(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl. 7 G06F 17/30	(11) 공개번호 특2002-0034998 (43) 공개일자 2002년05월09일

(21) 출원번호	10-2001-7015068		
(22) 출원일자	2001년11월24일		
번역문 제출일자	2001년11월24일		
(86) 국제출원번호	PCT/US2000/14497	(87) 국제공개번호	WO 2000/72165
(86) 국제출원출원일자	2000년05월24일	(87) 국제공개일자	2000년11월30일

(81) 지정국

국내특허 : 알바니아, 아르메니아, 오스트리아, 오스트레일리아, 아제르바이잔, 보스니아-헤르체고비나, 바베이도스, 불가리아, 브라질, 벨라루스, 캐나다, 스위스, 중국, 쿠바, 체코, 독일, 덴마크, 에스토니아, 스페인, 핀랜드, 영국, 그루지야, 헝가리, 이스라엘, 아이슬란드, 일본, 케냐, 키르기즈, 대한민국, 카자흐스탄, 세인트루시아, 스리랑카, 라이베리아, 레소토, 리투아니아, 룩셈부르크, 라트비아, 몰도바, 마다가스카르, 마케도니아, 몽고, 말라위, 멕시코, 노르웨이, 뉴질랜드, 슬로베니아, 슬로바키아, 타지키스탄, 투르크메니스탄, 터어키, 트리니다드토바고, 우크라이나, 우간다, 우즈베키스탄, 베트남, 폴란드, 포르투칼, 루마니아, 러시아, 수단, 스웨덴, 싱가포르, 가나, 감비아, 인도네시아, 유고슬라비아, 짐바브웨, 시에라리온, 북한, 그레나다, 크로아티아, 인도, 남아프리카, 아랍에미리트,
AP ARIPO특허: 케냐, 레소토, 말라위, 수단, 스와질랜드, 우간다, 가나, 감비아, 짐바브웨, 시에라리온, 탄자니아, 모잠비크,
EA 유라시아특허: 아르메니아, 아제르바이잔, 벨라루스, 키르기즈, 카자흐스탄, 몰도바, 러시아, 타지키스탄, 투르크메니스탄,
EP 유럽특허: 오스트리아, 벨기에, 스위스, 독일, 덴마크, 스페인, 프랑스, 영국, 그리스, 아일랜드, 이탈리아, 룩셈부르크, 모나코, 네덜란드, 포르투칼, 스웨덴, 핀랜드, 사이프러스,
OA OAPI특허: 부르키나파소, 베넹, 중앙아프리카, 콩고, 코트디브와르, 카메룬, 가봉, 기네, 말리, 모리타니, 니제르, 세네갈, 차드, 토고, 기네비쏘,

(30) 우선권주장	09/317,773	1999년05월24일	미국(US)

(71) 출원인

컴퓨터 어소시에이츠 싱크, 인코포레이티드
엘그레시 도론
미국, 뉴욕 11749, 아일랜디아, 원 컴퓨터 어소시에이츠 플라자

(72) 발명자

만-얀트세,에바
미국77479텍사스주슈거랜드아메리칸엘름코트1835
로레,마이클,딘
미국77450텍사스주카트리호카데이드라이브22714
아타웨이,제임스,다니엘
미국77494텍사스주카리카운티다운코트24715

심사청구 : 없음

(54) 단일 집합 프로세스에서 다수의 데이터 마트를 분포시키는방법 및 장치

요약

단일 집합(aggregation) 프로세스로 데이터 베이스에서 유지되는 트랜잭션 데이터 세트(set)로부터 단일 동작으로 다수의 데이터 마트(mart)를 분포(populate)시키는 방법이 개시되며, 집합 값들은 단 한번 계산되며, 어떤 출력 데이터 마트가 집합 값을 요구하였는지에 대한 결정이 이루어지며, 그 집합 값들은 적절한 데이터 마트로 출력된다. 그 출력 집합 레코드에 관련된 디멘젼 데이터 또한 적절한 데이터 마트로 출력된다.

대표도

도 1

색인어

단일 집합 프로세스, 데이터 마트 분배, 크로스 프로덕트, 디멘젼 프로세싱

명세서

기술분야

본 특허 출원은 데이터 마트를 분포(populate)시키는 방법에 관한 것이다.

배경기술

데이터 마트는 경영자로 하여금 자신의 비지니스에 대한 전략적 결정을 내리는데 도움을 주도록 고안된 데이터베이스 또는 데이터베이스의 집합이다. 데이터웨어하우스(data warehouse)는 전사적으로 데이터베이스를 결합하는 반면, 데이터 마트는 일반적으로 보다 작으며, 특정 주제 또는 부서에 집중된다. 종종, 데이터 마트는 보다 큰 데이터 웨어하우스의 서브세트이다. 라몬 발퀸(Ramon Barquin) 및 허브 에델스타인(Herb Edelstein)이 지은 프렌티스 홀 PTR사의 데이터 웨어하우스의 플래닝 및 디자인(ISBN 0-13-255746-0)은 데이터 마트의 정의 및 사용법을 기술하고 있다. 랄프 킴볼(Ralph Kimball)이 지은 존 윌리 앤 선스사의 데이터 웨어하우스 툴키트(Data Warehouse Toolkit)(ISBN 0-471-15337-0)는 데이터 웨어하우징의 배경 및 개념에 대해 좋은 설명을 제공한다.

