(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2012년11월14일
(11) 등록번호 10-1201061
(24) 등록일자 2012년11월07일

(51) 국제특허분류(Int. Cl.)
G06F 15/16 (2006.01) *G06F 13/00* (2006.01)
(21) 출원번호 10-2006-0002188
(22) 출원일자 2006년01월09일
심사청구일자 2010년12월30일
(65) 공개번호 10-2006-0095449
(43) 공개일자 2006년08월31일
(30) 우선권주장
11/195,320 2005년08월02일 미국(US)
60/657,522 2005년02월28일 미국(US)
(56) 선행기술조사문헌
KR1020050000352 A

(73) 특허권자
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
휘튼, 아서 티.
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내
알바해리, 벤자민
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내
(뒷면에 계속)
(74) 대리인
제일특허법인

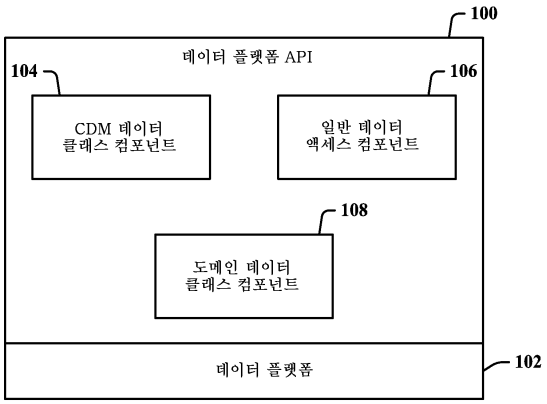
전체 청구항 수 : 총 15 항 심사관 : 이상헌

(54) 발명의 명칭 **공통 데이터 플랫폼에 대한 스토리지 ＡＰＩ**

(57) 요 약

데이터 플랫폼에 대한 애플리케이션 프로그램 인터페이스(API)가 제공된다. API는 데이터 플랫폼의 스토어, 세션, 트랜잭션 및 쿼리 서비스 중 적어도 하나를 노출시키는 일반 데이터 액세스 컴포넌트를 포함하며, 이 데이터 플랫폼은 데이터 스토어와 연관되어 있다. API의 데이터 클래스 컴포넌트는 데이터 플랫폼의 데이터 모델의 유형 및 관계를 노출시키는 표준형의 애플리케이션 독립적인 클래스를 제공한다. API는 데이터 플랫폼의 도메인 관련 속성 및 거동을 노출시키는 애플리케이션 관련 클래스 및 프레임워크 관련 클래스의 도메인 데이터 클래스 컴포넌트를 포함한다. 데이터 플랫폼은 복수의 개별적인 애플리케이션 프레임워크에 의해 액세스가능한 데이터 서비스를 제공하기 위해 데이터 스토어와 인터페이스하는 공통 데이터 플랫폼일 수 있으며, 이 데이터 서비스는 서로 다른 프레임워크의 대응하는 애플리케이션이 데이터 스토어에 액세스할 수 있게 해준다.

대 표 도 - 도1



(72) 발명자

쉐퍼드, 에드워드 지.

미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

딤, 마이클 이.

미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

피조, 마이클 제이.

미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

나가라잔, 라메시

미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

특허청구의 범위

청구항 1

데이터 플랫폼에 대한 애플리케이션 프로그래밍 인터페이스를 제공하기 위해 컴퓨팅 시스템에 이용하기 위한 컴퓨터 판독 가능 저장 매체로서 - 상기 애플리케이션 프로그래밍 인터페이스는 복수의 정보 포맷에 일반적인 방식(generic manner)으로의 액세스를 허용함 - ,

상기 컴퓨터 판독 가능 저장 매체는 컴퓨터 실행 가능 명령어들을 저장하는 하나 이상의 저장 요소(storage element)를 포함하며,

상기 컴퓨터 실행 가능 명령어들은, 데이터 플랫폼에 대한 일반(generic) 애플리케이션 프로그래밍 인터페이스(API)를 포함하며,

상기 일반 API는,

상기 데이터 플랫폼의 스토어들, 세션들, 트랜잭션들 및 쿼리 서비스들의 각각을 노출시키는 일반 데이터 액세스 컴포넌트(generic data access component) - 상기 데이터 플랫폼은 데이터 스토어와 연관되어 있음 - ;

상기 데이터 플랫폼의 데이터 모델의 유형들 및 관계들을 노출시키는 표준형의 애플리케이션 독립적인 클래스(canonical, application-independent class)의 데이터 클래스 컴포넌트(data class component);

상기 데이터 플랫폼의 도메인 고유 속성 및 거동을 노출시키는 애플리케이션 고유 클래스 및 프레임워크 고유 클래스의 도메인 데이터 클래스 컴포넌트(domain data class component) - 도메인은 애플리케이션과는 상이함 - ;

스키마의 테이블들에 대한 액세스를 제공하는 스키마 클래스 - 상기 스키마 클래스는 유형 미지정 스키마 클래스(untyped schema class)로부터 파생됨 - ;

상기 스키마 내에 정의된 테이블들에 대해 강타입(strongly typed) 액세스를 제공하는 TableSet 클래스;

상기 API 내의 나머지 클래스들이 동작하는 스토어를 정의하는 StorageDomain 클래스;

세션에 대한 컨텍스트를 제공하며, 또한 현재 컨텍스트 내의 객체들에 대한 변경을 리프레쉬 또는 저장하는 메소드(method)들에 의해 식별자 관리(identity management), 변경 추적(change tracking) 또는 동시성 충돌 처리(concurrency conflict handling) 중 적어도 하나에 대한 스코프(scope)를 정의하는 StorageContext 클래스;

데이터 스토어에 대해 쿼리들을 구축(build)하도록 구성되는 StorageSearcher 클래스; 및

쿼리로부터의 결과 세트에 대한 애플리케이션 뷰를 제공하는 StorageView 클래스

를 포함하는 컴퓨터 판독 가능 저장 매체.

청구항 2

제1항에 있어서,

상기 데이터 플랫폼은 복수의 서로 다른(disparate) 애플리케이션 프레임워크에 의해 액세스가능한 데이터 서비스들을 제공하기 위해 상기 데이터 스토어에 인터페이스하는 공통 데이터 플랫폼(common data platform)이고,

상기 데이터 서비스는 서로 다른 프레임워크들의 대응 애플리케이션이 상기 데이터 스토어에 액세스할 수 있게 해주는 컴퓨터 판독 가능 저장 매체.

청구항 3

제1항에 있어서,

상기 도메인 데이터 클래스 컴포넌트는 다른 클래스들이 동작하고 있는 스토어를 정의하는 도메인 클래스(domain class)를 포함하는 컴퓨터 판독 가능 저장 매체.

청구항 4

제1항에 있어서,

상기 데이터 클래스 컴포넌트는 세션에 대한 컨텍스트를 제공하는 컨텍스트 클래스(context class)를 포함하는 컴퓨터 판독 가능 저장 매체.

청구항 5

제4항에 있어서,

상기 컨텍스트 클래스는 현재의 컨텍스트 내의 객체들에 대한 변경을 리프레쉬 또는 저장하기 위한 메소드에 의해, 식별자 관리(identity management), 변경 추적(change tracking), 및 동시성 충돌 처리(concurrency conflict handling)에 대한 스코프를 정의하는 컴퓨터 판독 가능 저장 매체.

청구항 6

제1항에 있어서,

상기 데이터 클래스 컴포넌트는 상기 데이터 스토어에 대해 합성가능 객체 기반 쿼리(composable object-based query)를 구축하는 데 이용되는 탐색기 클래스(searcher class)를 포함하는 컴퓨터 판독 가능 저장 매체.

청구항 7

제1항에 있어서,

상기 일반 API는 결과 세트에 대한 뷰를 제공하는 뷰 클래스를 더 포함하는 컴퓨터 판독 가능 저장 매체.

청구항 8

삭제

청구항 9

제1항에 있어서,

상기 스키마 클래스는 타겟 스키마에 기초하여 유형 미지정 스키마 클래스(untyped schema class)로부터 파생되는 강타입 스키마 클래스(strongly typed schema class)인 컴퓨터 판독 가능 저장 매체.

청구항 10

정보의 특정 포맷에 관계없이, 일반적인 방식으로 정보에 대한 액세스를 제공하기 위해 일반(generic) 애플리케이션 프로그래밍 인터페이스를 이용하여 데이터 플랫폼을 노출하는 컴퓨터 구현 방법으로서,

일반 API에 대한 액세스를 수행하는 단계;

상기 API에 의해,

상기 데이터 플랫폼의 스토어들, 세션들, 트랜잭션들 및 쿼리 서비스들의 각각을 노출시키는 단계 - 상기 데이터 플랫폼은 데이터 스토어와 연관되어 있음 - ;

상기 데이터 플랫폼의 데이터 모델의 유형들 및 관계들을 노출시키는 단계 - 하나 이상의 표준형의 애플리케이션 독립적인 클래스(canonical, application-independent class)를 이용함 - ;

상기 데이터 플랫폼의 도메인 고유 속성 및 거동을 노출시키는 단계 - 하나 이상의 애플리케이션 고유 클래스 및 프레임워크 고유 클래스를 이용하며, 도메인은 애플리케이션과는 상이함 - ;

스키마 클래스를 통해 스키마의 테이블들에 대한 액세스를 제공하는 단계 - 상기 스키마 클래스는 유형 미지정 스키마 클래스(untyped schema class)로부터 파생됨 - ;

상기 스키마 내에 정의된 테이블들에 대해 강타입 액세스를 제공하는 단계 - 액세스는 TableSet 클래스를 통해 제공됨 - ;

상기 API 내의 나머지 클래스들이 동작하는 스토어를 정의하는 StorageDomain 클래스를 이용하는 단계;

StorageContext 클래스를 이용하여 세션에 대한 컨텍스트를 제공하는 단계 - 상기 StorageContext 클래스는,

현재 컨텍스트 내의 객체들에 대한 변경을 리프레쉬 또는 저장하는 메소드(method)들에 의해 식별자 관리(identity management), 변경 추적(change tracking) 또는 동시성 충돌 처리(concurrency conflict handling) 중 적어도 하나에 대한 스코프(scope)를 더 정의함 - ;

StorageSearcher 클래스를 이용하여 상기 데이터 스토어에 대해 쿼리들을 구축하는 단계; 및

StorageView 클래스를 이용하여 쿼리로부터의 결과 세트에 대한 애플리케이션 뷰를 제공하는 단계

를 포함하는 컴퓨터 구현 방법.

청구항 11

제10항에 있어서,

상기 스토어의 스토어 정보를 캡슐화하는 단계를 더 포함하며,

상기 스토어 정보는 서버 정보, 인증 정보 및 매핑 정보를 포함하는 컴퓨터 구현 방법.

청구항 12

제10항에 있어서,

스토리지 탐색기(storage searcher)를 통해 스토리지 뷰를 구축하는 단계를 더 포함하는 컴퓨터 구현 방법.

청구항 13

제10항에 있어서,

스토리지 탐색기를 통해 스토리지 도메인에 쿼리를 행하는 단계를 더 포함하는 컴퓨터 구현 방법.

청구항 14

제10항에 있어서,

클라이언트와 하나 이상의 스토어들 간의 연결을 캡슐화하는 클래스를 제공하는 단계를 더 포함하는 컴퓨터 구현 방법.

청구항 15

제10항에 있어서,

스토어들 중 적어도 하나를 노출시키는 것은 일반 데이터 액세스 컴포넌트를 통하는 컴퓨터 구현 방법.

청구항 16

데이터 플랫폼에 대한 애플리케이션 프로그래밍 인터페이스를 제공하기 위해 컴퓨팅 시스템에 이용하기 위한 컴퓨터 판독 가능 저장 매체로서 - 상기 애플리케이션 프로그래밍 인터페이스는 서로 다른 다양한 타입의 데이터 객체들에 대해 일반적인 방식(generic manner)으로 액세스를 제공함 - ,

상기 컴퓨터 판독 가능 저장 매체는 컴퓨터 실행 가능 명령어들을 저장하는 하나 이상의 저장 요소(storage element)를 포함하며,

상기 컴퓨터 실행 가능 명령어들은, 데이터 플랫폼에 대한 일반(generic) 애플리케이션 프로그래밍 인터페이스(API)를 포함하며,

상기 일반 API는,

상기 데이터 플랫폼의 스토어들, 세션들, 트랜잭션들 및 쿼리 서비스들의 각각을 노출시키는 일반 데이터 액세스 컴포넌트 - 상기 데이터 플랫폼은 데이터 스토어와 연관되어 있음 - ;

상기 데이터 플랫폼의 데이터 모델의 유형들 및 관계들을 노출시키는 표준형의 애플리케이션 독립적인 클래스의 데이터 클래스 컴포넌트;

상기 데이터 플랫폼의 도메인 고유 속성 및 거동을 노출시키는 애플리케이션 고유 클래스 및 프레임워크 고유 클래스의 도메인 데이터 클래스 컴포넌트 - 도메인은 애플리케이션과는 상이함 - ;

스키마의 테이블들에 대한 액세스를 제공하는 스키마 클래스 - 상기 스키마 클래스는 타겟 스키마에 기초하여 유형 미지정 스키마 클래스로부터 파생되는 강타입 스키마 클래스임 - ;

데이터 모델 스키마로부터 생성되며 상기 스키마 내에 정의된 테이블들에 대해 강타입 액세스를 제공하는 TableSet 클래스;

상기 API 내의 나머지 클래스들이 동작하는 스토어를 정의하는 StorageDomain 클래스;

세션에 대한 컨텍스트를 제공하며, 또한 현재 컨텍스트 내의 객체들에 대한 변경을 리프레쉬 또는 저장하는 메소드(method)들에 의해, 식별자 관리, 변경 추적, 및 동시성 충돌 처리에 대한 스코프(scope)를 정의하는 StorageContext 클래스;

데이터 스토어에 대해 합성 가능한 객체-기반 쿼리들을 구축하도록 구성되는 StorageSearcher 클래스; 및

쿼리로부터의 결과 세트에 대한 애플리케이션 뷰를 제공하며 필터링, 정렬, 스크롤, 그룹화, 섹션화, 섹션의 확장 및 축소를 지원하는 StorageView 클래스

를 포함하는 컴퓨터 판독 가능 저장 매체.

청구항 17

삭제

청구항 18

삭제

청구항 19

삭제

청구항 20

삭제

명 세 서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

[0022] 본 출원은 2005년 2월 28일자로 출원된 발명의 명칭이 "공통 데이터 플랫폼에 대한 스토리지 API(STORAGE API FOR A COMMON DATA PLATFORM)"인 미국 가특허출원 제60/657,522호를 우선권 주장한다. 본 출원은 2005년 2월 28일자로 출원된 발명의 명칭이 "개별적인 애플리케이션 프레임워크에 걸친 데이터 서비스를 위한 플랫폼(PLATFORM FOR DATA SERVICES ACROSS DISPARATE APPLICATION FRAMEWORKS)"인 미국 가특허출원 제60/657,556호, 2005년 6월 30일자로 출원된 발명의 명칭이 "개별적인 애플리케이션 프레임워크에 걸친 데이터 서비스를 위한 플랫폼(PLATFORM FOR DATA SERVICES ACROSS DISPARATE APPLICATION FRAMEWORKS)"인 미국 특허출원 제11/171,905호, 2005년 2월 28일자로 출원된 발명의 명칭이 "객체-관계형 데이터에 대한 데이터 모델(DATA MODEL FOR OBJECT-RELATIONAL DATA)"인 미국 가특허출원 제60/657,295호, 및 _____일자로 출원된 발명의 명칭이 "객체-관계 데이터에 대한 데이터 모델(DATA MODEL FOR OBJECT-RELATIONAL DATA)"인 미국 특허출원 제_____호(대리인 문서 번호 MSFTP974USA)에 관한 것이다. 상기한 출원들은 여기에 인용함으로써 그 전체 내용이 본 명세서에 포함된다.

[0023] 제품을 브라우징하고 주문을 발생하는 데 이용되는 LOB(Line-of-Business) 애플리케이션 프레임워크든 사람들 간의 만남을 스케쥴링하는 데 사용되는 PIM(Personal Information Management) 최종 사용자 애플리케이션이든 지 간에, 데이터는 거의 모든 애플리케이션에 있어서 중요한 자산이 되었다. 애플리케이션은 애플리케이션 데이터에 대해 데이터 액세스/처리 및 데이터 관리 동작 둘다를 수행한다. 일반적인 애플리케이션 동작은 데이터 컬렉션에 쿼리를 행하고, 결과 세트(result set)를 페치(fetch)하며, 데이터의 상태를 변경하는 어떤 애

플리케이션 로직을 실행하고, 또 마지막으로 데이터를 저장 매체에 영구 보존한다.

