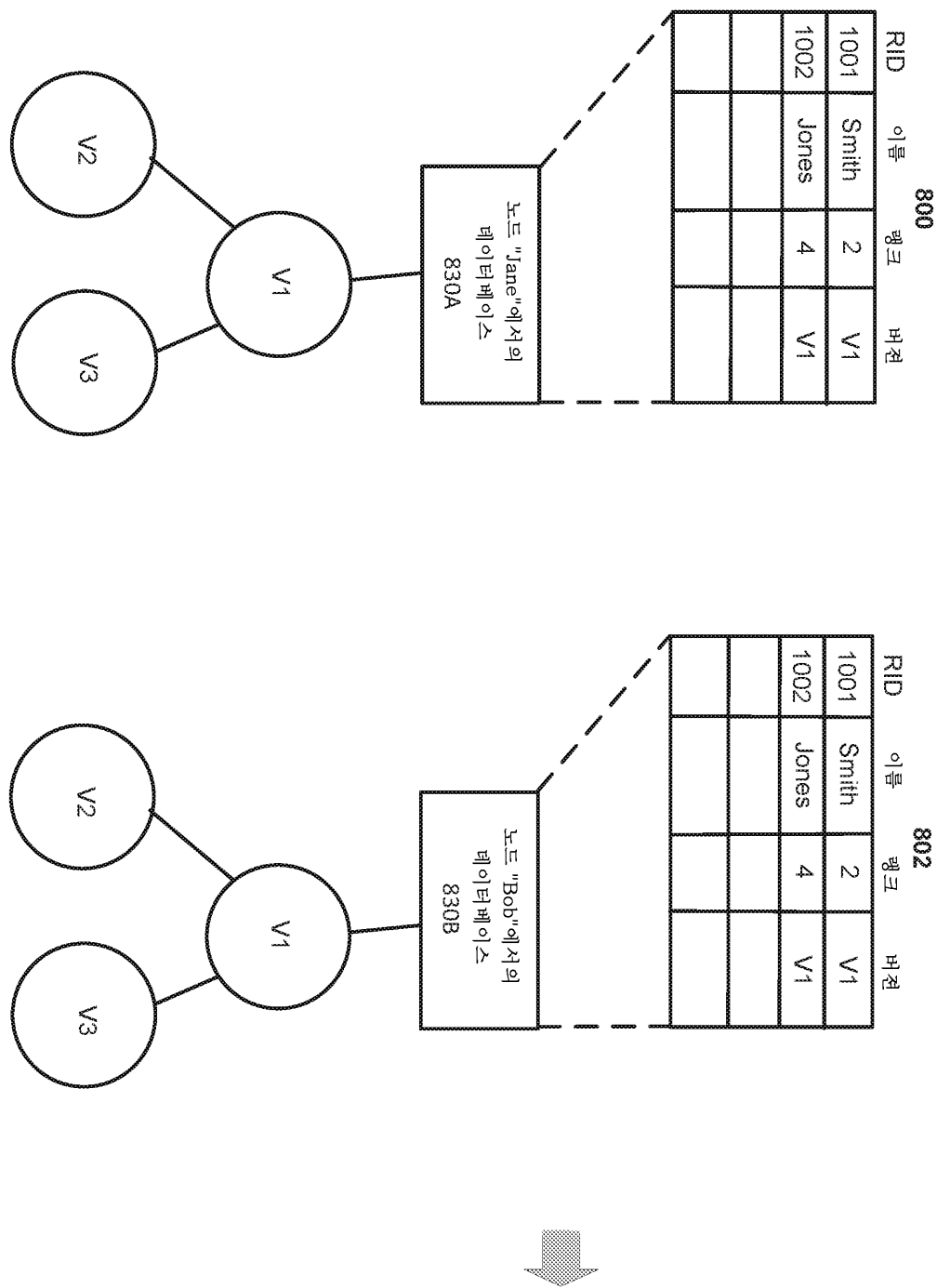


도면7

500

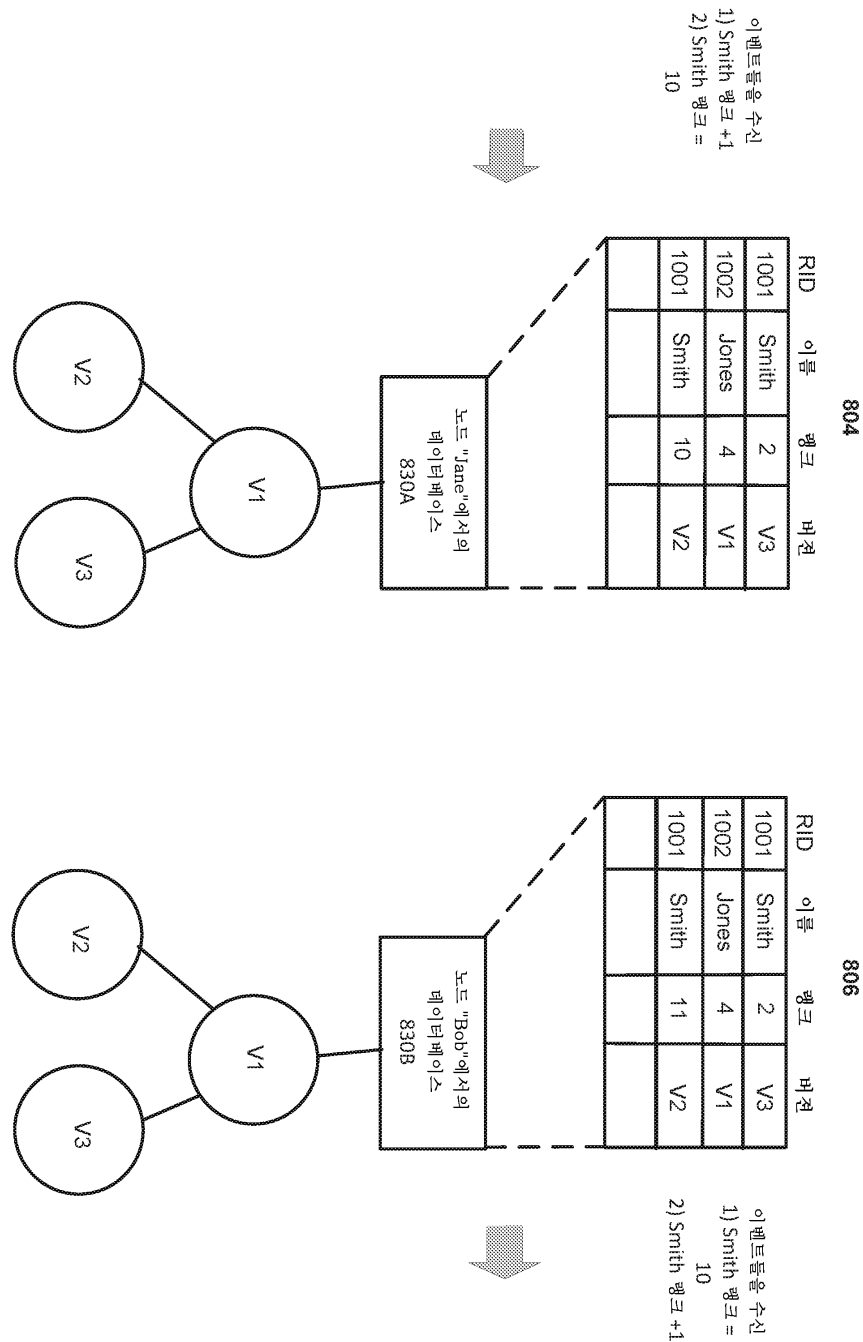