성공적인 데이터 마트를 제작하는데에 있어 제1 단계들 중 하나는 비지니스 구조내에서 서로 다른 디멘젼(dimension) 및 팩트(fact) 세트(set)를 올바르게 식별하는 것이다. 이것은 종종 디멘젼 모델링으로 알려져 있다. 각각의 디멘젼은 다른 디멘젼으로부터의 엔티티(entity)에 독립적인 팩트 세트에 참여하는 고유 엔티티의 집합(collection)을 나타낸다. 일반적으로 팩트 세트는 각각의 디멘젼으로부터의 엔티티의 조합에 의해 각각의 트랜잭션(transaction)(또는 레코드)이 식별되는 트랜잭션 데이터를 포함한다. 도 1은 스타 스키마(star schema)가 디멘젼 모델링 프로세스의 출력인 슈퍼마켓 비지니스에 대한 스타 스키마를 도시한다.

각각의 디멘젼은, 각각의 레코드(record)가 각각의 엔티티 및 속성(attribute)의 리스트를 고유하게 식별하는 키(또는 복합키)를 포함하여 대응하는 엔티티(또는 키)를 기술하거나 또는 인가하는 테이블이다. 팩트 테이블내의 각각의 팩트 레코드는 트랜잭션 데이터를 나타내는 측정치(measures)의 리스트 및 각각의 디멘젼에 결합하는 것을 허용하는 외부키(foreign key)를 포함한다. 일반적으로 디멘젼의 크기가 팩트 테이블의 크기보다 훨씬 작기 때문에 디멘젼 테이블은 일반적으로 더 이상 표준화(normalize)되지 않는다; 따라서, 표준화에 의해 절약되는 공간은 중요하지 않다. 또한, 쿼리(query) 런타임에서 OLAP 쿼리 툴이 표준화된 디멘젼 테이블과 결합하는 것은 시간적으로 효율적이지 않다.

이론적으로, OLAP 툴은 상기 스타 스키마 레이아웃에서 트랜잭션 데이터를 포함하는 데이터 마트에 대해 직접 질문할 수 있다. 그러나, 하이 레벨 쿼리, 예를 들면, 각 스테이트(state)에 대해 특정 브랜드 제품의 월별 판매량을 얻기 위한 쿼리에서 빠른 응답 시간을 허용하기 위해, 데이터 웨어하우스 내의 데이터의 사전 집합(pre - aggregation)이 필요하다.

집합의 목적을 위해 각각의 디멘젼에 데이터의 레벨이 지정된다. 각각의 레벨은 조건에 기초하여 디멘젼 엔트리(entry)의 그룹핑을 정의한다. 예를 들면, 도 1의 스토어 디멘젼(store dimension)에서, 적어도 하나의 스토어를 가지는 각각의 스테이트에 대해 하나의 집합된 디멘젼 레코드를 포함하는 스테이트 레벨(state level)이 특정될 수 있다. 즉, 특정 스테이트에 대한 각각의 집합된 디멘젼 레코드는 상기 스테이트에 있는 모든 스토어들로부터의 데이터 집합을 나타낸다. 유사하게, 각각의 엔트리가 특정 시티의 모든 스토어의 집합을 나타내는 집합된 디멘젼 레코드의 생성을 허용하기 위해 스토어 디멘젼에 시티 레벨(city level)을 지정할 수 있다.

입력(또는 상세(detail)) 레벨로 표시되는 레벨은 가장 낮은 레벨이고 입력 디멘젼 데이터로서 동일한 개수의 레코드를 포함한다. 집합 목적으로 사용자에 의해 지정된 레벨은 집합체 레벨(aggregate level)로 나타낸다. 각각의 집합체 레벨은 레벨 조건에 따라서 서로 다른 개수의 레코드를 포함할 것이다. 예를 들면, 스테이트 레벨은 아마도 시티 레벨보다 적은 레코드를 포함할 것이다. 각각의 (입력 및 집합된) 레벨은 레벨 코드에 의해 고유하게 식별된다. 레벨 코드는 효율성을 위해 일반적으로 정수로 나타낸다.

출력 팩트 데이터에서 요구되는 집합은 각각의 디멘젼으로부터의 레벨들의 조합으로 지정된다. 집합을 지정하기 위해 사용되는 레벨의 조합은 레벨의 크로스 프로덕트(cross product)로도 나타낸다. 상기 스타 스키마 예에서 월별 브랜드별 스테이트 쿼리를 실행하기 위해서는, 대응하는 레벨이 각각의 디멘젼에 정의될 필요가 있고, 3개의 특정 레벨의 크로스 프로덕트에 기초하여 집합이 트랜잭션 데이터를 요청한다는 것을 지정할 필요가 있다. 사용자들은 집합을 원하는 것에 대해 크로스 프로덕트의 리스트를 지정할 수 있다. 입력 레벨의 크로스 프로덕트는 입력 또는 상세 팩트 데이터를 나타낸다. 집합될 각각의 입력 팩트 측정치와 연관된 하나 이상의 집합 표현이 있을 수 있다. 소정의 공통 집합 표현들은 최대, 최소, 평균, 합, 카운트, 가중 평균, 및 집합된 데이터에 대해 일정한 값을 갖는 필(fill)을 포함한다.

모든 디멘젼 엔트리들을 포함하는 단일 멤버를 갖는 각각의 디멘젼내에 " 모든 값(all values)" 레벨이 일반적으로 제공된다. 대응하는 디멘젼이 크로스 프로덕트에 포함되지 않을 경우 이러한 레벨이 크로스 프로덕트에 사용되어, 모든 디멘젼으로부터의 레벨을 각각 포함하는 크로스 프로덕트에 대한 단일 포맷이 정의된다. " 모든 엔트리" 레벨을 크로스 프로덕트에 추가하는 것은 집합의 결과에 아무런 영향을 주지 못하고 크로스 프로덕트에서 더미 엔트리의 역할을 한다. 예를 들면, 도 1의 스타 스키마가 사용되고 시티별 월별 크로스 프로덕트가 요구될 경우, 프로덕트 " 모든 엔트리" 레벨이 크로스 프로덕트내에 포함될 수 있다. 실제로, 모든 프로덕트에 대한 데이터는 시티별 월별 집합체 레코드로 집합된다.