[0024] 전통적으로, 클라이언트/서버 애플리케이션은 쿼리(query) 및 영구 보존(persistence) 동작을, 데이터 계층에 배치되어 있는 데이터베이스 관리 시스템(DBMS)에 맡겨 왔다. 데이터 중심 로직(data-centric logic)이 있는 경우, 이는 데이터베이스 시스템에서 저장 프로시저(stored procedure)로서 코딩된다. 데이터베이스 시스템은 테이블 및 행의 관점에서 데이터를 처리하였고, 애플리케이션은 애플리케이션 계층에서, 프로그래밍 언어 객체(예를 들어, Class 및 Struct)의 관점에서 데이터를 처리하였다. 애플리케이션 및 데이터 계층에서의 데이터 처리 서비스(및 메카니즘)의 불일치가 클라이언트/서버 시스템에서는 허용되었다. 그렇지만, 웹 기술(및 서비스 지향 아키텍처(Service Oriented Architecture))의 등장으로 또 애플리케이션 서버의 더욱 광범위한 채택으로, 애플리케이션은 다중-계층(multi-tier)으로 되고 있으며, 보다 중요한 것은 데이터가 현재 모든 계층에 존재한다는 것이다.

[0025] 이러한 계층화된 애플리케이션 아키텍처에서, 데이터는 다수의 계층에서 처리된다. 게다가, 주소 지정 능력(addressability) 및 대용량 메모리에 있어서의 하드웨어 진보에 따라, 더 많은 데이터가 메모리에 상주(memory resident)하게 되고 있다. 애플리케이션은 또한 예를 들어 객체, 파일, 및 XML(eXtensible Markup Language) 데이터 등의 서로 다른 유형의 데이터를 처리하고 있다.

[0026] 이러한 하드웨어 및 소프트웨어 환경에서, 프로그래밍 환경과 잘 통합된, 풍부한 데이터 액세스 및 처리 서비스의 필요성이 증가하고 있다. 상기한 문제점을 해소하기 위해 도입된 한가지 종래의 구현이 데이터 플랫폼이다. 데이터 플랫폼은 애플리케이션 프로그래밍 환경과 잘 통합되어 있는, 애플리케이션이 데이터에 대한 액세스, 처리 및 관리를 행하기 위한 서비스(메카니즘)들의 컬렉션을 제공한다. 그렇지만, 이러한 종래의 아키텍처는 많은 점에서 부족하다. 이러한 데이터 플랫폼에 대한 어떤 주된 요건은 복잡한 객체 모델링, 풍부한 관계, 논리적 및 물리적 데이터 추상화의 분리, 풍부한 데이터 쿼리 모델(query rich data model) 개념, 적극적 통지(active notification), 중간-계층 기반구조(middle-tier infrastructure)와의 더 나은 통합을 포함한다. 따라서, 당해 기술 분야에서 개선된 데이터 플랫폼의 필요성이 상당히 있다.

발명이 이루고자 하는 기술적 과제

[0027] 이하에서는 개시된 발명의 어떤 측면들의 기본적인 이해를 제공하기 위해 간략화된 요약을 제공한다. 이 요약은 전반적인 개요는 아니며, 또 주요/중요한 구성요소를 확인하거나 본 발명의 범위를 한정하기 위한 것이 아니다. 그의 유일한 목적은 이후에 제공되는 보다 상세한 설명에 대한 전제로서 어떤 개념을 간략화된 형태로 제공하는 데 있다.

[0028] 본 발명의 한 측면에 있어서, 본 명세서에 개시되고 청구된 특징은 데이터 플랫폼에 대한 애플리케이션 프로그램 인터페이스(API)를 포함한다. 이 API는 데이터 플랫폼의 스토어, 세션, 트랜잭션 및 쿼리 서비스 중 적어도 하나를 노출시키는 일반 데이터 액세스 컴포넌트(generic data access component)를 포함하며, 이 데이터 플랫폼은 데이터 스토어와 연관되어 있다. API의 데이터 클래스 컴포넌트(data class component)는 데이터 플랫폼의 데이터 모델의 유형 및 관계를 노출시키는 표준형의 애플리케이션 독립적인 클래스(canonical, application-independent class)를 제공한다. 이 API는 데이터 플랫폼의 도메인 관련 속성 및 거동을 노출시키는 애플리케이션 관련 클래스 및 프레임워크 관련 클래스의 도메인 데이터 클래스 컴포넌트(domain data class component)를 포함한다. 이 데이터 플랫폼은 복수의 개별적인 애플리케이션 프레임워크에 의해 액세스 가능한 데이터 서비스를 제공하기 위해 데이터 스토어와 인터페이스하는 공통 데이터 플랫폼(common data platform)일 수 있으며, 이 데이터 서비스는 서로 다른 프레임워크의 대응하는 애플리케이션이 데이터 스토어에 액세스할 수 있게 해준다.

[0029] 다른 측면에서, API는 5개의 코어 클래스를 포함한다. TableSet 클래스는 데이터 모델 스키마로부터 생성될 수 있고 이 스키마 내에 정의된 테이블에 대한 강타입(strongly typed) 액세스를 제공한다. StorageDomain 클래스는 나머지 클래스들이 동작하고 있는 스토어를 정의한다. StorageContext 클래스는 세션에 대한 컨텍스트를 제공한다. StorageContext 클래스는 현재 컨텍스트 내의 객체들에 대한 변경을 리프레쉬 또는 저장하는 메소드로 식별자 관리(identity management), 변경 추적(change tracking) 및 동시성 충돌 처리(concurrency conflict handling)에 대한 스코프(scope)를 정의한다. StorageSearcher 클래스는 데이터 스토어에 대해 합성가능 객체 기반 쿼리(composable object-based query)를 작성하는 데 사용된다. StorageView 클래스는 결과 세트에 대한 풍부한 애플리케이션 뷰를 제공한다. StorageView 클래스는 필터링, 정렬, 스크롤링, 그룹화, 섹션화, 섹션의 확장/축소, 기타 등등의 동작을 지원한다.

[0030] 상기한 목적 및 관련 목적을 달성하기 위해, 개시된 특징의 어떤 예시적인 측면이 이하의 설명 및 첨부 도면과 관련하여 본 명세서에 기술되어 있다. 그렇지만, 이들 측면은 본 명세서에 개시된 원리들이 이용될 수 있는 여러가지 방법 중 단지 몇개를 나타낸 것에 불과하며, 이러한 측면들 및 그의 등가물 모두를 포함하는 것으로 보아야 한다. 다른 이점 및 새로운 특징은 도면과 관련하여 살펴볼 때 이하의 상세한 설명으로부터 명백하게 될 것이다.

발명의 구성 및 작용

[0031] 이제부터, 그 전체에 걸쳐 유사한 참조 번호가 유사한 구성요소를 언급하는 데 사용되는 도면을 참조하여, 본 발명에 대해 기술한다. 이하의 설명에서, 설명의 목적상, 본 발명에 대한 철저한 이해를 제공하기 위해 여러가지 구체적인 상세가 기술된다. 그렇지만, 본 발명이 이들 구체적인 상세 없이도 실시될 수 있음은 명백할 수 있다. 다른 경우에, 본 발명의 설명을 용이하게 해주기 위해 공지의 구조 및 장치가 블록도 형태로 도시되어 있다.

[0032] 본 출원에서 사용되는 바와 같이, 용어 "컴포넌트" 및 "시스템"은 컴퓨터 관련 개체, 즉 하드웨어, 하드웨어와 소프트웨어의 조합, 소프트웨어, 또는 실행 중인 소프트웨어를 언급하기 위한 것이다. 예를 들어, 컴포넌트는 프로세서 상에서 실행 중인 프로세스, 프로세서, 하드 디스크 드라이브, (광학 및/또는 자기 저장 매체의) 다중 저장 장치, 객체, 실행 파일, 실행 쓰레드, 프로그램, 및/또는 컴퓨터일 수 있지만, 이에 한정되는 것은 아니다. 예로서, 서버 상에서 실행 중인 애플리케이션 및 서버는 컴포넌트일 수 있다. 프로세스 및/또는 실행 쓰레드 내에 하나 이상의 컴포넌트가 존재할 수 있고, 컴포넌트는 하나의 컴퓨터 상에 로컬화 및/또는 2개 이상의 컴퓨터 간에 분산되어 있을 수 있다.

[0033] 정보를 사용자에게 디스플레이하는 어떤 방법들이 어떤 도면과 관련하여 스크린샷으로서 도시되고 설명되어 있지만, 당업자라면 여러가지 다른 대안이 이용될 수 있음을 잘 알 것이다. 용어 "스크린", "웹 페이지" 및 "페이지"는 본 명세서에서 일반적으로 상호교환가능하게 사용되고 있다. 페이지 또는 스크린은 디스플레이 설명으로서, 그래픽 사용자 인터페이스로서, 또는 스크린(예를 들어, 퍼스널 컴퓨터, PDA, 모바일 전화, 또는 다른 적합한 장치 등) 상에 정보를 표시하는 다른 방법에 의해, 저장 및/또는 전송되며, 여기서 페이지 상에 디스플레이될 레이아웃 및 정보 또는 콘텐츠는 메모리, 데이터베이스 또는 다른 저장 설비에 저장되어 있다.

[0034] 신규의 공통 데이터 플랫폼(common data platform, CDP)은 객체 및 이들이 어떻게 관련되어 있는지를 기술하는 공통 데이터 모델(common data model, CDM), 및 이들 객체의 인-메모리(in-memory) 표현을 처리하는 영속적 스토어(persistent store) 및 서비스로 이루어져 있다. CDP는 영속적 데이터를 애플리케이션 데이터로서 처리하는 혁신적인 플랫폼을 제공한다. CDP는 플랫폼의 일부로서 정의된 하부의 데이터 모델 및 서비스에 커스터마이즈되어 있는 신규의 애플리케이션 프로그램 인터페이스(API)를 포함한다. CDP의 기능은 일련의 클래스를 통해 노출된다. 공개 멤버(public member)(예를 들어, 메소드 및 속성)를 포함하는 이들 클래스의 정의는 CDP 내의 객체들을 처리하는 API를 포함한다.

[0035] 우선 도면들을 참조하면, 도 1은 본 발명의 한 측면에 따른 데이터 플랫폼(102)(예를 들어, CDP)의 스토리지 API(100)을 나타내고 있다. API(100)는 데이터 플랫폼(예를 들어, CDP)을 사용하는 애플리케이션에 대한 프로그래밍 인터페이스를 클래스, 인터페이스 및 정적 헬퍼 함수(static helper function)의 형태로 제공한다. 데이터베이스 프로그래밍 언어 통합(예를 들어, C# 시퀀스 연산자)도 역시 이 API 계층의 일부이다. 이를 지원하기 위해, API(100)는 개체(Entity), 관계(Relationship), 확장(Extension), 기타 등등의 CDM 개념을 노출시키는 표준형의 애플리케이션 독립적인 클래스의 세트인 CDM 데이터 클래스 컴포넌트(104)를 포함한다. 일반 데이터 액세스 컴포넌트(106)는 스토어, 세션, 트랜잭션(예를 들어, StorageContext), 쿼리 서비스(예를 들어, StorageSearcher) 및 CRUD 서비스(예를 들어, SaveChanges)를 노출시키기 위해 API(100)의 일부로서 제공된다. CRUD(Create, Retrieve, Update, Delete) 서비스는 데이터에 적용되는 기본적인 프로세스이다. API(100)은 또한 CDM에 부합하지만 도메인 관련 속성 및 거동을 노출시키는 Contact(연락처), Message(메시지), PurchaseOrders(구매 주문) 등의 애플리케이션/프레임워크 관련 클래스인 도메인 데이터 클래스 컴포넌트(108)를 포함한다.

[0036] 도 2는 개시된 측면에 따른 스토리지 API를 제공하는 방법을 나타낸 것이다. 설명의 간략함을 위해, 본 명세서에 예를 들어 플로우차트 또는 흐름도의 형태로 도시되어 있는 하나 이상의 방법이 일련의 단계로서 도시되고 기술되어 있지만, 본 발명이 단계들의 순서에 의해 제한되지 않는데 그 이유는 본 발명에 따라 본 명세서에 도시되고 기술되어 있는 것과 다른 순서로 및/또는 다른 단계들과 동시에 행해질 수 있기 때문이라는 것을 잘 알 것이다. 예를 들어, 당업자라면 방법이 다른 대안으로서 상태도에서와 같이 일련의 상호 관련된 상태

또는 이벤트로서 표현될 수 있음을 잘 알 것이다. 게다가, 예시된 모든 단계들이 본 발명에 따라 방법을 구현해야만 하는 것은 아닐 수 있다. 단계(200)에서, 스토리지 API가 수신된다. 단계(202)에서, API는 데이터를 쿼리하기 위한 클래스를 정의한다. 단계(204)에서, API는 데이터를 검색하기 위한 클래스를 정의한다. 단계(206)에서, API는 데이터를 순회하기 위한 클래스를 정의한다. 단계(208)에서, API는 데이터를 수정하기 위한 클래스를 정의한다. 단계(210)에서, API는 데이터를 영구 보존시키기 위한 클래스를 정의한다.

[0037] 도 3은 일반 데이터 액세스 컴포넌트(106)를 보다 상세히 나타낸 도면이다. API(300)는 기능을, 사용하기 쉽고 확장가능하며 강력하고 또 합성가능한 클래스 및 메소드로 인수화(factoring)하는 것을 정의한다. 스토리지 API(300)는 공통 데이터 모델에 부합하는 데이터에 대한 변경을 쿼리, 검색, 순회, 수정 및 영구 보존시키기 위한 클래스를 정의한다. 스토리지 API(300)는 이러한 쿼리, 순회 및 영구 보존 기능을, 스토리지 API(300)에 의해 쿼리, 순회 및 영구 보존된 규범적 데이터 클래스(prescriptive data class)에 정의된 기능과 분리시킨다.

[0038] 스토리지 API(300)는 이하의 코어 클래스로 이루어져 있으며, 도 3은 StorageContext, TableSet, StorageSearcher 및 StorageView 간의 관계를 나타낸 것이다. 부가의 클래스가 이들 코어 클래스를 지원하기 위해 정의될 수 있다.

[0039] TableSet - TableSet 클래스는 데이터 모델 스키마로부터 생성될 수 있으며 스키마 내에 정의된 테이블들에 대한 강타입 액세스(strongly typed access)를 제공한다. TableSet 인스턴스는 하나 이상의 StorageContext 인스턴스를 래핑(wrap)하고, 객체를 쿼리, 순회 및 업데이트하기 위해 하부의 StorageContext 클래스 및 연관된 StorageDomain 클래스를 사용한다. 스키마 관련 또는 프레임워크 관련 기능을 위해, 생성된 TableSet 클래스에 부가의 메소드가 추가될 수 있다.

[0040] StorageDomain - 나머지 클래스들이 동작하고 있는 스토어를 정의하는 클래스. 서로 다른 유형의 클래스는 그 자신의 고유 StorageDomain 클래스를 구현한다. StorageDomain은 직접 또는 TableSet와 관련하여 사용될 수 있다.

[0041] StorageContext - 세션에 대한 컨텍스를 제공하는 클래스. StorageContext 클래스는 현재의 컨텍스트 내의 객체들에 대한 변경을 리프레쉬 또는 저장하기 위한 메소드로 식별자 관리, 변경 추적 및 동시성 충돌 처리에 대한 스코프(scope)를 정의한다. StorageContext 클래스는 (예를 들어, 데이터를 리프레쉬하거나 변경을 영구 보존시킬 때에) 스토어와 통신하기 위해 StorageDomain 클래스를 사용한다. StorageContext는 직접 또는 TableSet와 관련하여 사용될 수 있다.

[0042] StorageSearcher - StorageSearcher 클래스는 데이터 스토어에 대한 합성가능 객체 기반 쿼리를 작성하는 데 사용된다. StorageSearcher 클래스는 일반적으로 StorageContext 내에서 StorageDomain에 의해 실행되는 StorageExpression 클래스를 생성한다. StorageSearcher는 전방 전용 스트림(forward-only streamed) 방식으로 결과들을 열거하는 것 또는 풍부한 스크롤가능 StorageView의 구축을 지원한다.

[0043] StorageView - StorageView 클래스는 결과 세트에 대한 풍부한 애플리케이션 뷰를 제공한다. StorageView는 필터링, 정렬, 스크롤, 그룹화, 섹션화, 섹션의 확장/축소, 기타 등등의 동작을 지원한다.