도면8a



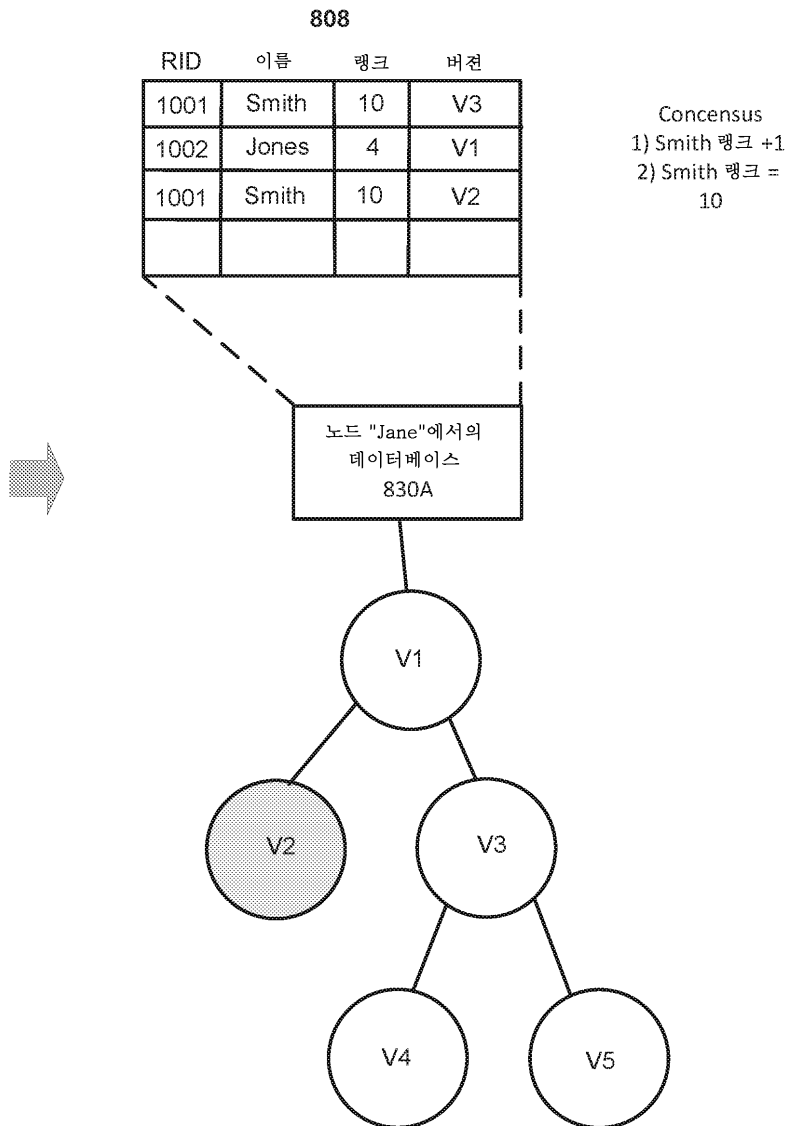
동기화

도면8b