집합 프로세스 이후에, 데이터 마트가 생성된다. 각각의 데이터 마트는 디멘젼 및 팩트 테이블의 세트를 포함한다. 데이터 마트의 각각의 디멘젼은 소스 스타 스키마내의 하나의 디멘젼과 대응한다. 데이터 마트의 팩트 테이블은 지정된 크로스 프로덕트의 리스트에 대한 집합된 데이터를 포함한다.

본 양수인에 의해 제공된 이전의 프로덕트는 집합된 디멘젼 레코드와 상세한 디멘젼 레코드사이의 포함 관계(containment relationship), 팩트 데이터의 집합, 및 집합된 디멘젼과 팩트 데이터의 분포를 인식하는 컨설팅 서비스이다.

상술한 프로덕트는 출력 분포에 기본적인 메카니즘을 가진다. 상세하고 집합된 디멘젼 레코드의 서로 다른 레벨을 서로 다른 출력 디멘젼 파일/테이블에 분배하고 서로 다른 집합체 레벨 크로스 프로덕트를 서로 다른 출력 팩트 파일/테이블에 분배할 수 있다. 그러나, 다수의 데이터 마트를 분포시키는 개념을 실제로 지원하지 않는다.

단지 하나의 타겟(출력) 스타 스키마 구성이 허용된다. 이것은 서로 다른 스타 스키마 구성들인 데이터 마트를 가지는 개념을 지원하지 않는다. 따라서, 발생된, 서로 다른 출력 디멘젼 또는 팩트 테이블은 동일한 출력 타겟 스타 스키마 구성에 해당된다.

유사하게, 상술한 프로덕트는 서로 다른 타겟 팩트 테이블에서 서로 다른 집합체의 발생을 허용하지 않는다. 모든 팩트 테이블은 동일한 형태의 집합체를 가져야 한다. 그러나, 사용자들중의 하나의 그룹에 관심있는 집합체는 사용자들의 다른 그룹에는 관심이 없을 수 있다. 따라서, 서로 다른 집합체를 허용한다는 것은 필수적이지 않는 데이터를 처리하는데 낭비되는 자원을 절약하고 사용자들에 의해 접근 가능한 데이터의 영역을 제한할 수 있다.

더구나, 디멘젼 테이블은 임의의 특정 팩트 테이블에 연결되어 있지 않다. 프로덕트가 다수의 디멘젼 및 팩트 테이블을 생성하더라도, 올바른 디멘젼 테이블들이 올바른 팩트 테이블에 결합되는 것을 보장하도록 데이터 마트의 개념이 강요되지는 않는다. 또한, 사용자들은 디멘젼 테이블을 지정하여 그 대응되는 팩트 테이블의 크로스 프로덕트와 연관되지 않는 레벨들을 포함할 수 있다. 이것은 디멘젼과 팩트 테이블사이의 참조 무결성(referential integrity)을 체크하는 데이터베이스 시스템에서 에러가 발생할 가능성이 매우 높다.

충분한 유연성과 효율성이 없이, 단일 집합 생성 프로세스에서 다수의 데이터 마트를 생성하는 기술들은 오늘날 많이 존재한다. 다음은 유연성 및 성능이 요구되는 이유들이다:

1) 다른 사용자 그룹에 대한 불필요한 데이터를 포함하고 있기 때문에 모든 사용자 그룹이 필요로 하는 모든 집합체들을 가지는 단일 데이터 마트를 생성하는 것은 바람직하지 않다. 더우기, 생성된 집합체의 세트는 너무 커서 서로 다른 물리적 위치에서 서로 다른 사용자 그룹들사이에서 공유되거나 또는 복제될 수 없다.

2) 작고 전문화된 데이터 마트는 타겟이 된 사용자 그룹에 대해 보다 작은 양의 데이터 및 보다 덜 복잡한 데이터의 뷰(view)를 포함한다. 전문화된 데이터 마트에 대한 쿼리도 집합된 데이터의 보다 작거나 또는 일정 범위의 세트때문에 보다 효율적이다.

3) 서로 다른 사용자 그룹들이 디멘젼 속성 또는 팩트 집합체의 서로 다른 세트에 관심이 있기 때문에, 데이터 마트는 종종 서로 다른 타겟 스타 스키마 구성 및 집합체를 갖도록 요구된다. 더우기, 서로 다른 사용자 그룹들은 종종 서로 다른 방식으로 데이터를 주목한다.

4) 데이터 분할(data partitioning)은 저장 자원을 절약하는데 매우 필수적이다. 동일한 디멘젼 테이블을 공유하는 서로 다른 팩트 테이블을 갖는 능력이 일반적으로 필요하다. 일례는 2개의 데이터 마트: 실질적인 판매에 대한 하나의 데이터 마트와 예측된 판매에 대한 하나의 데이터 마트를 가진다. 양쪽 데이터 마트는 동일한 디멘젼 테이블을 공유할 수 있다: 그러나, 각각의 데이터 마트는 팩트 데이터의 자신의 뷰를 포함하는 자신의 팩트 테이블을 가질 필요가 있을 것이다.