[0044] 이제 도 4를 참조하면, 데이터 모델에 대한 스토리지 API를 제공하는 방법이 도시되어 있다. 단계(400)에서, 데이터 플랫폼(예를 들어, CDP)이 데이터 스토어에 대해 활용하기 위해 수신된다. 단계(402)에서, 예를 들어, 개체, 관계, 확장 등의 CDM 개념을 표현하는 기본 클래스를 포함하는 API가 제공된다. 데이터 플랫폼의 하부의 기능이 본 발명의 API에 정의되어 있는 공통 CDM 데이터 클래스를 통해 상부의 애플리케이션 및 애플리케이션 플랫폼에 노출될 수 있다. 단계(404)에서, 다른 API 클래스들이 동작하고 있는 데이터 스토어를 정의하는 클래스가 제공된다. 단계(406)에서, 데이터 스토어에 대한 객체 기반 쿼리를 작성하는 데 사용되는 클래스가 제공된다. 단계(408)에서, 세션 컨텍스트를 정의하고 또 식별자 관리, 변경 추적, 충돌 처리, 기타 등등을 포함하는 클래스가 제공된다. 단계(410)에서, 스키마로부터 생성되고 스키마의 테이블에 대한 유형 지정된 액세스(typed access)를 제공하는 클래스가 제공된다. 단계(412)에서, 결과 세트(들)를 보는 것을 용이하게 해주는 클래스가 제공된다. 단계(414)에서, CDM 스키마의 인스턴스에 의해 기술된 특정의 개체 및 관계를 표현하기 위해 일련의 도메인 관련 클래스가 정의된다.

[0045] 이하의 섹션들에서 공통 데이터 모델에 대한 API를 구성하는 클래스 및 멤버 정의에 대해 상세히 기술한다.

[0046] StorageDomain 클래스. StorageDomain 클래스는 서버, 인증, 매핑, 기타 등등의 스토어 정보를 캡슐화하는 데 사용된다. 기본 스토리지 도메인 클래스(base storage domain class)는 스토어 관련 정보를 제공하기 위

해 각각의 유형의 스토어로부터 파생된다. 기본 StorageDomain 유형은 다음과 같이 정의될 수 있다.

```
public abstract class StorageDomain : IDisposable
{}
```

[0047]

[0048] WinFSDomain 클래스. WinFS 스토어에 대한 StorageDomain의 예는 다음과 같을 수 있다.

```
public class WinFSDomain : StorageDomain
{
   public WinFSDomain();
   public WinFSDomain(string share);
}
```

[0049]

[0050] WinFSDomain 생성자(constructor)는 예를 들어 UNC(universal naming convention) 공유 이름(share name)를 통해 스토어 및 스토어 내에서의 스코프를 지정하기 위한 정보를 취할 수 있다. 다른 대안으로서, 디폴트 생성자(default constructor)는 예를 들어 디폴트 스토어의 루트에 대해 디폴트 스토어 정보를 사용할 수 있다. UNC는 UNIX 커뮤니티로부터 창안된, 네트워크 내의 서버, 프린터 및 다른 자원을 식별하기 위한 표준이다. UNC 경로는 컴퓨터의 이름 앞에 이중 슬래쉬 또는 백슬래쉬를 사용한다.

[0051] SqlStorageDomain 클래스. 관계형 스토어(예를 들어, SQL 데이터베이스)에 대한 StorageDomain의 예는 다음과 같을 수 있다.

```
public class SqlStorageDomain : StorageDomain
{
   public SqlStorageDomain();
   public SqlStorageDomain(String connectionString);
   public SqlStorageDomain(SqlConnection connection, String
      mappingFile);
   public SqlStorageDomain(SqlConnection connection,
      IRelationalMapping mapping);
}
```

[0052]

[0053] SqlStorageDomain 생성자는 연결 정보를, 예를 들어 연결 및 매핑 정보를 포함하는 연결 문자열(connection string) 또는 이러한 정보를 포함하는 명명된 구성(named configuration)의 형태로 취할 수 있다. 다른 대안으로서, 생성자는 매핑 정보와 함께 연결 객체를, 표준 매핑 인터페이스를 구현하는 객체 또는 매핑 파일의 형태로 취할 수 있다. 다른 대안으로서, 디폴트 생성자는 예를 들어 구성 파일로부터 디폴트 연결 또는 매핑 정보를 사용할 수 있다.

[0054] 도 5는 테이블 세트 유형을 노출시키는 방법을 나타낸 것이다. 단계(500)에서, 테이블 유형에 대한 기본 클래스가 수신된다. TableSet 클래스는 테이블 세트 유형에 대한 기본 클래스로서 사용된다. 이 유형의 인스턴스도 역시 원하는 경우 애플리케이션에 의해 직접 생성되고 사용될 수 있다. 기본 TableSet 유형은 다음과 같은 멤버를 갖는다.

```
public class TableSet : IDisposable
{

   public TableSet( StorageContext context, string tableSetName );
   public TableSet( StorageDomain domain, string tableSetName );
   public TableSet( StateManager manager, string tableSetName);

   public void Dispose();

   public StorageContext Context { get; }

   public string Name { get; }

   public Table<T> GetTable<T>(string propertyName);
   public object GetTableSetReference(string propertyName);

   public void SaveChanges();

}
```

[0055]

[0056] TableSet은 일반적으로 스키마(Schema) 내의 테이블 세트의 이름으로 생성된다. 다른 대안으로서, 스키마 내의 테이블 세트는 대체 메카니즘을 통해, 예를 들어 디폴트 네이밍(default naming), 구성 파일, 기타 등등을 통해 결정될 수 있다. StorageContext는 TableSet를 기존의 StorageContext와 연관시키기 위해 TableSet에 제공될 수 있다. 다른 대안으로서, StorageDomain은 TableSet를 StorageDomain과 연관시키기 위해 TableSet에 제공될 수 있다. 또다른 대안으로서, TableSet는 공통 상태 관리자에 제공될 수 있다.

[0057] 단계(502)에서, 테이블 세트와 연관된 데이터 객체를 저장하기 위해 SaveChanges 메소드가 제공될 수 있다. 이 메소드의 비동기적 버전도 역시 제공된다. 단계(504)에서, 제공된 이름에 기초하여 스키마 내의 테이블을 나타내는 객체를 생성 및 반환하기 위한 GetTable 메소드가 제공될 수 있다(예를 들어, Table<T>). 단계(506)에서, TableSetReference를 반환하기 위해 GetTableSetReference 메소드가 제공될 수 있다.

[0058] 도 6은 본 발명의 API에 WinFS 기능을 제공하는 방법을 나타낸 것이다. 단계(600)에서, WinFS 기능에 대한 클래스가 사용된다. WinFSData 클래스는 WinFS 관련 기능을 제공하기 위해 TableSet 클래스로부터 파생될 수 있다. WinFSData 클래스는 다음의 멤버를 갖는다.

```
public partial WinFSData : TableSet {

  public WinFSData( StorageContext context );
  public WinFSData( StorageContext context, string tableSetName );
  public WinFSData();
  public WinFSData( string share );

  public Item GetRootItem() {}
  public Item GetItemByPath( string path ) {}

  public Table<Item> Items { get {} }
  public Table<Link> Links { get {} }
  public Table<ItemExtension> ItemExtensions { get {} }
  public Table<ItemFragment> ItemFragments { get {} }
```

[0059]

[0060] // 복사(Copy) 메소드

```
  public Ref<Item> CopyItem(string sourceItemName, string
     destinationItemName );
  public Ref<Item> CopyItem(string sourceItemName, string
     destinationItemName,
                            CopyItemOptions options );
  public Ref<Item> CopyItem(string sourceItemName, string
     destinationItemName,
                            StorageContext destinationContext,
     CopyItemOptions options );
  public Ref<Item> CopyItem( Ref<Item> sourceItemRef,
                     Ref<Item> destinationContainerRef );
  public Ref<Item> CopyItem( Ref<Item> sourceItemRef,
                            Ref<Item> destinationContainerRef,
     CopyItemOptions options);
  public Ref<Item> CopyItem( Ref<Item> sourceItemRef,
                            Ref<Item> destinationContainerRef, string
     newNamespaceName,
                            CopyItemOptions options);
  public Ref<Item> CopyItem( Item sourceItem, Item
     destinationContainer );
  public Ref<Item> CopyItem( Item sourceItem, Item
     destinationContainer,
                             CopyItemOptions options);
  public Ref<Item> CopyItem( Item sourceItem, Item
     destinationContainer,
                            string newNamespaceName, CopyItemOptions
     options);
```

[0061]

[0062] // 이동(Move) 메소드

```
  public void MoveItem( string sourceItemName, string
     destinationItemName );
  public void MoveItem( string sourceItemName, string
     destinationItemName,
                        MoveItemOptions options );
  public void MoveItem( Ref<Item> sourceItemRef, Ref<Item>
     destinationContainerRef );
  public void MoveItem( Ref<Item> sourceItemRef, Ref<Item>
     destinationContainerRef,
                            string newNamespaceName, MoveItemOptions
     options );
  public void MoveItem( Item sourceItem, Item destinationContainer
     );
  public void MoveItem( Item sourceItem, Item
     destinationContainer,
                            string newNamespaceName, MoveItemOptions
     options );
```

[0063]

[0064] // 삭제(Delete) 메소드

```
public void DeleteItem ( string itemName );
public void DeleteItem ( string itemName, ItemDeleteOptions
   options );
public void DeleteItem ( Ref<Item> itemRef );
public void DeleteItem ( Ref<Item> itemRef, ItemDeleteOptions
   options );
public void DeleteItem ( Item item );
public void DeleteItem ( Item item, ItemDeleteOptions options );
```

[0065]

[0066] // 내보내기(Export) 메소드

```
public void ExportItem( string itemName, Stream stream );
public void ExportItem( string itemName, string fileName );
public void ExportItem( string itemName, string fileName,
          ExportItemOptions options );
public void ExportItem( Ref<Item> itemRef, Stream stream );
public void ExportItem( Ref<Item> itemRef, string fileName );
public void ExportItem( Ref<Item> itemRef, string fileName,
              ExportItemOptions options );
public void ExportItem( Item item, Stream stream );
public void ExportItem( Item item, string fileName );
public void ExportItem( Item item, string fileName,
   ExportItemOptions options );
```

[0067]

[0068] // 가져오기(Import) 메소드

```
public void ImportItem( Stream stream, string itemName );
public void ImportItem( string fileName, string itemName );
public void ImportItem( string fileName, string itemName,
                        ImportItemOptions options );
public void ImportItem( Stream stream, Ref<Item>
   containerItemRef,
                        string namespaceName );
public void ImportItem( Stream stream, Ref<Item>
   containerItemRef,
                        string namespaceName );
public void ImportItem( string fileName, Ref<Item>
   containerItemRef,
                        string namespaceName );
public void ImportItem( string fileName, Ref<Item>
   containerItemRef,
                        string namespaceName,
   ImportItemOptions options );
public void ImportItem( Stream stream, Item containerItem,
   string uniuqeName );
```

[0069]

```
  public void ImportItem( string fileName, Item containerItem,
     string namespaceName );
  public void ImportItem( string fileName, Item item, string
     namespaceName,
                          ImportItemOptions options );
}
```

[0070]

[0071] WinFSData 생성자는 기존의 StorageContext로 생성될 수 있거나 지정된 정보(UNC 공유 등) 또는 디폴트 정보(예를 들어, 디폴트 스토어의 루트)를 사용하여 StorageContext를 생성할 수 있다. 게다가, WinFSData 클래스를 특정의 명명된 tableset 인스턴스와 연관시키기 위해 tableset Name이 지정될 수 있다.

[0072] 단계(602)에서, 도메인의 루트를 반환하기 위해 GetRootItem 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(604)에서, 그의 경로가 주어지면 아이템을 반환하기 위해 GetItemByPath 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다.

[0073] 단계(606)에서, Items, ItemsExtensions 및 ItemFragments 테이블을 나타내는 객체를 반환하기 위해 Items, ItemExtensions 및 ItemFragments 속성이 제공될 수 있다. 단계(608)에서, Links 테이블을 나타내는 객체를 반환하기 위해 Links 속성이 제공될 수 있다. 단계(610)에서, 아이템을 복사, 이동 및 삭제하기 위한 메소드가 제공된다. 지정된 아이템을 스토어 내의 다른 장소로 복사하기 위해 CopyItem 메소드가 제공될 수 있다. 지정된 아이템을 스토어 내에서 이동시키기 위해 MoveItem 메소드가 제공될 수 있다. DeleteItem 메소드는 지정된 아이템의 스토어로부터의 삭제를 제공한다. 단계(612)에서, 아이템을 가져오기(import) 및 내보내기(export) 하기 위한 메소드가 제공된다. 지정된 아이템을 스토어로부터 내보내기 하기 위해 ExportItem 메소

드가 제공될 수 있다. 지정된 아이템을 스토어 내로 가져오기 하기 위해 ImportItem 메소드가 제공될 수 있다. CopyItem 메소드, MoveItem 메소드, DeleteItem 메소드, ExportItem 메소드 및 ImportItem 메소드의 비동기 버전도 역시 제공될 수 있다.

[0074] 도 7은 스토어 내의 클래스를 나타내는 방법을 나타낸 것이다. 단계(700)에서, 스토어 내에서의 익스텐트(extent)를 나타내는 클래스가 정의된다. Table<T> 클래스는 스토어 내에서의 익스텐트(extent)를 나타내는 데 사용된다. Table<T> 클래스는 객체를 익스텐트에 부가 또는 그로부터 제거하는 메소드는 물론 익스텐트의 콘텐츠에 대한 StorageSearcher를 작성하는 메소드를 가질 수 있다.

```
public class Table<T> {
```

[0075]

```
  public Table(StorageContext context, string TableName);
  public Table(StorageDomain domain, string TableName);

public StorageContext Context { get; internal set;}
public StorageDomain Domain { get; internal set;}
```

[0076]

```
public StorageSearcher<T> Searcher { get; }
```

[0077]

```
// ICollection을 지원

  bool ICollection<T>.Add(T obj);
  void ICollection<T>.Remove(T obj);
  void ICollection<T>.Clear();
  bool ICollection<T>.Contains(T t);
  public virtual int Count { get; }
  void ICollection<T>.CopyTo(T[] array, int arrayIndex);
  bool ICollection<T>.IsReadOnly { get { }}
```

[0078]

```
}
```

[0079] Table<T> 클래스는 스키마 내의 대응하는 테이블의 이름과 함께, StorageContext 또는 StorageDomain을 지정하는 정보로 생성될 수 있다. 단계(702)에서, Table<T>와 연관된 StorageContext를 반환하기 위해 Context 속성이 제공될 수 있다. 단계(704)에서, Table<T> 클래스와 연관된 StorageDomain을 반환하기 위해 Domain 속성이 제공될 수 있다. 단계(706)에서, 스토어 내의 대응하는 테이블에 대해 StorageSearcher를 반환하기 위해 Searcher 속성이 노출될 수 있다. 단계(708)에서, 객체를 부가(add), 제거(remove) 및 클리어(clear)하기 위한 메소드가 제공된다. 테이블에 객체를 부가하기 위해 Add 메소드가 노출될 수 있다. 테이블로부터 제거될 객체를 지정하기 위해 Remove 메소드가 노출될 수 있다. 테이블을 클리어하기 위해 Clear 메소드가 노출될 수 있다. 단계(710)에서, 테이블이 지정된 객체를 포함하는지 여부를 반환하기 위해 Contains 메소드가 노출될 수 있다. 단계(712)에서, 테이블 내의 객체의 총수를 지정하기 위해 Count 메소드가 노출될 수 있다. 단계(714)에서, 객체들을 테이블로 복사하는 메소드가 제공된다. 단계(716)에서, 테이블이 읽기 전용인지를 노출시키는 속성이 제공된다. 지정된 객체들을 테이블로 복사하기 위해 CopyTo 메소드가 노출될 수 있다. 테이블에 부가되거나 테이블로부터 제거될 수 있는지 여부를 반환하기 위해 IsReadOnly 속성이 노출될 수 있다.

[0080] 도 8은 클라이언트와 하나 이상의 스토어 간의 연결을 캡슐화하는 방법을 나타낸 것이다. 게다가, 클래스는 세션 컨텍스트(session context), 식별자 관리(identity management)를 위한 스코프(scope), 변경 추적(change tracking) 및 동시성 충돌 처리(concurrency conflict handling)를 정의한다. StorageContext 클래스는 클라이언트와 하나 이상의 스토어 간의 연결을 캡슐화하고, CRUD(Creat, Read, Update 및 Delete) 동작을 위한 게이트웨이이다.

```
public class StorageContext : IDisposable {
```

[0081]

```
  public StorageContext();
```

```
    public StorageContext(StorageDomain domain);

    public object GetObjectByKey(StorageKey key);
    public StorageKey GetObjectKey(object o);
    public void SaveChanges();

    public void Refresh(RefreshMode options, IEnumerable<object>
       objects);
    public void Refresh(RefreshMode options, params object[] objects);

    public void Dispose();

    public StorageDomain Domain { get; }
    public void Add(object o);
    public void MarkForDeletion(object o);
}
```

[0082]

[0083] 스토어 정보를 제공하는 StorageDomain이 주어지면 StorageContext가 생성된다. 다른 대안으로서, StorageContext는 StorageDomain이 없이 생성될 수 있고 또 구성 파일 등의 디폴트 소스로부터 스토어 정보를 획득할 수 있다.