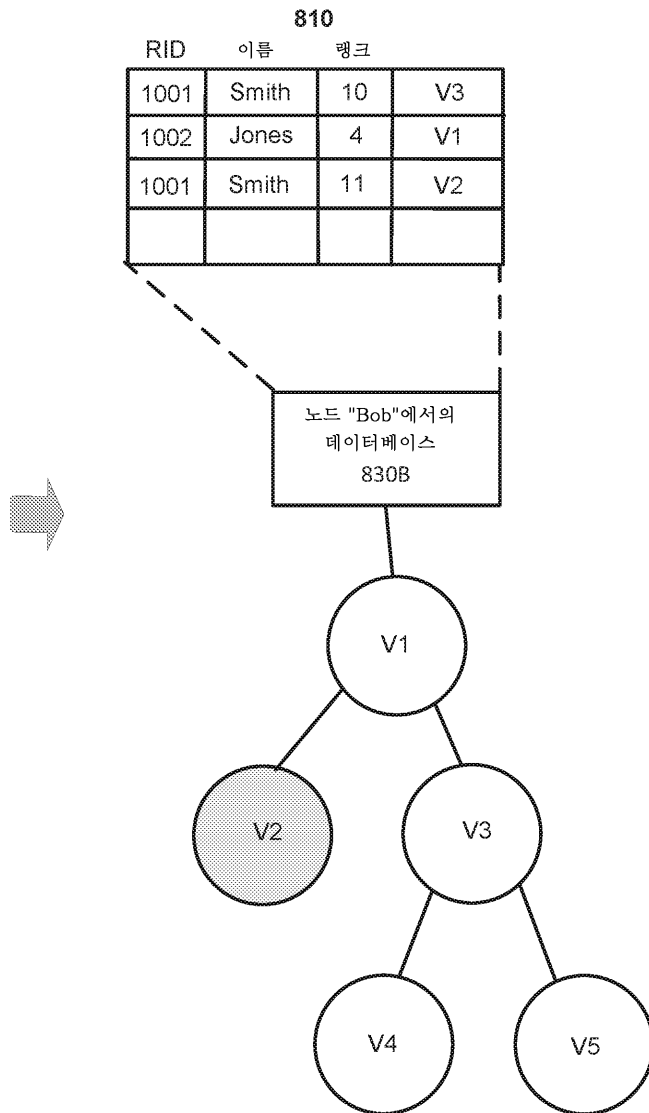
도면8c

합의 (JANE)



도면8d

합의 (BOB)



도면9