5) 전문화된 데이터 마트를 생성하기 위하여 집합 생성을 여러번 실행하는 것은 너무 비용이 많이 든다. 여러번에 걸쳐 서로 다른 집합 버킷(bucket)내의 입력 팩트의 포함(containment)을 분석하고 입력 팩트 파일을 판독하는 것을 포함하기 때문에 집합을 여러번 실행하는 것에 대한 오버헤드는 상당히 높다.

데이터 마트는 효과적인 하이 레벨 쿼리에 대한 OLAP 툴에 사용되는 공표된 데이터를 포함한다. 데이터 마트내의 집합된 데이터는 효과적인 비지니스 경영을 위한 의사결정 지원용으로 주로 사용된다. 그러나, 서로 다른 사용자 그룹들은 트랜잭션 데이터의 동일한 세트로부터의 집합된 데이터의 서로 다른 세트들을 요구할 수 있다. 예를 들면, 서쪽 영역

마케팅 그룹은 서쪽 영역에 대한 데이터에만 관심이 있을 것이고; 따라서, 국내 판매 데이터의 한 세트로부터 서로 다른 지리학적 서브세트에 대한 서로 다른 데이터 마트를 만드는 것이 효과적이다. 반대로, 사용자들은 또한 전국적인 판매 숫자들의 합과 같이 모든 영역들사이에서 관심있는 데이터를 요구할 수도 있다. 이러한 경우에, 데이터 마트는 소정의 오버랩된 집합 데이터를 포함할 수도 있다.

발명의 상세한 설명

상술한 필요성들을 효과적으로 해결하기 위하여, 단일 집합 프로세스에서 트랜잭션 데이터의 한 세트로부터 다수의 데이터 마트를 분포시키는 방법이 제공된다. 즉, 이러한 방법은 동일한 동작 데이터 소스로부터 집합된 데이터를 분할한다. 각각의 데이터 마트 또는 분할에 대해 하나를 여러번 실행하는 것보다 다수의 데이터 마트를 하나의 집합 실행에 분포시키는 것이 자원이 훨씬 더 효율적이다. 다수의 실행은 입력 팩트 데이터를 여러번 판독하고 각각의 실행에서 유사한(동일하지는 않다면) 집합 버킷을 생성하는 것이 요구된다. 다수의 데이터 마트 분포(또는 데이터 분할)의 필요성 및 이점들은 다음 섹션에서 더 설명될 것이다.

시스템은 3개의 주요 프로세싱 단계: 디멘젼 프로세싱, 집합 프로세싱, 및 데이터 마트 분배를 포함한다. 본 발명은 다수의 데이터 마트를 분포시키는 것에 주로 관련되기 때문에, 집합 프로세스의 초기 2개의 단계들은 설명된 기술의 실행 이전에 종료되는 것으로 가정한다. 디멘젼 및 집합 프로세싱을 달성하기 위한 기술은 잘 알려져 있고 이용 가능하다. 본 발명의 한 측면에서는, 사용자가 특정된 데이터 마트 정보, 팩트 집합체의 리스트, 및 각 디멘젼에 대한 디멘젼 데이터의 하나의 리스트가 주어진 다수의 데이터 마트의 분배에 대한 기술이 제공된다.

도면의 간단한 설명

본 발명의 특정 실시예는 첨부된 도면을 참조하여 이하에 설명된다.

도 1은 간단한 수퍼마켓 비지니스의 한 예시적인 스타 스키마를 도시한 도면이다.

도 2는 정의 포함(containment) 관계를 도시한 도면으로서, 화살표 라인은 포함 관계를 나타내고, 각 화살표 라인 다음에 붙은 값은 경우(instances)의 수를 나타낸다.

도 3은 데이터 마트 구조를 도시한 도면으로서, 점선으로 표시된 화살표는 디멘션, 데이터 마트 및 측정치가 데이터 구조 내에 어떻게 인덱스되는 지를 나타내는 도면이다.

도 4는 출력 타겟 팩트 테이블에 대한 팩트의 집합체 레코드 맵핑의 한 예를 도시한 도면으로서, 이 예는 단지 하나의 타겟 팩트 테이블에 대한 3개의 디멘션 및 5개의 측정치 맵핑을 나타낸다.

도 5는 다수의 데이터 마트 분포 기술의 전체적인 데이터플로우와 동작을 도시한 도면이다.

실시예

이하, 본 발명의 특정 실시예에 대해 설명하겠다.

다음은 본 발명의 특정 실시예에 대한 프로세스로 들어가는데 필요한 각 구성요소의 설명이다.

사용자는 데이터 마트 분포에 대한 디멘션과 데이터 마트 정보를 지정해야 한다. 이것은 각각 디멘션 정의 및 데이터 마트 정의라 불리운다. 그러므로, 사용자는 디멘션 정의 리스트(각 디멘션마다 하나) 및 데이터 마트 정의 리스트(생성될 각 데이터 마트마다 하나)를 지정해야 한다.

각 디멘션 정의에 있어서, 사용자는 다음과 같은 정보를 지정해야 한다. 이것은 디멘션 및 집합 프로세싱에 필요한 정보를 포함하지 않는다.

1. 디멘션 이름

디멘션을 식별하기 위한 유일한 이름.

2. 레벨 조건 및 레벨 코드

집합에 이용가능한 집합된 레벨의 리스트. 각 레벨은 레벨 코드에 의해 유일하게 식별된다. 레벨 조건은 각 레벨 내의 디멘셔널 레코드의 그룹핑을 결정한다.