[0084] 단계(802)에서, 키(key)를 통해 객체를 반환하는 메소드가 제공된다. 특정의 키와 연관된 StorageContext 내의 객체를 반환하기 위해 GetObjectByKey 메소드가 제공될 수 있다. 다른 대안으로서, 이 메소드는 개별적인 StateManagement 객체로 분해될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(804)에서, StorageContext 내의 특정의 객체와 연관된 키를 반환하기 위해 GetObjectKey 메소드가 제공될 수 있다. 다른 대안으로서, 이 메소드는 개별적인 StateManagement 객체로 분해될 수 있다. 단계(806)에서, StorageContext 내에서의 객체에 대한 부가, 삭제 또는 수정을 저장하기 위해 SaveChanges 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다.

[0085] 단계(808)에서, 현재의 스토어 값을 갖는 StorageContext 내의 객체들을 리프레쉬하기 위해 Refresh 메소드가 제공될 수 있다. 리프레쉬할 명시적인 객체 세트가 예를 들어 열거자(enumerator)를 통해 또는 파라미터로서 지정될 수 있다. 변경 충돌(change conflict)이 어떻게 처리되는지를 제어하기 위해 부가의 옵션이 지정될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(810)에서, 새로운 객체를 StorageContext와 연관시키기 위해 Add 메소드가 제공될 수 있다. 다른 대안으로서, 이 메소드는 개별적인 StateManagement 객체로 분해될 수 있다. 단계(812)에서, SaveChanges가 호출될 때 삭제될 StorageContext 내의 객체에 표시를 하기 위해 MarkForDeletion 메소드가 제공될 수 있다. 다른 대안으로서, 이 메소드는 개별적인 StateManagement 객체로 분해될 수 있다. 단계(814)에서, StorageContext와 연관된 StorageDomain을 반환하기 위해 StorageDomain 속성이 제공될 수 있다.

[0086] 도 9는 스토어에 대한 쿼리를 작성하는 방법을 나타낸 것이다. 단계(900)에서, 스토어에 대한 쿼리를 작성하기 위한 기본 클래스가 정의된다. 스토어에 대한 합성가능 객체 기반 쿼리를 작성하는 데 StorageSearcher 클래스가 사용된다. StorageSearcher는 일반적으로 StorageContext 내에서 StorageDomain에 의해 실행되는 StorageExpression을 생성한다. StorageSearcher는 전방 전용 스트림 방식으로 결과들을 열거하는 것 또는 풍부한 스크롤가능 StorageView의 생성을 지원한다.

```
public class StorageSearcher<T> : IStorageSearcher, IEnumerable<T>
  where T : class
{

  public StorageSearcher(string expression);
  public StorageSearcher(string expression, object[] parameters);
  public StorageSearcher(string expression, object[] parameters,
     StorageContext context);
  public StorageSearcher(StorageExpression expression);
  public StorageSearcher(StorageExpression expression,
     StorageContext context);
  public StorageSearcher(string expression, object[] parameters,
     StorageDomain store);
  public StorageSearcher(StorageExpression expression,
     StorageDomain store);
```

[0087]

```
public StorageContext Context { get; }
public StorageDomain Domain { get; }

public StorageExpression Expression { get; }
Type IStorageSearcher.ResultType { get; }

public StorageSearcher<T> BindContext(StorageContext context);
IStorageSearcher IStorageSearcher.BindContext(StorageContext
   context);

public StorageSearcher<T> BindParameters(IDictionary<string,
   object> parameters);
IStorageSearcher IStorageSearcher.BindParameters(IDictionary
   parameters);
```

[0088]

```
public StorageSearcher<T> Filter(string expression, params
   object[] parameters);
public StorageSearcher<U> FilterByType<U>() where U : T;
public StorageSearcher<U> TreatAsType<U>();
public StorageSearcher<T> Sort(string expression, params
   object[] parameters);
public StorageSearcher<StorageRecord> Project(string expression,
   params object[] parameters);
public StorageSearcher<StorageRecord> Group(string expression,
   params object[] parameters);
public StorageSearcher<T> Union(StorageSearcher<T> searcher);
```

[0089]

```
public StorageSearcher<U> Query<U>(string expression, params
   object[] parameters);
public StorageSearcher<U> Query<U>(StorageExpression
   expression);
IStorageSearcher IStorageSearcher.Query(Type resultType,
   StorageExpression expression);
IStorageSearcher IStorageSearcher.Query(Type resultType, string
   expression,
    params object[] parameters);

public IEnumerator<T> GetEnumerator();
IEnumerator IEnumerable.GetEnumerator();
```

[0090]

```
 public T GetFirst();
 object IStorageSearcher.GetFirst();
 public int GetCount();
 public List<T> GetList();

 public StorageView<StorageViewRecord> CreateView();
 public StorageView<StorageViewRecord>
    CreateView(StorageViewDefinition definition);
 public StorageView<StorageViewRecord> CreateView(
    StorageViewDefinition definition, StorageViewOptions options);
 public StorageView<T> CreateView<T>(StorageViewDefinition
    definition, StorageViewOptions options) where T :
    StorageViewRecord {}
}
```

[0091]

[0092] StorageSearcher는 StorageSearcher가 바인딩되어 있는 컨텍스트 또는 스토어를 지정하기 위해 StorageContext 또는 StorageDomain으로 생성될 수 있다. 게다가, StorageSearcher를 문자열 또는 StorageExpression 객체 트리로서 초기화하기 위해 쿼리 표현식이 지정될 수 있다.

[0093] 단계(902)에서, 임의의 쿼리 표현식을 캡슐화하는 새로운 StorageSearcher를 생성하기 위해 Query 메소드가 제공될 수 있다. 단계(904)에서, 필터 메소드가 제공된다. 입력 탐색기에 의해 생성되어지는 쿼리 결과에 대한 필터를 캡슐화하는 새로운 StorageSearcher를 생성하기 위해 Filter 메소드가 제공될 수 있다. 입력 탐색기에 의해 생성되어지는 쿼리 결과에 걸쳐 필터를 캡슐화하는 새로운 StorageSearcher를 생성하기 위해 FilterByType 메소드가 제공될 수 있다. 입력 탐색기에 의해 생성되어지는 쿼리 결과를 다른 유형으로서 취급하는 새로운 StorageSearcher를 생성하기 위해 FilterByType 메소드가 제공될 수 있다.

[0094] 단계(906)에서, Project, Group 및 Union 메소드가 제공된다. 입력 탐색기에 의해 생성되어지는 쿼리 결과의 정렬(sort)을 캡슐화하는 새로운 StorageSearcher를 생성하기 위해 Sort 메소드가 제공될 수 있다. 입력 탐색기에 의해 생성되어지는 쿼리 결과의 투영(projection)을 캡슐화하는 새로운 StorageSearcher를 생성하기 위해 Project 메소드가 제공될 수 있다. 입력 탐색기에 의해 생성되어지는 쿼리 결과의 그룹(grouping)을 캡슐화하는 새로운 StorageSearcher를 생성하기 위해 Group 메소드가 제공될 수 있다. 2개의 입력 탐색기에 의해 생성되어지는 쿼리 결과의 합집합(union)을 캡슐화하는 새로운 StorageSearcher를 생성하기 위해 Union 메

소드가 제공될 수 있다. 상기한 것은 예에 불과하며, 제한적인 것으로 해석되어서는 안된다. 부가의 쿼리 동작을 표현하기 위해 StorageSearcher 상에 부가의 메소드가 제공될 수 있다. 환언하면, 쿼리 동작은 새로운 StorageSearcher를 반환하는 StorageSearcher 클래스 상의 메소드로서 노출될 수 있다.

[0095] 단계(908)에서, 쿼리 결과에 액세스하는 데 사용될 수 있는 열거자를 반환하기 위해 GetEnumerator 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(910)에서, 쿼리의 첫번째 결과를 반환하는 메소드가 제공된다. 이들 메소드의 비동기 버전도 역시 제공될 수 있다. 첫번째 결과를 반환하기 위해 GetFirst 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 결과의 개수를 반환하기 위해 GetCount 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(912)에서, StorageSearcher 쿼리로부터 StorageView를 생성하기 위해 CreateView 메소드가 제공될 수 있다. CreateView는 뷰에 고유한 정보를 지정하기 위한 부가의 옵션을 갖거나 갖지 않는 StorageViewDefinition을 취할 수 있다.

[0096] StorageRecord 클래스는 쿼리가 임의의 특정의 애플리케이션 정의 유형에 대응하지 않는 데이터를 반환할 때 탐색기의 결과 유형으로서 사용된다. 예를 들어, Project 또는 Group 동작의 결과는 StorageRecord 객체의 컬렉션이다.

[0097] // StorageRecord는 구조체 유형의(structurally typed) 쿼리 결과로 값을 표현한다.

```
public class StorageRecord :
     System.ComponentModel.ICustomTypeDescriptor
```

[0098] {

[0099] // 필드의 값을 가져옴

```
   public object this[string name] { get; }
```

[0100] }

[0101] 도 10은 결과 세트를 보는 방법을 나타낸 것이다. 단계(1000)에서, 결과를 보기 위한 클래스가 제공된다. StorageView 클래스는 결과 세트에 대한 풍부한 애플리케이션 뷰를 제공한다. StorageView는 필터링, 정렬, 스크롤링, 그룹화, 섹션화, 섹션의 확장/축소, 기타 등등의 동작을 지원한다.

```
sealed public class StorageView<T> : IVirtualList,
     IServiceContainer, IEnumerable, IListSource, IDisposable
   where T: StorageViewRecord
{
   public StorageViewDefinition CopyDefinition() {}
   public void ApplyDefinition(StorageViewDefinition definition) {
      }

   public int Count { get;}
   public T Current { get;}
   public T this[ViewRecordBookmark bookmark] {get;}

   public T FindRecord(ViewRecordBookmark bookmark, bool forward,
     string expression, params object[] parameters) {}
   public T FindRecord(StorageViewSeekOrigin seekOrigin, bool
     forward, string expression, params object[] parameters) {}

   public void MoveCurrentPosition(StorageViewSeekOrigin
     seekOrigin, int offset);
   public void MoveCurrentPosition(ViewRecordBookmark bookmark, int
     offset);
```

[0102]

```
    public void Refresh() { }

    public ViewRecordBookmark GetBookmarkFromBinary(byte[] bookmark)
       { }
    public byte[] GetBinaryFromBookmark(ViewRecordBookmark bookmark)
       { }

    public void CollapseSection(params object[] sectionValues){}
    public void CollapseSection(ViewRecordBookmark bookmark){}
    public void ExpandSection(params object[] sectionValues){}
    public void ExpandSection(ViewRecordBookmark bookmark){}
    public void CollapseAllSections() { }
    public void ExpandAllSections() { }
    public void ExpandSectionLevel(int sectionLevel){}
    public void LoadSectionExpandState(System.Xml.XmlReader reader);
    public void SaveSectionExpandState(System.Xml.XmlWriter writer);

    public void SetExtendedFields(StorageViewRecord[] records,
      string fields);

    public IList IListSource.GetList();

    public event ViewChangedEventHandler ViewChanged;
}
```

[0103]

[0104] 단계(1002)에서, StorageViewDefinition의 새로운 인스턴스를 생성하기 위해 CopyDefinition 메소드가 제공될 수 있다. 지정된 StorageViewDefinition을 현재의 StorageView에 적용하기 위해 ApplyDefinition 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(1004)에서, 레코드를 찾고 레코드 수 및 현재의 레코드를 반환하기 위한 메소드가 제공된다. 지정된 필터에 따라 지정된 위치 또는 북마크에 대해 현재의 StorageView 내에서 StorageViewRecord를 찾기 위해 FindRecord 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 현재의 StorageView 내의 레코드의 수를 반환하기 위해 Count 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. StorageView 내의 현재의 StorageViewRecord를 반환하기 위해 Current 메소드가 제공될 수 있다.

[0105] 단계(1006)에서, 주어진 북마크(Bookmark)에 대한 StorageViewRecord를 반환하기 위해 색인 액세서(indexed accessor)(예를 들어, this[])가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(1008)에서, 위치를 이동시키고 뷰를 리프레쉬하기 위한 메소드가 제공된다. 지정된 위치 또는 북마크 및 오프셋에 따라 StorageView 내에서 현재 위치를 이동시키기 위해 MoveCurrentPosition 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 스토어로부터의 현재의 값으로 정적(static) StorageView 내의 데이터를 리프레쉬하기 위해 Refresh 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 단계(1010)에서, 북마크 및 그의 이진 표현을 가져오기 위한 메소드가 제공된다. 영속적 이진 표현으로부터 북마크를 가져오기 위해 GetBookmarkFromBinary 메소드가 제공될 수 있다. Bookmark로부터 영속적 이진 표현을 가져오기 위해 GetBinaryFromBookmark 메소드가 제공될 수 있다.

[0106] 단계(1012)에서, 섹션, 레벨 및 필드를 확장, 축소하기 위한 메소드가 제공된다. StorageView 내에 정의된 모든 섹션을 축소시키기 위해 CollapseAllSections 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. StorageView 내에 정의된 모든 섹션을 확장시키기 위해 ExpandAllSections 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 모든 섹션을 지정된 레벨(이 레벨을 포함함)까지 확장시키기 위해 ExpandSectionLevel 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다.

[0107] 단계(1014)에서, 레코드의 필드를 확대시키기 위한 메소드가 제공된다. 일련의 StorageViewRecord와 연관되어 있는 확대된 필드를 정의하기 위해 SetExtendedFields 메소드가 제공될 수 있다. 단계(1016)에서, 확장된 섹션의 상태를 저장 및 로드하기 위한 메소드가 제공된다. 확장된 일련의 섹션을 지정하는 상태를 로드하기 위해 LoadSectionExpandState 메소드가 제공될 수 있다. 이 메소드의 비동기 버전도 역시 제공될 수 있다. 확장된 일련의 섹션을 지정하는 상태를 저장하기 위해 SaveSectionExpandState 메소드가 제공될 수 있다. StorageView는 StorageView가 변경되었을 때 리스너(listener)에게 통지하기 위해 ViewChanged 이벤트를 노출시킬 수 있다.

[0108] 도 11은 결과에 대한 초기 데이터 뷰를 제공하는 방법을 나타낸 것이다. 단계(1100)에서, 초기 데이터 뷰를 정의하는 클래스가 제공된다. StorageViewDefinition 클래스는 StorageSearcher에 의해 정의된 결과에 대한 초기 데이터 뷰를 정의한다.

```
public class StorageViewDefinition
{
   public string Sort { get; set;}
   public IList<StorageViewSection> Sections { get;}
   public int SectionExpandLevel { get; set; }

   public string Filter { get; set;}
   public void SetFilter(string expression, params object[]
      parameters);

   public string Fields { get; set;}
   public void SetFields(string expression, params object[]
      parameters);

   public IDictionary<string,object> Parameters { get;}

   public bool AutoRefresh { get; set; }

   public int PageSize { get; set;}

}
```

[0109]

[0110] 단계(1102)에서, StorageView에 대한 정렬 조건을 가져오거나 설정하기 위해 Sort 속성이 제공될 수 있다. 단계(1104)에서, 섹션을 변경 및 확장하기 위한 속성이 제공된다. StorageView 내에 정의된 섹션들(Sections)의 리스트를 변경하기 위해 Sections 속성이 제공될 수 있다. 지정된 레벨까지(이 레벨을 포함함) 섹션을 확장시키기 위해 SectionExpandLevel 속성이 제공될 수 있다. 단계(1106)에서, 동작을 필터링하기 위한 속성 및 메소드가 제공된다. StorageView를 필터링하여 지정된 필터 조건과 일치하는 StorageViewRecord만을 노출시키기 위해 Filter 속성이 노출될 수 있다. StorageView를 필터링하여 지정된 파라미터를 사용하여 지정된 필터 조건과 일치하는 StorageViewRecord만을 노출시키기 위해 SetFilter 메소드가 노출될 수 있다. 단계(1108)에서, 노출된 필드를 제한하는 속성 및 메소드가 제공된다. StorageView에 의해 노출된 필드를 지정된 필드(Fields)로 제한하기 위해 Fields 속성이 노출될 수 있다. StorageView에 의해 노출된 필드를 지정된 파라미터를 사용하여 지정된 필드(Fields)로 제한하기 위해 Fields 메소드가 노출될 수 있다.