3. 디멘션 출력 필터링 플래그(임의 선택)

디멘션 출력 필터링 플래그는 각 데이터 마트 내에서 출력될 디멘션 레코드를 제어한다. 하나의 필터링 옵션은 각 디멘션마다 지정된다. 이 플래그는 설명된 데이터 마트 집단 기술의 요구조건이 아니다. 즉, 그 기술은 이 플래그를 사용하지 않고 실현될 수 있다; 그러나, 이것은 몇몇 시나리오에서 매우 유용하다는 것이 알려졌기 때문에 설명된 구현의 일부분으로서 포함된다. 다음은 4가지 출력 필터링 옵션이다.

a) 출력 없음. 어떠한 디멘션 레코드도 모든 데이터 마트에 대한 디멘션 테이블 내에서 출력되지 않는다. 이것은 사용자가 입력 디멘션 테이블 내에 새로운 레코드를 갖지 않은 경우에 유용하다. 예를 들어, 수퍼마켓 스키마에 있어서, 프로덕트 디멘션 내에 새로운 프로덕트가 없고 사용자가 생성될 새로운 집합체에 대해 새로운 레벨을 지정하지 않았으면, 디멘션 레코드는 이전의 집합체 생성 시에 데이터베이스 내에 이미 존재해야 하기 때문에 디멘션 레코드를 생성할 필요가 없다.

b) 전부 레코드. 특정 데이터 마트의 출력 크로스 프로덕트 내에 있는 레벨 코드를 갖는 소정의 디멘션 레코드는 데이터 마트의 디멘션 테이블로 출력될 수 있다. 예를 들어, 수퍼마켓 스키마에 있어서, 브랜드별 월별 스테이트 집합체가 데이터 마트에 출력되고 이 출력 옵션이 프로덕트 디멘션으로 선택되면, 모든 브랜드 레코드는 데이터 마트용 프로덕트 디멘션 테이블에 출력될 수 있다. 이것은 현재의 팩트 테이블 내에 발생되지 않은 브랜드를 포함할 것이다.

c) 데이터 마트에서의 액티브. 이것은 '전부 레코드' 옵션으로부터의 레코드의 서브세트이다. 디멘션 레코드는 또한 디멘션 테이블에 출력되도록 하기 위해 데이터 마트 내에서 액티브되어야 한다. 이전의 옵션에 주어진 예와 같이, 브랜드는 또한 디멘션 레코드가 디멘션 테이블에 출력되도록 하기 위해 현재의 집합 프로세스(입력 팩트 테이블)에서 액티브되어야 한다.

d) 소정의 데이터 마트에서의 에버 액티브. 이것은 '전부 레코드' 옵션으로부터의 레코드의 서브세트이다. 디멘션 레코드는 또한 디멘션 테이블에 출력되도록 하기 위해 데이터 마트의 현재의 또는 임의의 이전의 집합체 내에서 액티브되어야 한다. '전부 레코드' 옵션에서 주어진 예와 같이, 현재의 또는 임의의 이전의 집합체 내에서 액티브한 브랜드 레코드는 디멘션 테이블에 출력될 수 있다. 그 데이터 마트에 대한 출력 팩트 데이터에 전혀 관련되지 않은 브랜드인 경우, 그들은 디멘션 테이블에 출력될 수 없다.

각 데이터 마트 정의마다, 사용자는 다음과 같은 정보를 지정해야 한다.

4. 데이터 마트 이름

데이터 마트를 식별하기 위한 유일한 이름.

5. 레벨 크로스 프로덕트 리스트

이것은 현재의 데이터 마트에서 계산된 집합체를 가져야 하는 레벨의 모든 조합의 리스트이다. 현재의 데이터 마트가 입력 데이터 팩트의 출력을 요구하면, 레벨 조합의 리스트는 입력 레벨 크로스 프로덕트의 엔트리를 포함할 것이다. 가장 간단한 경우(하나의 디멘션)에, 이것은 단순히 집합체를 생성하기 위한 레벨의 리스트이다. 각 레벨 크로스 프로덕트는 각 디멘션에 대응하는 하나의 레벨 코드 엔트리를 갖는다. 크로스 프로덕트 내의 레벨 코드의 차수는 디멘션 정의 리스트 내에 지정된 디멘션의 차수에 대응해야 한다(이것은 요구사항은 아니지만, 맞다면 일을 더욱 단순화시킨다).

6. 디멘션 테이블 정의 리스트

디멘션 테이블 정의 리스트는 각 디멘션마다 하나의 디멘션 테이블 정의 엔트리를 포함한다. 각 디멘션 테이블 정의는 칼럼 위치와 같은 칼럼 정보, 데이터 타입, 디폴트 값 및 그밖의 데이터베이스 관련 정보를 포함한다. 게다가, 각 칼럼마다, 사용자는 키이 칼럼인지 속성 칼럼인지를 식별할 필요가 있다. 속성 칼럼인 경우, 각 칼럼은 하나의 입력 속성 칼럼에 대응한다.

하나의 디멘션에 대해 새로운 테이블 정의를 지정하는 대신에, 사용자는 다른 데이터 마트 내의 동일한 디멘션에 대해 다른 테이블 정의와 동일하게 디멘션 테이블 정의를 지정해도 된다. 이 경우에, 2개의 데이터 마트는 동일한 타겟 디멘션 테이블을 공유할 수 있다. 2개의 데이터 마트로부터의 디멘션에 대한 출력 기록은 동일한 타겟 디멘션 테이블 내로 병합될 것이다. 실제로, 하나 이상의 데이터 마트는 이들의 디멘션 데이터를 동일한 타겟 테이블에 출력할 수 있다.

7. 팩트 테이블 정의(생성될 측정치의 리스트를 포함)

각 데이터 마트 정의는 하나의 팩트 테이블 정의를 포함할 수 있다. 팩트 테이블 정의는 칼럼 위치와 같은 칼럼 정보, 데이터 타입, 디폴트 값, 및 그밖의 다른 데이터베이스 관련 정보를 포함한다. 게다가, 사용자는 또한 이것이 키이 칼럼인지 그리고 측정치 칼럼인지를 확인할 필요가 있다.

각 디멘션마다 대응하는 하나의 키이 칼럼이 있어야 한다. 키이 칼럼의 데이터 타입은 디멘션 테이블 정의에 지정된 키이 칼럼의 데이터 타입과 모순되지 않아야 된다.

각 측정치 칼럼은 측정치 정의에 대응해야 한다. 각 측정치 정의는 집합체 타입에 의존하는 입력 팩트 테이블로부터 집합체 타입과 소정의 측정치 칼럼(들)을 지정한다. 현재의 바람직한 실현에 있어서, 이들 집합체 타입은 다음과 같이 지원된다; 최대, 최소, 합, 평균, 가중 평균(평균될 칼럼 및 " 가중" 칼럼을 수반), 카운트, 및 필(fill)(주어진 값을 집합체 내로 배치). 다른 타입의 집합체는 구현 시에 용이하게 포함될 수 있다.

현재의 데이터 마트에 대해 새로운 팩트 테이블 정의를 지정하는 대신에, 사용자는 다른 데이터 마트로부터 팩트 테이블 정의를 지정해도 된다. 이 경우에, 2개의 데이터 마트는 동일한 타겟 팩트 테이블을 공유할 수 있다. 2개의 데이터 마트로부터의 출력 집합체 레코드는 동일한 타겟 팩트 테이블 내로 병합될 수 있다. 실제로, 하나 이상의 데이터 마트는 이들의 팩트 데이터를 동일한 타겟 테이블에 출력할 수 있다.

팩트 집합체의 리스트는 집합 생성 프로세스의 출력이다. 이것은 입력 및 집합된 레벨의 팩트 데이터를 포함한다. 팩트 집합체의 리스트는 본 발명의 데이터 마트 분산 알고리즘에 대한 입력들 중의 하나로서 소용된다. 각 팩트 집합체 레코드는 각 디멘션을 결합하는 키, 크로스 프로덕트 레벨 코드 조합, 및 집합된 측정치 리스트를 포함한다. 레코드는 다음과 같은 논리 표현을 갖는다.

[표 1]

키이$_1$	...	키이$_N$	레벨$_1$	...	레벨$_N$	측정치$_1$	...	측정치$_M$

각 디멘션마다 하나의 외부 키이가 있어야 한다. 외부 키이는 비교하기가 훨씬 더 빠르지만, 다른 데이터 타입이 선택될 수 있기 때문에 (현재의 양호한 구현화에서처럼) 정수로 표현되는 것이 가장 좋다. 레코드 내의 키이는 데이터 마트 구조 내의 디멘션 정의와 동일한 차수로 리스트되는 것이 이상적이다. 그러나, 이것은 요구사항이 아니다. 이들은 후술되는 바와 같이 원하는 차수로 다시 맵핑될 수 있다.

레벨 코드 조합은 현재의 집합체 레코드가 대응하는 크로스 프로덕트 레벨 조합을 나타낸다. 각 레벨 코드는 각 디멘션으로부터의 집합체 또는 세부 레벨을 나타낸다. 레벨 코드가 데이터 마트 구조에 지정된 디멘션 정의 리스트와 동일한 차수이면 더욱 단순화될 수 있다. 그러나, 이것은 요구사항이 아니다. 레벨 코드 조합에 기초하여, 집합체 레코드가 특정 데이터 마트에 기록될 필요가 있는 지의 여부를 판정한다. 레벨 코드 조합은 또한 모든 디멘션으로부터의 세부 레벨을 포함해도 된다. 이 경우에, 레코드는 세부 레벨 팩트 레코드이고, 측정치는 집합체 타입에 의존하는 입력 팩트 레코드와 동일한 값을 포함할 수도 있다. 몇몇 데이터 마트는 기록될 세부 레벨 집합체를 필요로 할 수도 있다.

측정치는 레코드를 위해 생성된 집합체이다. 측정치 리스트는 모든 데이터 마트에 의해 요구된 모든 측정치의 수퍼세트가 되어야 한다. 측정치는 소정의 순서로 리스트된다. 후술되는 바와 같이, 측정치의 순서는 데이터 마트 집단에 대해 원하는 순서로 다시 맵핑될 수 있다.

팩트 집합체 레코드의 필드가 임의의 사전에 추정된 순서로 리스트되지 않기 때문에, 사용자는 또한 팩트 집합체 레코드 정의를 정의할 필요가 있다. 이 정의는 팩트 기록 필드 정의 리스트를 포함할 것이다. 각 필드 정의는 필드가 키이인지, 레벨 코드인지, 또는 측정치인지를 나타내는 목록을 포함한다. 필드가 키이 또는 레벨 코드이면, 이것의 필드 정의는 디멘션 정의에 대한 레퍼런스를 포함할 것이다. 필드가 측정치이면, 이것의 필드 정의는 측정치 정의를 포함할 것이다. 팩트 레코드 정의는 각 필드가 참조되고 있는 것, 따라서 각 필드를 재맵핑하는 것을 알수 있도록 분산 처리를 허용한다.

각 디멘션에 대해서 하나의 디멘션 레코드 리스트가 존재한다. 이는 입력 레벨과 집합된 레벨 디멘션 레코드들을 모두 포함한다. 각각의 디멘션 레코드는 키이, 레벨 코드, 일련의 속성들(attributes), 및 선택적 에버 액티브 스위치(optional ever active switch)를 포함한다. 디멘션 레코드들의 리스트는 또한 본 발명의 데이터 마트 분포 알고리즘의 입력으로서 작용한다. 다음 테이블은 디멘션 레코드의 논리적 화상을 도시한다.