[0111] 단계(1110)에서, Filter, Sort 및 Sections에 의해 사용되는 파라미터를 열거하는 컬렉션이 제공된다. Filter, Sort 및 Sections 명세에 의해 사용되는 파라미터를 열거하는 Parameters 컬렉션이 노출될 수 있다. 단계(1112)에서, StorageView가 자동적으로 스토어에 대한 변경와 동기하여 유지되는지 여부를 지정하기 위해 부울 AutoRefresh 속성이 노출될 수 있다. 단계(1114)에서, 스토어로부터 한번에 검색될 StorageViewRecord의 수를 지정하기 위해 PageSize 속성이 노출될 수 있다.

[0112] 도 12는 스토리지 레코드 클래스를 확장하는 방법을 나타낸 것이다. 단계(1200)에서, 뷰 속성을 부가하는 클래스가 제공된다. StorageViewRecord는 StorageRecord를 확장하여, 섹션화 정보, 북마크 및 필드 설정자(field setter) 등의 StorageView 관련 속성을 부가한다. StorageViewRecord는 StorageViewRecord 내의 값들이 변경될 때 리스너에게 통지하기 위해 IPropertyChange를 지원한다.

```
public class StorageViewRecord : StorageRecord, IPropertyChange
{
   public virtual bool IsSectionRecord { get; }

   public virtual int SectionLevel { get; }
   public virtual string SectionName { get; }

   public virtual bool IsSectionExpanded { get; }

   public virtual ViewRecordBookmark Bookmark { get; }

   protected virtual void SetValueinRecord(int i, object value);
   protected virtual void SetValueinRecord(string name, object
      value);

}
```

[0113]

[0114] 단계(1202)에서, StorageViewRecord가 StorageView에서의 섹션 헤더 레코드를 나타내는지 여부를 반환하기 위해 IsSectionRecord 속성이 노출될 수 있다. 단계(1204)에서, 섹션 정보를 위한 속성이 제공된다. StorageView 내의 StorageViewRecord의 레벨을 반환하기 위해 SectionLevel 속성이 노출될 수 있다. StorageView 내의 섹션의 이름을 반환하기 위해 SectionName 속성이 노출될 수 있다. 섹션이 확장되는지 여부를 반환하기 위해 IsSectionExpanded 속성이 노출될 수 있다. 단계(1206)에서, 현재의 StorageViewRecord

에 대한 북마크를 반환하기 위해 Bookmark 속성이 노출될 수 있다. 단계(1208)에서, StorageViewRecord 내의 지정된 필드의 값을 설정하기 위해 SetValueinRecord 메소드가 노출될 수 있다. 필드는 이름(name) 또는 서수(ordinal)에 의해 지정될 수 있다.

[0115] StorageView 내에 섹션(그룹)을 정의하기 위해 StorageViewSection이 사용된다.

```
public class StorageViewSection
{
  public StorageViewSection(string field) {}

  public string Field { get; }

  public string AggregateFields { get; set; }

  public string Sort { get; set; }

  public string Having { get; set; }
  public void SetHaving(string expression, params object[]
    parameters);
}
```

[0116]

[0117] 섹션이 정의되고 있는 StorageView 내에 필드를 지정하는 StorageViewSection이 생성될 수 있다. 섹션이 정의되어 있는 StorageView 내의 필드를 반환하기 위해 Field 속성이 노출될 수 있다. 섹션에 대한 계산을 하기 위한 합계(aggregate)를 가져오거나 설정하기 위해 AggregateFields 속성이 노출될 수 있다. 섹션 내의 StorageViewRecord에 대한 순서를 지정하기 위해 Sort 속성이 노출될 수 있다. 지정된 Aggregate 필드에 따라 StorageViewRecord를 제한하기 위해 Having 속성이 노출될 수 있다. 일련의 파라미터와 함께 지정된 Aggregate 필드에 따라 StorageViewRecord를 제한하기 위해 SetHaving 메소드가 노출될 수 있다.

[0118] 채움(population)이 지연될 수 있는 강타입 객체들의 컬렉션을 나타내기 위해 StorageCollection<T> 클래스가 사용된다. 예를 들어, 부모 객체의 컬렉션 속성에서 StorageCollection이 사용될 수 있다. StorageCollection은 그의 콘텐츠에 액세스할 때 명시적으로 또는 암시적으로 채워질 수 있다.

```
public class StorageCollection<T> : ICollection<T>, IBindingList,
    ITypedList {
  public StorageCollection();
  public StorageCollection(object parent, StorageContext ctx,
    string role);
  public StorageContext Context { get; internal set;}
  public StorageDomain Domain { get; internal set;}

  public void Fill();
  public void Fill(StorageSearcher<T> searcher);
  public void Fill(IEnumerable<T> values);

  public bool IsFilled { get;}
  public void Reset();

  public StorageSearcher<T> Searcher { get; }

  public IEnumerator<T> GetEnumerator();

  // ICollection에 대한 지원

  bool ICollection<T>.Add(T obj);
  void ICollection<T>.Remove(T obj);
  void ICollection<T>.Clear();
  bool ICollection<T>.Contains(T t);
  public virtual int Count { get; }
  void ICollection<T>.CopyTo(T[] array, int arrayIndex);
  bool ICollection<T>.IsReadOnly { get { }}
}
```

[0119]

[0120]

[0121]

[0122]

[0123] StorageCollection은 StorageContext 또는 StorageDomain을 지정하는 정보, 부모 객체, 및 예를 들어 StorageCollection이 부모 객체의 컬렉션 속성 내의 객체들을 나타내는 경우 StorageCollection과 연관된 역할(role)로 생성될 수 있다.

[0124] StorageCollection과 연관된 StorageContext를 반환하기 위해 Context 속성이 제공될 수 있다. StorageCollection과 연관된 StorageDomain을 반환하기 위해 Domain 속성이 제공될 수 있다. 객체들을 컬렉

션에 부가하기 위해 Fill 메소드가 제공될 수 있다. Fill 메소드는 IEnumerable<T> 또는 StorageSearcher를 취하거나 StorageCollection을 채우라는 요청을 발생하기 위해 StorageDomain 또는 StorageContext와 함께 parent 및 role 속성을 사용할 수 있다. StorageCollection이 채워졌는지 여부를 반환하기 위해 IsFilled 속성이 노출될 수 있다. StorageCollection을 리셋시키기 위해 Reset 메소드가 노출될 수 있다.

[0125] 컬렉션의 정의에 대응하는 스토어에 대한 StorageSearcher을 반환하기 위해 Searcher 속성이 노출될 수 있다. StorageCollection의 콘텐츠에 걸친 열거자를 반환하기 위해 GetEnumerator 메소드가 노출될 수 있다. StorageCollection에 객체를 부가하기 위해 Add 메소드가 노출될 수 있다. StorageCollection으로부터 객체를 제거하기 위해 Remove 메소드가 노출될 수 있다. StorageCollection을 클리어시키기 위해 Clear 메소드가 노출될 수 있다. StorageCollection이 지정된 객체 인스턴스를 가지고 있는지 여부를 반환하기 위해 Contains 메소드가 노출될 수 있다. StorageCollection 내의 객체의 총수를 지정하기 위해 Count 메소드가 노출될 수 있다. 지정된 객체들을 StorageCollection으로 복사하기 위해 CopyTo 메소드가 노출될 수 있다. StorageCollection에 부가되거나 그로부터 제거될 수 있는지 여부를 반환하기 위해 IsReadOnly 속성이 노출될 수 있다.

[0126] 도 13은 CDP(1302)에 대한 스토리지 API(100)를 이용하는 시스템(1300)을 나타낸 것이다. 데이터 애플리케이션 및 애플리케이션 프레임워크(1304)와 데이터 스토어(1306) 상의 데이터 간의 데이터 관리를 제공하기 위해 CDP(1302)가 이용된다. CDP(1302)는 애플리케이션 프레임워크 및 그와 연관된 최종 사용자 애플리케이션에 걸쳐 공통인 데이터 서비스를 제공한다. CDP(1302)는 또한 애플리케이션 및 애플리케이션 프레임워크(1304), 런타임 컴포넌트(1308), 및 제약/보안 엔진(constraint/security engine) 컴포넌트(1310)와의 인터페이싱을 용이하게 해주는 API(100)를 포함한다. API(100)는 공개 클래스(public class), 인터페이스 및 정적 헬퍼(static helper) 함수 형태의 CDP(1302)를 사용하여 애플리케이션에 대한 프로그래밍 인터페이스를 제공한다. 예로는 *StorageContext, StorageSearcher, Entity, TableSet, Table, EntityReference,* 및 *TableReference*가 있다.

[0127] CDP 런타임 컴포넌트(1308)는 공개 API 계층(100)에 노출된 여러가지 특징을 구현하는 계층을 말한다. 이는 객체 관계형 매핑 및 쿼리 매핑을 제공함으로써, 데이터 모델 제약을 실시함으로써, 기타 등등에 의해 공통 데이터 모델을 구현한다. 보다 구체적으로는, CDP 런타임(1308)은 공통 데이터 모델 컴포넌트 구현, 쿼리 프로세서 컴포넌트, 세션 및 트랜잭션 컴포넌트, 세션 캐쉬 및 명시적 캐쉬(explicit cache)를 포함할 수 있는 객체 캐쉬, 변경 추적?충돌 검출 및 이벤팅(eventing)을 포함하는 서비스 컴포넌트, 커서 및 규칙 컴포넌트, 비지니스 로직 호스팅 컴포넌트, 및 코어 영구 보존(core persistence) 및 쿼리 서비스를 제공하는 영구 보존 및 쿼리 엔진을 포함한다. 영구 보존 및 쿼리 서비스 내부에는 쿼리/업데이트 매핑을 비롯한 객체 관계형 매핑이 있다. CDP(1302)는 또한 데이터 스토어(1306)에 대한 제약 및 보안 정책, 예를 들어 역할 기반 보안의 적용을 제공하는 제약/보안 엔진(1310)을 포함한다.

[0128] 이제부터 도 14를 참조하면, 개시된 API 아키텍처를 실행하는 동작을 하는 컴퓨터의 블록도가 도시되어 있다. 본 발명의 여러가지 측면에 대한 부가의 컨텍스트를 제공하기 위해, 도 14 및 이하의 설명은 본 발명의 여러가지 측면이 구현될 수 있는 적합한 컴퓨팅 환경(1400)에 대한 간략하고 일반적인 설명을 제공하기 위한 것이다. 본 발명이 하나 이상의 컴퓨터 상에서 실행될 수 있는 컴퓨터 실행가능 명령어의 일반적인 관점에서 기술되어 있지만, 당업자라면 본 발명이 또한 다른 프로그램 모듈들과 조합하여 및/또는 하드웨어 및 소프트웨어의 조합으로서 구현될 수 있음을 잘 알 것이다.

[0129] 일반적으로, 프로그램 모듈은 특정의 작업을 수행하거나 특정의 추상 데이터 유형을 구현하는 루틴, 프로그램, 컴포넌트, 데이터 구조, 기타 등등을 포함한다. 게다가, 당업자라면 본 발명의 방법들이 단일 프로세서 또는 멀티프로세서 컴퓨터 시스템, 미니컴퓨터, 메인프레임 컴퓨터는 물론 퍼스널 컴퓨터, 핸드-헬드 컴퓨팅 장치, 마이크로프로세서-기반 또는 프로그램가능 가전 제품, 및 기타 등등(이들 각각은 하나 이상의 연관된 장치와 연결되어 동작할 수 있음)을 비롯한 다른 컴퓨터 시스템 구성에서 실시될 수 있음을 잘 알 것이다.

[0130] 본 발명의 예시된 측면들은 또한 어떤 작업들이 통신 네트워크를 통해 연결되어 있는 원격 프로세싱 장치들에 의해 수행되는 분산 컴퓨팅 환경에서 실시될 수 있다. 분산 컴퓨팅 환경에서, 프로그램 모듈들은 로컬 및 원격 메모리 저장 장치 둘다에 위치할 수 있다.

[0131] 컴퓨터는 일반적으로 각종의 컴퓨터 판독가능 매체를 포함한다. 컴퓨터 판독가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 이용가능한 매체일 수 있으며 휘발성 및 비휘발성, 분리형 및 비분리형 매체 둘다를 포

함한다. 제한이 아닌 예로서, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체 및 통신 매체를 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터 등의 정보의 저장을 위한 임의의 방법 또는 기술로 구현된 휘발성 및 비휘발성, 분리형 및 비분리형 매체 둘다를 포함한다. 컴퓨터 저장 매체는 RAM, ROM, EEPROM, 플래쉬 메모리 또는 기타 메모리 기술, CD-ROM, DVD(digital video disk) 또는 기타 광학 디스크 저장 장치, 자기 카세트, 자기 테이프, 자기 디스크 저장 장치 또는 기타 자기 저장 장치, 또는 원하는 정보를 저장하는 데 사용될 수 있고 컴퓨터에 의해 액세스될 수 있는 임의의 다른 매체를 포함하지만, 이에 한정되는 것은 아니다.

[0132] 통신 매체는 일반적으로 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터를 반송파 또는 기타 전송 메카니즘 등의 변조된 데이터 신호로 구현하며, 임의의 정보 전달 매체를 포함한다. 용어 "변조된 데이터 신호"는 그의 특성 중 하나 이상이 그 신호에 정보를 인코딩하는 방식으로 설정 또는 변경된 신호를 의미한다. 제한이 아닌 예로서, 통신 매체는 유선 네트워크 또는 직접 유선 연결 등의 유선 매체, 및 음향, RF, 적외선 및 기타 무선 매체 등의 무선 매체를 포함한다. 상기한 것들 중 임의의 것의 조합도 역시 컴퓨터 판독가능 매체의 범위 내에 포함되어야 한다.

[0133] 다시 도 14를 참조하면, 본 발명의 여러가지 측면을 구현하는 예시적인 환경(1400)은 컴퓨터(1402)를 포함하고, 이 컴퓨터(1402)는 프로세싱 유닛(1404), 시스템 메모리(1406) 및 시스템 버스(1408)를 포함한다. 시스템 버스(1408)는 시스템 메모리(1406)(이에 한정되는 것은 아님)를 비롯한 시스템 컴포넌트를 프로세싱 유닛(1404)에 연결시킨다. 프로세싱 유닛(1404)은 여러가지 상업적으로 이용가능한 프로세서 중 임의의 것일 수 있다. 듀얼 마이크로프로세서 및 기타의 멀티프로세서 아키텍처도 역시 프로세싱 유닛(1404)으로서 이용될 수 있다.

[0134] 시스템 버스(1408)는 (메모리 콘트롤러가 갖거나 갖지 않는) 메모리 버스, 주변 버스, 및 각종의 상업적으로 이용가능한 버스 아키텍처 중 임의의 것을 사용하는 로컬 버스에 추가적으로 상호연결될 수 있는 몇가지 유형의 버스 구조 중 임의의 것일 수 있다. 시스템 메모리(1406)는 판독 전용 메모리(ROM)(1410) 및 랜덤 액세스 메모리(RAM)(1412)를 포함한다. 기본 입/출력 시스템(BIOS)은 ROM, EPROM, EEPROM 등의 비휘발성 메모리(1410)에 저장되며, 이 BIOS는 시동 중과 같은 때에 컴퓨터(1402) 내의 구성요소들 간의 정보의 전송을 돕는 기본적인 루틴을 포함한다. RAM(1412)은 또한 데이터를 캐싱하기 위한 정적 RAM(static RAM) 등의 고속 RAM을 포함할 수 있다.

[0135] 컴퓨터(1402)는 또한 내장형 하드 디스크 드라이브(HDD)(1414)(예를 들어, EIDE, SATA) - 이 내장형 하드 디스크 드라이브(1414)는 또한 적당한 샤시(도시 생략)에 넣어 외장형으로 사용하도록 구성될 수 있음 -, 자기 플로피 디스크 드라이브(FDD)(1416)(예를 들어, 분리형 디스켓(1418)으로부터 판독하거나 그에 기록함), 및 광학 디스크 드라이브(1420)(예를 들어, CD-ROM 디스크(1422)로부터 판독하거나, DVD 등의 기타 고용량 광학 매체로부터 판독하거나 그에 기록함)를 포함한다. 하드 디스크 드라이브(1414), 자기 디스크 드라이브(1416) 및 광학 디스크 드라이브(1420)는 각각 하드 디스크 드라이브 인터페이스(1424), 자기 디스크 드라이브 인터페이스(1426), 및 광학 드라이브 인터페이스(1428)에 의해 시스템 버스(1408)에 연결될 수 있다. 외장형 드라이브 구현을 위한 인터페이스(1424)는 USB(Universal Serial Bus) 및 IEEE 1394 인터페이스 기술 중 적어도 하나 또는 둘다를 포함한다. 다른 외장형 드라이브 연결 기술은 본 발명의 범위 내에 속한다.