[표 2]

키이	레벨 코드	Attribute $_1$	...	Attribute $_a$	에버 액티브 스위치

키이 필드는 현재의 디멘션 레코드의 키이를 포함한다. 이는 고속 비교를 위한 본 바람직한 실현에 있어서 정수값이다.

레벨 코드는 현재 레코드의 (상세 또는 집합) 레벨을 나타내는 유일한 레벨 식별자이다. 레벨 코드는 고속 비교를 위한 본 바람직한 실현에 있어서 정수이다.

Attribute $_1$ 내지 Attribute $_a$ 는 레코드의 속성값을 나타낸다. 상세 레코드의 경우, 이들은 입력 디멘션 레코드들과 동일한 값을 포함한다. 집합 레코드들의 경우, 몇몇 속성들은 레코드가 집합화된 속성이기 때문에 공값(empty values)을 가질 수 있다.

예를 들면, 브랜드 레벨 디멘션 레코드는, 각 브랜드 레코드가 많은 개개의 상품명들에 대한 집합이므로 상품명 속성을 포함하지 않아도 된다.

선택적 에버 액티브 스위치는 현재 레코드가 임의의 데이터 마트 또는 임의의 사전 집합 생성 프로세스에서 항상 액티브인 지를 나타내는 부울 값이다. 스위치는 디멘션 출력을 필터링하기 위해서 선택적 디멘션 출력 필터링 플래그와 함께 사용된다. 상술한 바와 같이, 출력 필터링 능력을 다수의 데이터 마트 분포 기술에 대해서는 요청할 필요가 없지만, 몇몇 사용자들에 의해 분배 프로세스의 일부가 되는 것이 유용하다는 것이 확인된다. 에버 액티브 스위치가 사용되지 않는 경우, 디멘션 레코드의 리스트는 불변일 필요가 없을 것이다. 에버 액티브 스위치가 사용되면, '데이터 마트들 내에서의 에버 액티브' 출력 필터링 옵션이 바람직하게 기능할 수 있도록 스위치와 함께 디멘션 레코드가 불변이 될 필요가 있을 것이다. 본 바람직한 실현에 있어서, 디멘션 레코드들을 항상 볼 수 있도록 하기 위해 마스터 데이터베이스가 각 디멘션을 위해 제공된다. 이 플래그는 데이터 마트 분포 동안 북-키핑 활동(book-keeping activity)으로서 유지된다.

디멘션 레코드 내의 필드들은 사전 가정된 순서로 리스트되지 않기 때문에, 사용자는 또한 디멘션 레코드 정의를 생성할 필요가 있다. 이 정의는 디멘션 레코드 필드 정의 리스트를 포함한다. 필드 정의는 필드가 키이, 속성, 또는 에버 액티브 스위치인지를 여부를 나타내는 목록을 포함한다. 이 필드가 속성값이면, 그 필드 정의는 그의 입력 디멘션 칼럼의 참조를 포함한다. 키이 필드 또는 에버 액티브 스위치에 대해서는 어떠한 추가적 정보도 필요하지 않다. 따라서, 디멘션 레코드 정의는 분배 프로세스에게 각 필드를 참조하여 각 필드를 재배치하는지를 알 수 있게 한다.

도 2는 상술된 상이한 테이블 또는 칼럼 정의들 사이의 포함 관계를 도시한다.

본 발명의 데이터 마트 분포 기술은 이전의 분포 기술들과, 각 데이터 마트 내에서 디멘션 테이블을 팩트 테이블과 밀접하게 관련시킨다는 점에서 현저한 차이가 있다. 단일 집합 생성 프로세스에서, 이는 복수의 데이터 마트들이 분포될 수 있게 한다. 각각의 데이터 마트는 상이한 스타 스키마를 나타낼 수 있고, 상이한 디멘션 속성들과 상이한 집합체들을 포함할 수 있다. 더욱이 데이터 분할은 데이터 마트들이 동일한 디멘션 또는 동일한 팩트 테이블을 공유할 수 있도록 허용한다. 따라서, 하나 이상의 데이터 마트로부터 생성된 디멘션 또는 팩트 테이블들은 이들 데이터 마트들로부터의 병합된 데이터를 포함할 것이다.

설명된 본 발명의 특정 실시예의 시스템은 다음의 구성 요소; 데이터 마트 구조, 팩트 라이터(writer), 및 디멘션 라이터 (디멘션당 하나의 작성자)를 포함한다. 이후의 단락들에서는 3개의 메인 시스템 구성 요소들 각각의 설계 및 주요 동작들을 설명한다.

도 3에 도시된 바와 같이 상기 사용자 특정 정보에 기초하여 정보가 내부적으로 분석 및 구조화될 수 있다. 이를 데이터 마트 구조라 한다. 이 구조는 주로 사용자에 의해 특정된 데이터 마트 속성들을 얻기 위해 다른 시스템 구성 요소들을 위한 억세스 방법을 제공한다.