[0136] 드라이브 및 그의 연관된 컴퓨터 판독가능 매체는 데이터, 데이터 구조, 컴퓨터 실행가능 명령어, 기타 등등의 비휘발성 저장을 제공한다. 컴퓨터(1402)에 있어서, 드라이브 및 매체는 임의의 데이터를 적당한 디지털 포맷으로 저장하기 위한 것이다. 이상에서의 컴퓨터 판독가능 매체에 대한 설명이 HDD, 분리형 자기 디스켓 및 CD 또는 DVD 등의 분리형 광학 매체를 언급하고 있지만, 당업자라면 ZIP 드라이브, 자기 카세트, 플래쉬 메모리 카드, 카트리지 및 기타 등등의 컴퓨터에 의해 판독가능한 다른 유형의 매체도 역시 예시적인 오퍼레이팅 환경에서 사용될 수 있음과 또한 임의의 이러한 매체가 본 발명의 방법을 수행하는 컴퓨터 실행가능 명령어를 포함할 수 있음을 잘 알 것이다.

[0137] 오퍼레이팅 시스템(1430), 하나 이상의 애플리케이션 프로그램(1432), 기타 프로그램 모듈(1434) 및 프로그램 데이터(1436)를 비롯한 다수의 프로그램 모듈이 드라이브 및 RAM(1412)에 저장될 수 있다. 오퍼레이팅 시스템, 애플리케이션, 모듈, 및/또는 데이터의 전부 또는 그 일부분은 또한 RAM(1412)에 캐싱될 수 있다. 본 발명이 상업적으로 이용가능한 오퍼레이팅 시스템 또는 오퍼레이팅 시스템들의 조합에서 구현될 수 있음을 잘 알 것이다.

[0138] 사용자는 하나 이상의 유선/무선 입력 장치, 예를 들어 키보드(1438) 및 마우스(1440) 등의 포인팅 장치를 통

해 명령 및 정보를 컴퓨터(1420)에 입력할 수 있다. 다른 입력 장치(도시 생략)으로는 마이크로폰, IR 리모콘, 조이스틱, 게임 패드, 스타일러스 펜, 터치 스크린, 기타 등등이 있을 수 있다. 이들 및 다른 입력 장치는 종종 시스템 버스(1408)에 연결되어 있는 입력 장치 인터페이스(1442)를 통해 프로세싱 유닛(1404)에 연결되지만, 병렬 포트, IEEE 1394 직렬 포트, 게임 포트, USB 포트, IR 인터페이스, 기타 등등의 다른 인터페이스에 의해 연결될 수 있다.

[0139] 모니터(1444) 또는 다른 유형의 디스플레이 장치도 역시 비디오 어댑터(1446) 등의 인터페이스를 통해 시스템 버스(1408)에 연결된다. 모니터(1444) 이외에, 컴퓨터는 일반적으로 스피커, 프린터, 기타 등등의 다른 주변 출력 장치(도시 생략)를 포함한다.

[0140] 컴퓨터(1402)는 원격 컴퓨터(들)(1448) 등의 하나 이상의 원격 컴퓨터로의 유선 및/또는 무선 통신을 통해 논리적 연결을 사용하여 네트워크화된 환경에서 동작할 수 있다. 원격 컴퓨터(들)(1448)는 워크스테이션, 서버 컴퓨터, 라우터, 퍼스널 컴퓨터, 휴대용 컴퓨터, 마이크로프로세서 기반 오락 기기, 피어 장치(peer device) 또는 다른 통상의 네트워크 노드일 수 있으며, 일반적으로 컴퓨터(1402)와 관련하여 기술된 구성요소들 대부분 또는 그 전부를 포함하지만, 간략함을 위해 메모리/저장 장치(1450)만이 예시되어 있다. 도시된 논리적 연결은 근거리 통신망(LAN)(1452) 및/또는 보다 대규모의 네트워크, 예를 들어 원거리 통신망(WAN)(1454)에의 유선/무선 연결을 포함한다. 이러한 LAN 및 WAN 네트워킹 환경은 사무실 및 기업에서 통상적인 것이며, 인트라넷 등의 전사적 컴퓨터 네트워크를 용이하게 해주며, 이들 모두는 글로벌 통신 네트워크, 예를 들어 인터넷에 연결될 수 있다.

[0141] LAN 네트워킹 환경에서 사용될 때, 컴퓨터(1402)는 유선 및/또는 무선 통신 네트워크 인터페이스 또는 어댑터(1456)를 통해 로컬 네트워크(1452)에 연결된다. 어댑터(1456)는 무선 어댑터(1456)와의 통신을 위해 배치되어 있는 무선 액세스 포인트를 포함할 수 있는 LAN(1452)으로의 유선 또는 무선 통신을 용이하게 해줄 수 있다.

[0142] WAN 네트워킹 환경에서 사용될 때, 컴퓨터(1402)는 모뎀(1458)을 포함할 수 있거나, WAN(1454) 상의 통신 서버에 연결되어 있거나, 또는 인터넷을 통하는 등 WAN(1454)을 통해 통신을 설정하는 다른 수단을 갖는다. 내장형 또는 외장형 및 유선 또는 무선 장치일 수 있는 모뎀(1458)은 직렬 포트 인터페이스(1442)를 통해 시스템 버스(1408)에 연결된다. 네트워크화된 환경에서, 컴퓨터(1402)와 관련하여 도시된 프로그램 모듈 또는 그의 일부분은 원격 메모리/저장 장치(1450)에 저장될 수 있다. 도시된 네트워크 연결이 예시적인 것이고 컴퓨터들 간의 통신 링크를 설정하는 다른 수단이 사용될 수 있음을 잘 알 것이다.

[0143] 컴퓨터(1402)는 무선 통신을 하도록 배치된 임의의 무선 장치 또는 개체, 예를 들어 프린터, 스캐너, 데스크톱 및/또는 휴대용 컴퓨터, 개인 휴대 단말기, 통신 위성, 무선 검출가능 태그와 연관된 임의의 장비 또는 장소(예를 들어, 키오스크, 신문 가판대, 휴게실), 및 전화와 통신하는 동작을 한다. 이것은 적어도 Wi-Fi 및 블루투스™ 무선 기술을 포함한다. 따라서, 통신은 종래의 네트워크에서와 같이 사전 정의된 구조이거나 적어도 2개의 장치 간의 애드혹 통신(ad hoc communication)일 수 있다.

[0144] Wi-Fi(Wireless Fidelity)는 유선 없이 가정의 소파, 호텔 객실의 침대, 또는 직장의 회의실에의 연결을 가능하게 해준다. Wi-Fi는 이러한 장치, 예를 들어 컴퓨터가 실내 및 실외에서, 즉 기지국의 통화권 내의 어느 곳에서라도 데이터를 전송 및 수신할 수 있게 해주는 셀 전화에서 사용되는 것과 유사한 무선 기술이다. Wi-Fi 네트워크는 안전하고 신뢰성있으며 고속인 무선 연결을 제공하기 위해 IEEE 802.11(a, b, g, 등등)이라고 하는 무선 기술을 사용한다. Wi-Fi 네트워크는 컴퓨터를 서로, 인터넷에 또한 (IEEE 802.3 또는 이더넷을 사용하는) 유선 네트워크에 연결하는 데 사용될 수 있다. Wi-Fi 네트워크는 비인가 2.4 및 5 GHz 무선 대역에서, 11 Mbps(802.11a) 또는 54 Mbps(802.11b) 데이터 레이트로, 또는 양 대역을 포함하는 제품에서 동작하며, 따라서 이 네트워크는 많은 사무실에서 사용되는 기본적인 10BaseT 유선 이더넷 네트워크와 유사한 실세계 성능을 제공할 수 있다.

[0145] 이제부터 도 15를 참조하면, 본 발명에 따른 예시적인 컴퓨팅 환경(1500)의 개략 블록도가 도시되어 있다. 시스템(1500)은 하나 이상의 클라이언트(들)(1502)를 포함한다. 클라이언트(들)(1500)는 하드웨어 및/또는 소프트웨어(예를 들어, 쓰레드, 프로세서, 컴퓨팅 장치)일 수 있다. 클라이언트(들)(1502)는 예를 들어 본 발명을 이용함으로써 쿠키(들) 및/또는 연관된 컨텍스트 정보를 저장해둘 수 있다.

[0146] 시스템(1500)은 또한 하나 이상의 서버(들)(1504)를 포함한다. 서버(들)(1504)도 역시 하드웨어 및/또는 소프트웨어(예를 들어, 쓰레드, 프로세스, 컴퓨팅 장치)일 수 있다. 서버(1504)는 예를 들어 본 발명을 이용함으로써 변환(transformation)을 수행하는 쓰레드를 저장해둘 수 있다. 클라이언트(1502)와 서버(1504) 간의

한가지 가능한 통신은 2개 이상의 컴퓨터 프로세스 간에 전송되도록 구성되어 있는 데이터 패킷 형태일 수 있다. 데이터 패킷은 예를 들어 쿠키 및/또는 연관된 컨텍스트 정보를 포함할 수 있다. 시스템(1500)은 클라이언트(들)(1502)와 서버(들)(1504) 간의 통신을 용이하게 해주기 위해 이용될 수 있는 통신 프레임워크(1506)(예를 들어, 인터넷 등의 글로벌 통신 네트워크)를 포함한다.

[0147] 통신은 유선(광 파이버를 포함함) 및/또는 무선 기술을 통해 용이하게 될 수 있다. 클라이언트(들)(1502)는 클라이언트(들)(1502)에 로컬인 정보(예를 들어, 쿠키(들) 및/또는 연관된 컨텍스트 정보)를 저장하는 데 이용될 수 있는 하나 이상의 클라이언트 데이터 스토어(들)(1508)와 연결되어 동작한다. 이와 유사하게, 서버(들)(1504)는 서버(1504)에 로컬인 정보를 저장하는 데 이용될 수 있는 하나 이상의 서버 데이터 스토어(들)(1510)와 연결되어 동작한다.

[0148] 이상에 기술된 바는 개시된 발명의 예들을 포함한다. 물론, 모든 생각할 수 있는 컴포넌트 및/또는 방법의 조합을 기술할 수는 없지만, 당업자라면 많은 추가의 조합 및 치환이 가능함을 잘 알 것이다. 따라서, 본 발명은 청구된 청구항의 정신 및 범위 내에 속하는 이러한 변경, 수정 및 변형 모두를 포괄하는 것으로 보아야 한다. 게다가, 상세한 설명 또는 청구항에서 용어 "포함하는"이 사용되는 범위까지, 이러한 용어는 "포함하는"이 청구항에서 이행구로서 사용될 때의 용어 "포함하는"이 해석되는 것과 유사하게 포함적인 것으로 보아야 한다.

발명의 효과

[0149] 본 발명에 따르면, 데이터 플랫폼에 대한 애플리케이션 프로그램 인터페이스(API)가 제공되고, 이 데이터 플랫폼은 복수의 개별적인 애플리케이션 프레임워크에 의해 액세스가능한 데이터 서비스를 제공하기 위해 데이터 스토어와 인터페이스하는 공통 데이터 플랫폼일 수 있으며, 이 데이터 서비스는 서로 다른 프레임워크의 대응하는 애플리케이션이 데이터 스토어에 액세스할 수 있게 된다.

도면의 간단한 설명

[0001] 도 1은 본 발명의 한 측면에 따른 데이터 플랫폼의 스토리지 애플리케이션 프로그램 인터페이스(API)를 나타낸 도면.

[0002] 도 2는 본 발명의 개시된 측면에 따른 스토리지 API를 제공하는 방법을 나타낸 도면.

[0003] 도 3은 스토리지 API의 일반 데이터 액세스 컴포넌트를 보다 상세히 나타낸 도면.

[0004] 도 4는 데이터 모델에 대한 스토리지 API를 제공하는 방법을 나타낸 도면.

[0005] 도 5는 테이블 세트 유형을 노출시키는 방법을 나타낸 도면.

[0006] 도 6은 API 내에 WinFS 기능을 제공하는 방법을 나타낸 도면.

[0007] 도 7은 스토어 내의 클래스를 표현하는 방법을 나타낸 도면.

[0008] 도 8은 클라이언트와 하나 이상의 스토어 간의 연결을 캡슐화하는 방법을 나타낸 도면.

[0009] 도 9는 스토어에 대한 쿼리를 작성하는 방법을 나타낸 도면.

[0010] 도 10은 결과 세트를 보는 방법을 나타낸 도면.

[0011] 도 11은 결과에 대한 초기 데이터 뷰를 제공하는 방법을 나타낸 도면.

[0012] 도 12는 스토리지 레코드 클래스를 확장하는 방법을 나타낸 도면.

[0013] 도 13은 공통 데이터 플랫폼에 대한 스토리지 API를 이용하는 시스템을 나타낸 도면.

[0014] 도 14는 개시된 아키텍처를 실행하는 동작을 하는 컴퓨터의 블록도.

[0015] 도 15는 예시적인 컴퓨팅 환경의 개략 블록도.

[0016] <도면의 주요 부분에 대한 부호의 설명>

[0017] 100: 데이터 플랫폼 API

[0018] 102: 데이터 플랫폼

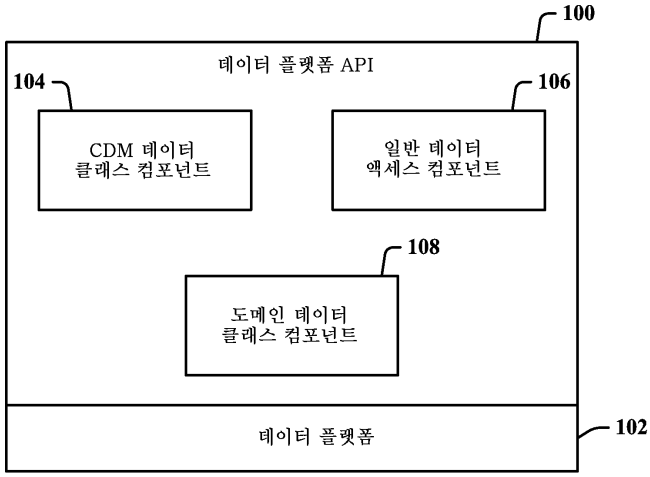
[0019] 104: CDM 데이터 클래스 컴포넌트

[0020] 106: 일반 데이터 액세스 컴포넌트

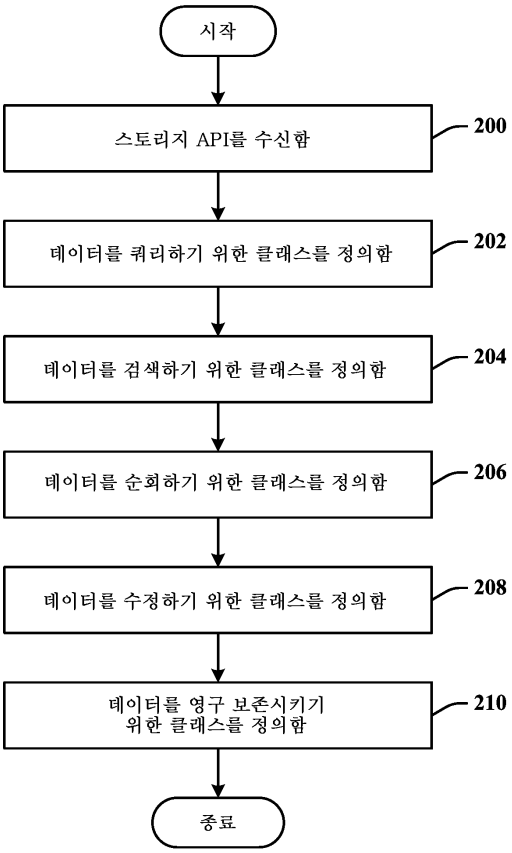
[0021] 108: 도메인 데이터 클래스 컴포넌트

도면

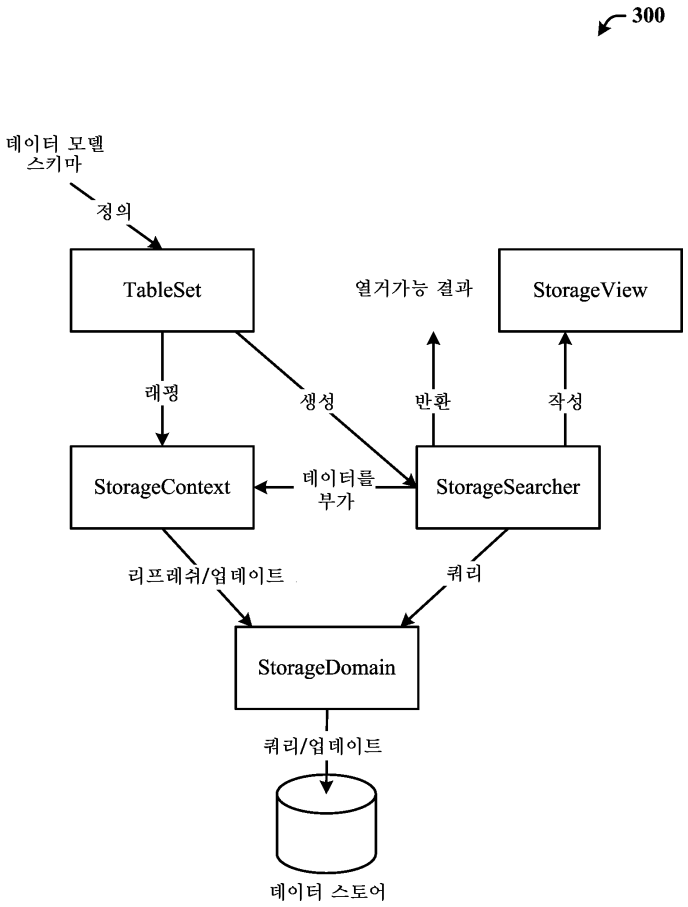
도면1



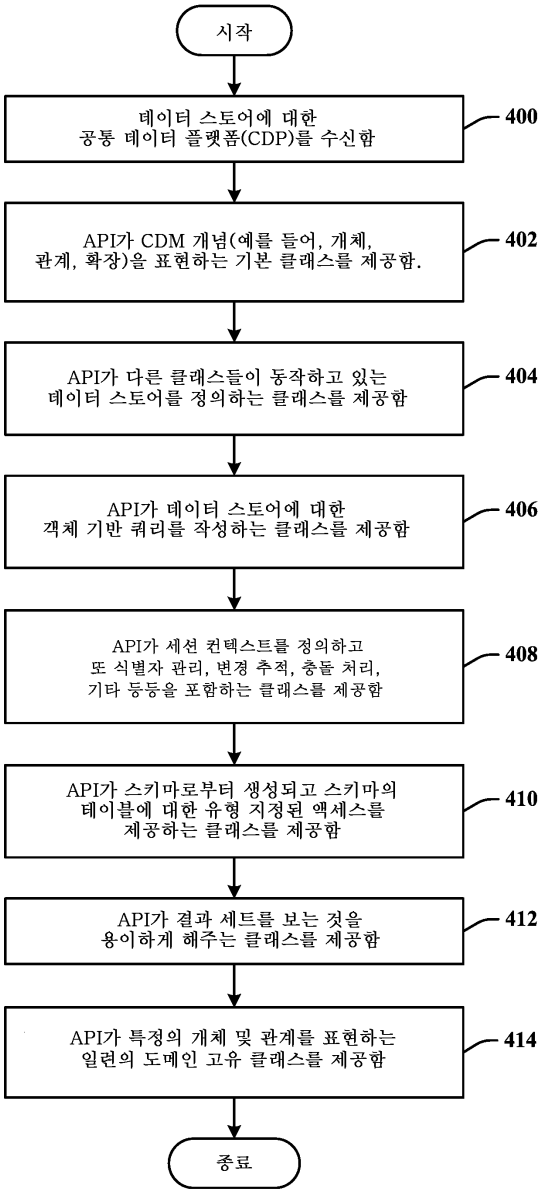
도면2



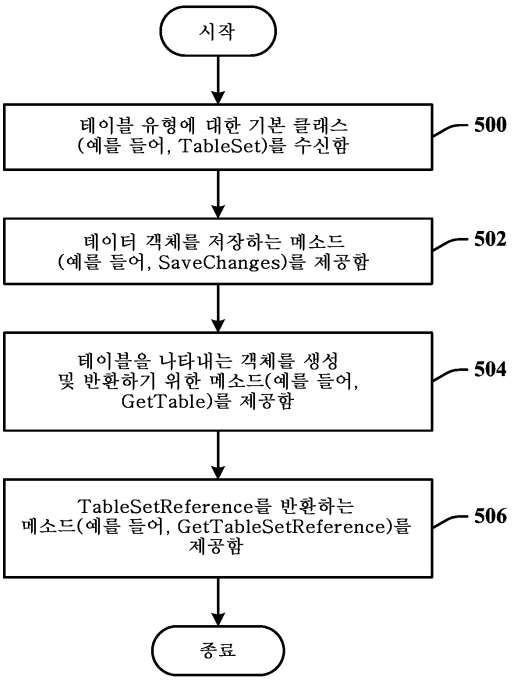
도면3


300
데이터 모델
스키마
정의
TableSet
열거가능 결과
StorageView
래핑
생성
반환
작성
StorageContext
데이터를
부가
StorageSearcher
리프레쉬/업데이트
쿼리
StorageDomain
쿼리/업데이트
데이터 스토어

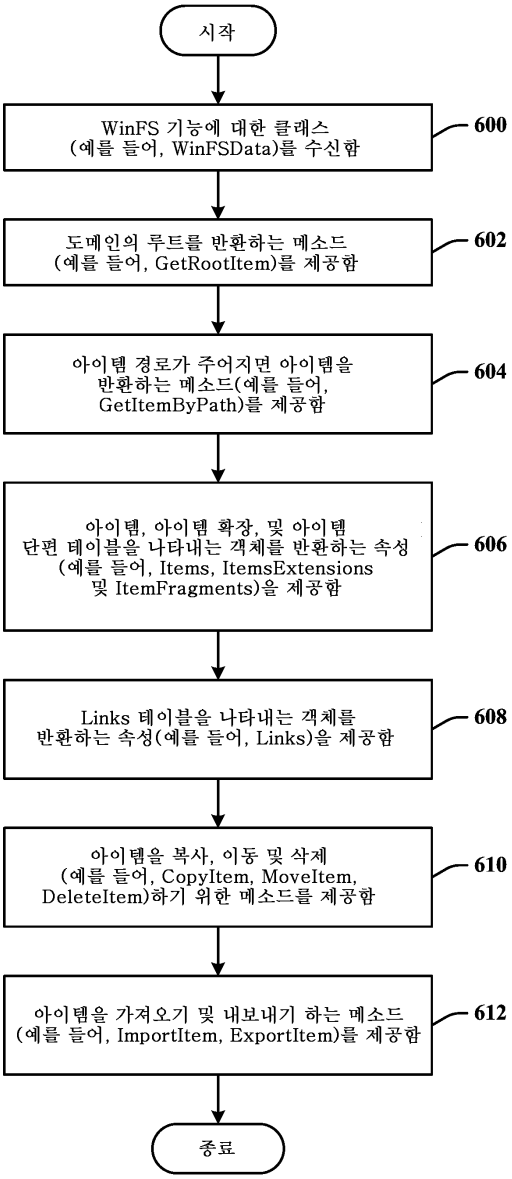
도면4


시작
데이터 스토어에 대한
공통 데이터 플랫폼(CDP)를 수신함
400
API가 CDM 개념(예를 들어, 개체,
관계, 확장)을 표현하는 기본 클래스를 제공함.
402
API가 다른 클래스들이 동작하고 있는
데이터 스토어를 정의하는 클래스를 제공함
404
API가 데이터 스토어에 대한
객체 기반 쿼리를 작성하는 클래스를 제공함
406
API가 세션 컨텍스트를 정의하고
또 식별자 관리, 변경 추적, 충돌 처리,
기타 등등을 포함하는 클래스를 제공함
408
API가 스키마로부터 생성되고 스키마의
테이블에 대한 유형 지정된 액세스를
제공하는 클래스를 제공함
410
API가 결과 세트를 보는 것을
용이하게 해주는 클래스를 제공함
412
API가 특정의 개체 및 관계를 표현하는
일련의 도메인 고유 클래스를 제공함
414
종료

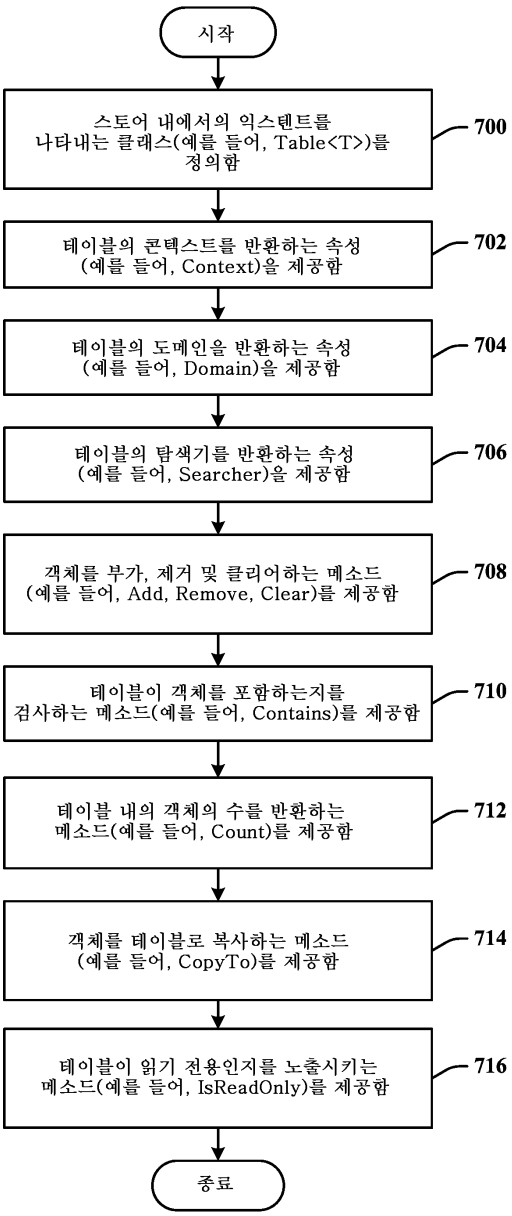
도면5


시작
테이블 유형에 대한 기본 클래스 (예를 들어, TableSet)를 수신함
500
데이터 객체를 저장하는 메소드 (예를 들어, SaveChanges)를 제공함
502
테이블을 나타내는 객체를 생성 및 반환하기 위한 메소드(예를 들어, GetTable)를 제공함
504
TableSetReference를 반환하는 메소드(예를 들어, GetTableSetReference)를 제공함
506
종료

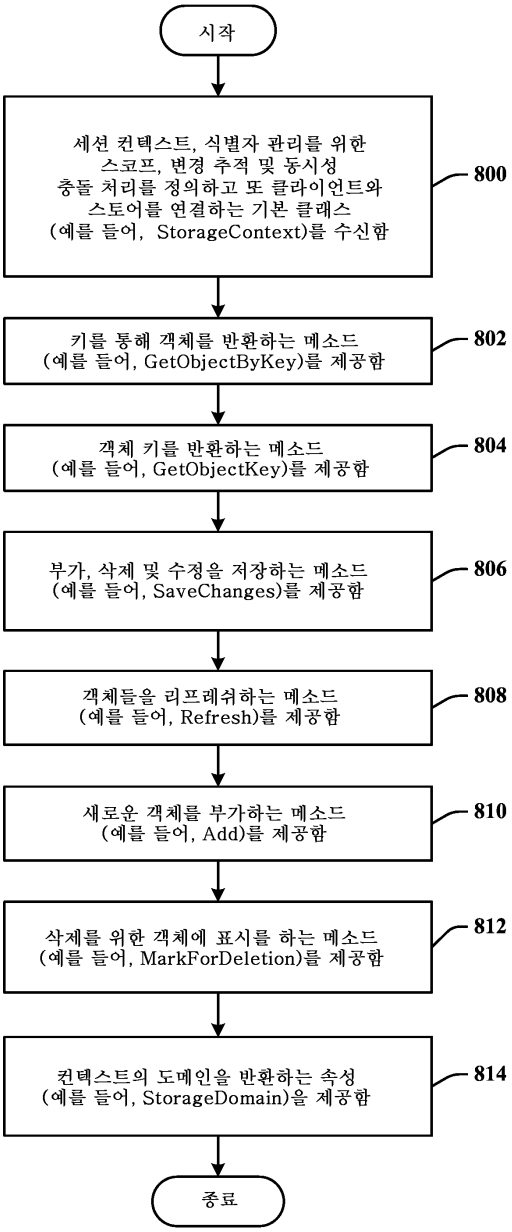
도면6


시작
WinFS 기능에 대한 클래스
(예를 들어, WinFSData)를 수신함
600
도메인의 루트를 반환하는 메소드
(예를 들어, GetRootItem)를 제공함
602
아이템 경로가 주어지면 아이템을
반환하는 메소드(예를 들어,
GetItemByPath)를 제공함
604
아이템, 아이템 확장, 및 아이템
단편 테이블을 나타내는 객체를 반환하는 속성
(예를 들어, Items, ItemsExtensions
및 ItemFragments)을 제공함
606
Links 테이블을 나타내는 객체를
반환하는 속성(예를 들어, Links)을 제공함
608
아이템을 복사, 이동 및 삭제
(예를 들어, CopyItem, MoveItem,
DeleteItem)하기 위한 메소드를 제공함
610
아이템을 가져오기 및 내보내기 하는 메소드
(예를 들어, ImportItem, ExportItem)를 제공함
612
종료

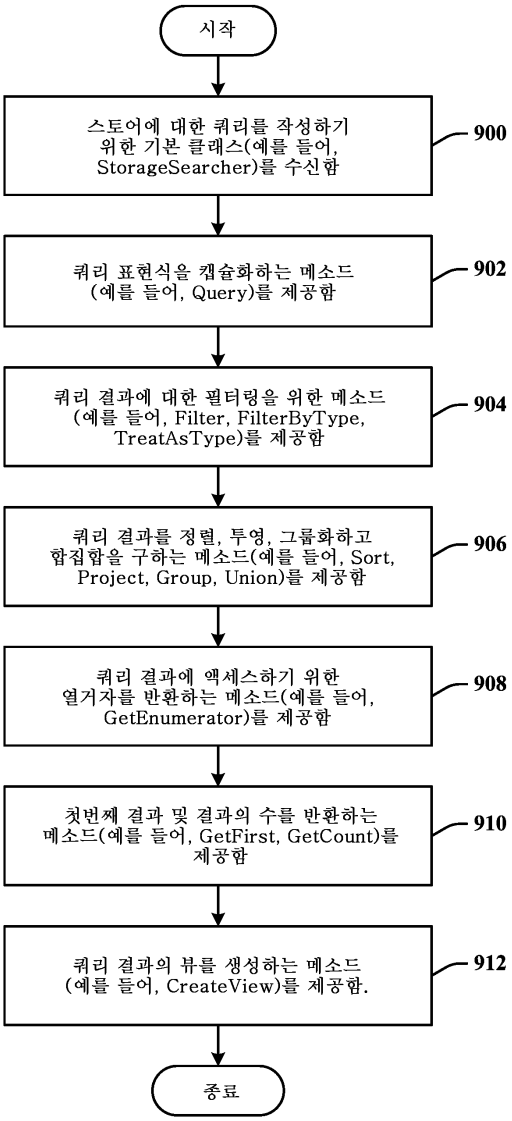
도면7


시작
스토어 내에서의 익스텐트를 나타내는 클래스(예를 들어, Table<T>)를 정의함
700
테이블의 콘텍스트를 반환하는 속성 (예를 들어, Context)을 제공함
702
테이블의 도메인을 반환하는 속성 (예를 들어, Domain)을 제공함
704
테이블의 탐색기를 반환하는 속성 (예를 들어, Searcher)을 제공함
706
객체를 부가, 제거 및 클리어하는 메소드 (예를 들어, Add, Remove, Clear)를 제공함
708
테이블이 객체를 포함하는지를 검사하는 메소드(예를 들어, Contains)를 제공함
710
테이블 내의 객체의 수를 반환하는 메소드(예를 들어, Count)를 제공함
712
객체를 테이블로 복사하는 메소드 (예를 들어, CopyTo)를 제공함
714
테이블이 읽기 전용인지를 노출시키는 메소드(예를 들어, IsReadOnly)를 제공함
716
종료

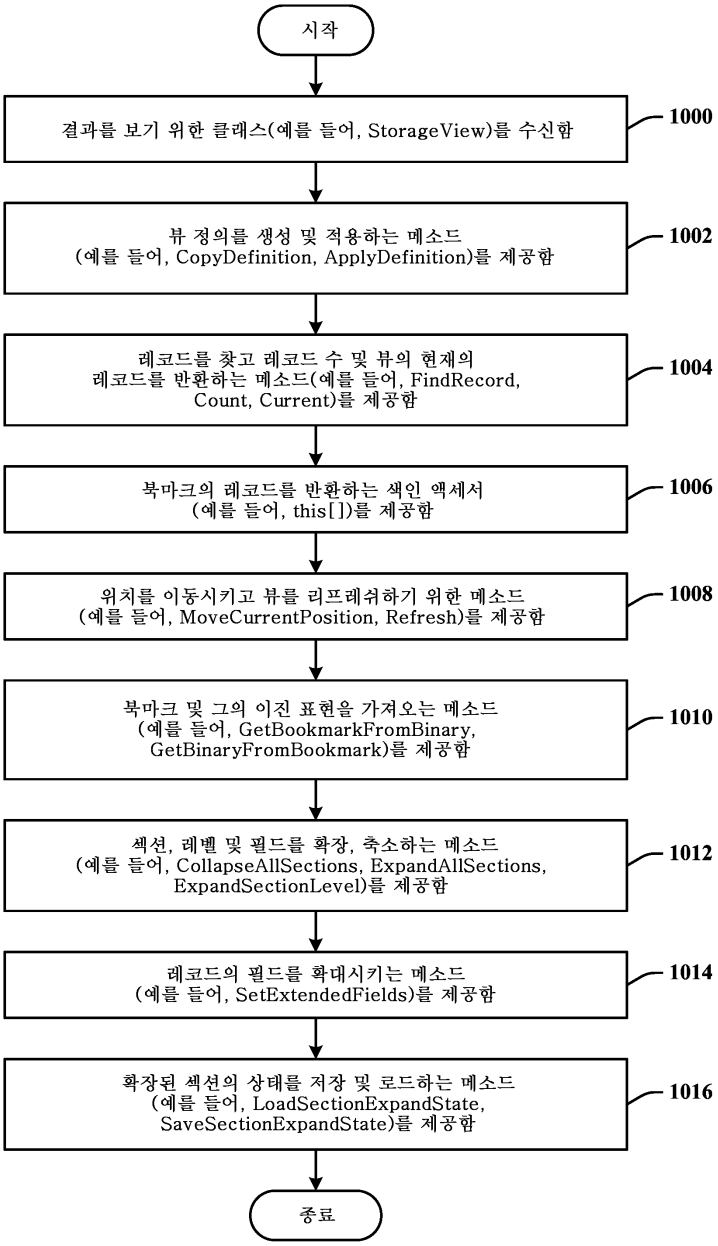
도면8


시작
세션 컨텍스트, 식별자 관리를 위한 스코프, 변경 추적 및 동시성 충돌 처리를 정의하고 또 클라이언트와 스토어를 연결하는 기본 클래스 (예를 들어, StorageContext)를 수신함
800
키를 통해 객체를 반환하는 메소드 (예를 들어, GetObjectByKey)를 제공함
802
객체 키를 반환하는 메소드 (예를 들어, GetObjectKey)를 제공함
804
부가, 삭제 및 수정을 저장하는 메소드 (예를 들어, SaveChanges)를 제공함
806
객체들을 리프레쉬하는 메소드 (예를 들어, Refresh)를 제공함
808
새로운 객체를 부가하는 메소드 (예를 들어, Add)를 제공함
810
삭제를 위한 객체에 표시를 하는 메소드 (예를 들어, MarkForDeletion)를 제공함
812
컨텍스트의 도메인을 반환하는 속성 (예를 들어, StorageDomain)을 제공함
814
종료

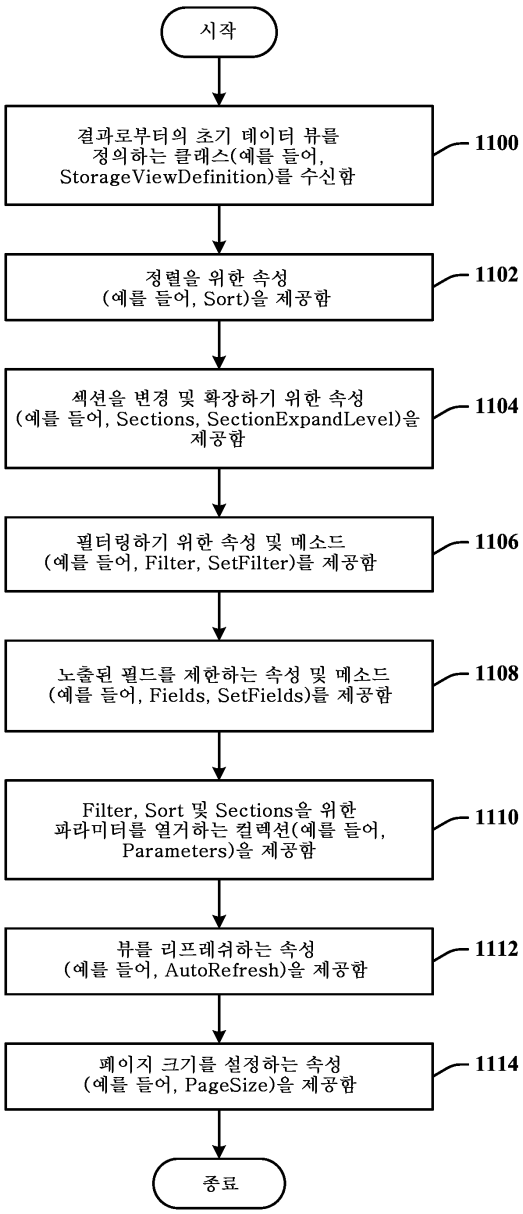
도면9


시작
스토어에 대한 쿼리를 작성하기 위한 기본 클래스(예를 들어, StorageSearcher)를 수신함
900
쿼리 표현식을 캡슐화하는 메소드 (예를 들어, Query)를 제공함
902
쿼리 결과에 대한 필터링을 위한 메소드 (예를 들어, Filter, FilterByType, TreatAsType)를 제공함
904
쿼리 결과를 정렬, 투영, 그룹화하고 합집합을 구하는 메소드(예를 들어, Sort, Project, Group, Union)를 제공함
906
쿼리 결과에 액세스하기 위한 열거자를 반환하는 메소드(예를 들어, GetEnumerator)를 제공함
908
첫번째 결과 및 결과의 수를 반환하는 메소드(예를 들어, GetFirst, GetCount)를 제공함
910
쿼리 결과의 뷰를 생성하는 메소드 (예를 들어, CreateView)를 제공함.
912
종료

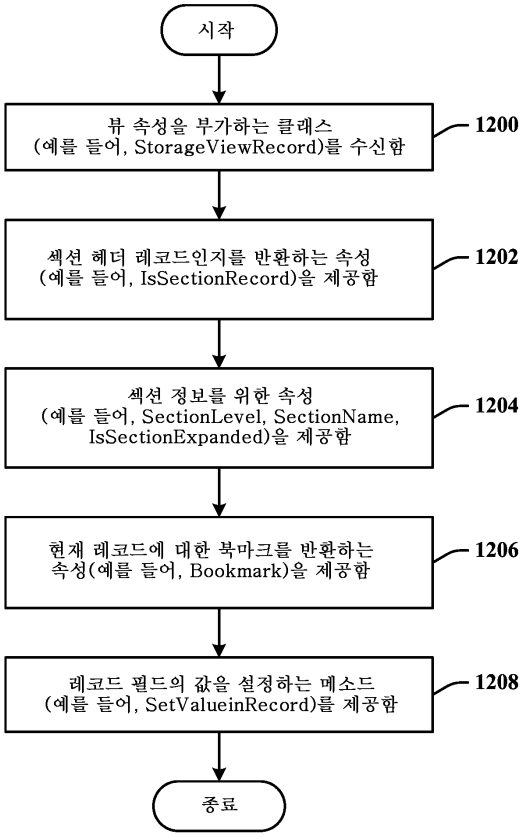
도면10


시작
결과를 보기 위한 클래스(예를 들어, StorageView)를 수신함
1000
뷰 정의를 생성 및 적용하는 메소드
(예를 들어, CopyDefinition, ApplyDefinition)를 제공함
1002
레코드를 찾고 레코드 수 및 뷰의 현재의
레코드를 반환하는 메소드(예를 들어, FindRecord,
Count, Current)를 제공함
1004
북마크의 레코드를 반환하는 색인 액세서
(예를 들어, this[])를 제공함
1006
위치를 이동시키고 뷰를 리프레쉬하기 위한 메소드
(예를 들어, MoveCurrentPosition, Refresh)를 제공함
1008
북마크 및 그의 이진 표현을 가져오는 메소드
(예를 들어, GetBookmarkFromBinary,
GetBinaryFromBookmark)를 제공함
1010
섹션, 레벨 및 필드를 확장, 축소하는 메소드
(예를 들어, CollapseAllSections, ExpandAllSections,
ExpandSectionLevel)를 제공함
1012
레코드의 필드를 확대시키는 메소드
(예를 들어, SetExtendedFields)를 제공함
1014
확장된 섹션의 상태를 저장 및 로드하는 메소드
(예를 들어, LoadSectionExpandState,
SaveSectionExpandState)를 제공함
1016
종료

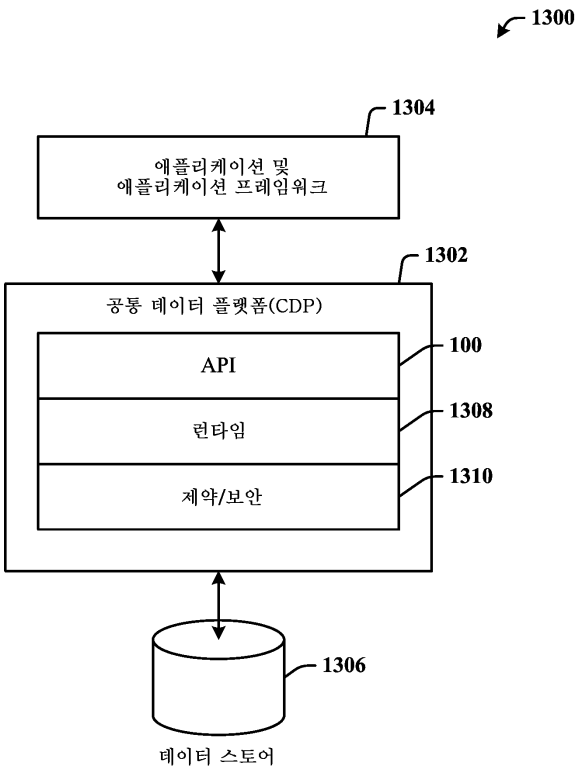
도면11


시작
결과로부터의 초기 데이터 뷰를 정의하는 클래스(예를 들어, StorageViewDefinition)를 수신함
1100
정렬을 위한 속성 (예를 들어, Sort)을 제공함
1102
섹션을 변경 및 확장하기 위한 속성 (예를 들어, Sections, SectionExpandLevel)을 제공함
1104
필터링하기 위한 속성 및 메소드 (예를 들어, Filter, SetFilter)를 제공함
1106
노출된 필드를 제한하는 속성 및 메소드 (예를 들어, Fields, SetFields)를 제공함
1108
Filter, Sort 및 Sections을 위한 파라미터를 열거하는 컬렉션(예를 들어, Parameters)을 제공함
1110
뷰를 리프레쉬하는 속성 (예를 들어, AutoRefresh)을 제공함
1112
페이지 크기를 설정하는 속성 (예를 들어, PageSize)을 제공함
1114
종료

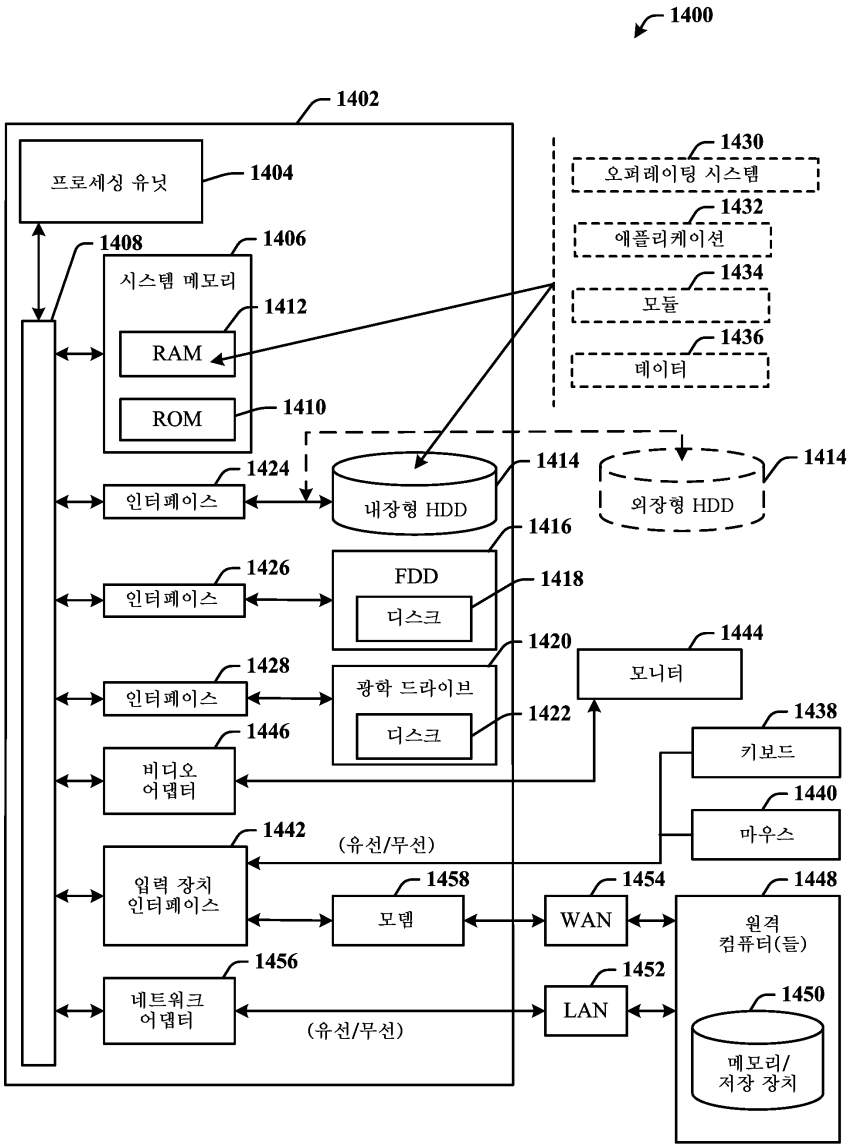
도면12


시작
뷰 속성을 부가하는 클래스
(예를 들어, StorageViewRecord)를 수신함
1200
섹션 헤더 레코드인지를 반환하는 속성
(예를 들어, IsSectionRecord)을 제공함
1202
섹션 정보를 위한 속성
(예를 들어, SectionLevel, SectionName,
IsSectionExpanded)을 제공함
1204
현재 레코드에 대한 북마크를 반환하는
속성(예를 들어, Bookmark)을 제공함
1206
레코드 필드의 값을 설정하는 메소드
(예를 들어, SetValueinRecord)를 제공함
1208
종료

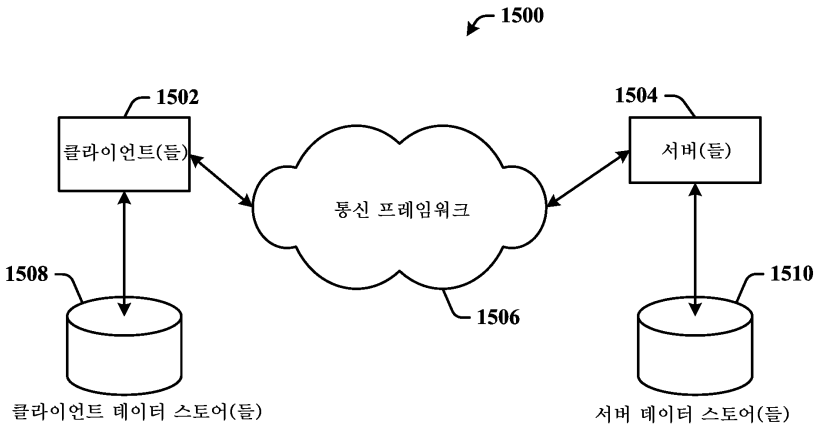
도면13


1300
1304
애플리케이션 및
애플리케이션 프레임워크
1302
공통 데이터 플랫폼(CDP)
API
100
런타임
1308
제약/보안
1310
1306
데이터 스토어

도면14


1400
1402
프로세싱 유닛
1404
1408
1406
시스템 메모리
1412
RAM
ROM
1410
1424
인터페이스
내장형 HDD
1414
1414
외장형 HDD
1416
1426
인터페이스
FDD
디스크
1418
1420
1428
인터페이스
광학 드라이브
디스크
1422
1444
모니터
1446
비디오
어댑터
1438
키보드
1440
마우스
1442
(유선/무선)
입력 장치
인터페이스
1458
모뎀
1454
WAN
1448
원격
컴퓨터(들)
1456
네트워크
어댑터
(유선/무선)
1452
LAN
1450
메모리/
저장 장치
1430
오퍼레이팅 시스템
1432
애플리케이션
1434
모듈
1436
데이터

도면15


1500
1502
클라이언트(들)
통신 프레임워크
1504
서버(들)
1508
1506
1510
클라이언트 데이터 스토어(들)
서버 데이터 스토어(들)