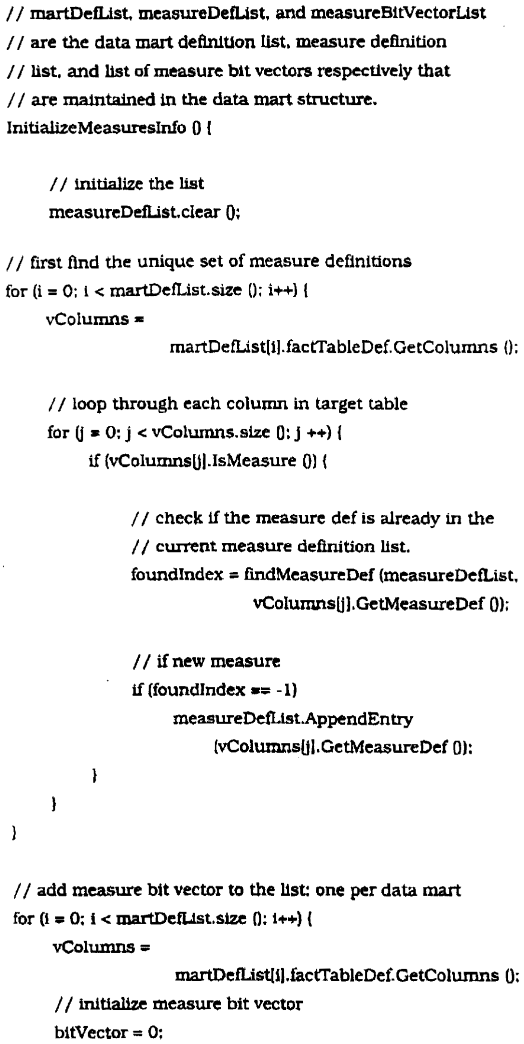
상기 데이터 마트 구조는 사용자 특정 정보를 분석함으로써 구성된다. 데이터 마트 구조 내의 크로스 프로덕트 테이블(cross product table)은 크로스 프로덕트 레벨 코드와 데이터 마트 비트 벡터 쌍의 리스트를 포함한다. 각 크로스 프로덕트 내의 레벨 코드들은 디멘션 정의 리스트에 의해서 특정된 것과 동일한 순서로 리스트된다. 데이터 마트 비트 벡터의 비트들은 데이터 마트 정의 리스트에 의해서 특정된 것과 동일한 순서로 리스트된다.

크로스 프로덕트 테이블은 모든 데이터 마트들에 의해 요청된 별개의 크로스 프로덕트 레벨 코드 세트를 포함한다. 이는 각 데이터 마트 내에 특정된 크로스 프로덕트 레벨 코드들을 병합함으로써 얻어진다. 이 테이블은 탐색 속도를 높일 수 있도록 레벨 코드 조합에 의해서 기억된다. 레벨 코드들은 고속 비교를 위한 본 바람직한 실현에서는 정수값이다. 각 레벨 코드 조합과, 크로스 프로덕트가 관여하고 있는 데이터 마트들을 나타내는 비트 벡터가 관련된다. 크로스 프로덕트 테이블 내의 값들은 각 데이터 마트 내의 레벨 크로스 프로덕트 리스트를 분석하고, 테이블에 각 새로운 레벨 크로스

프로덕트 엔트리를 부가하며, 새로운 또는 발견된 엔트리 내에 현재의 데이터 마트 비트를 1로 설정함으로써 구성된다.

측정치 정의 리스트와 측정치 비트 벡터 리스트는 데이터 마트 정의를 더 분석함으로써 생성된다. 측정치 정의 리스트는 모든 데이터 마트들에 의해 요청된 개개의 측정치 정의 세트를 포함한다. 벡터는 각 데이터 마트의 각 팩트 테이블에 의해 요청된 유일한 측정치들을 찾음으로써 얻어진다. 각 측정치 정의는 집합된 입력 측정치 칼럼(들) 및 그의 집합 유형 (예를 들면, SUM, MAX, MIN 등)에 의해서 유일하게 규정된다. 측정치 비트 벡터들의 리스트는 데이터 마트마다 하나의 엔트리를 포함한다. 각 측정치 비트 벡터는 그의 대응하는 데이터 마트를 위해 요청되는 측정치들을 나타낸다. 2개의 리스트들은 각 데이터 마트로부터의 팩트 테이블의 각 특정치 정의를 통해서 루프화함으로써 초기화된다. 요청된 주요 단계들은

다음의 의사 코드로 표현된다.



```
// martDefList, measureDefList, and measureBitVectorList
// are the data mart definition list, measure definition
// list, and list of measure bit vectors respectively that
// are maintained in the data mart structure.
InitializeMeasuresInfo () {

      // initialize the list
      measureDefList.clear ();

// first find the unique set of measure definitions
for (i = 0; i < martDefList.size (); i++) {
     vColumns =
                    martDefList[i].factTableDef.GetColumns ();

      // loop through each column in target table
      for (j = 0; j < vColumns.size (); j ++) {
           if (vColumns[j].IsMeasure ()) {

                // check if the measure def is already in the
                // current measure definition list.
                foundIndex = findMeasureDef (measureDefList,
                               vColumns[j].GetMeasureDef ());

                // if new measure
                if (foundIndex == -1)
                     measureDefList.AppendEntry
                          (vColumns[j].GetMeasureDef ());
           }
      }
}

// add measure bit vector to the list: one per data mart
for (i = 0; i < martDefList.size (); i++) {
     vColumns =
                    martDefList[i].factTableDef.GetColumns ();
      // initialize measure bit vector
      bitVector = 0;
```

```
            // loop through each column in target table
            for (j = 0; j < vColumns.size (); j ++) {
                  if (vColumns[j].IsMeasure ()) {

                        // find the measure definition from the list;
                        // the measure definition should be found
                        foundIndex = findMeasureDef (measureDefList.
                                       vColumns[j].GetMeasureDef ());
                        measureBit = 1 << foundIndex;
                        bitVector = bitVector | measureBit;
                  }
            }

            // add the bit vector of required measures to the
            // list
            measureBitVectorList.AppendEntry (bitVector);
      }
}
```

일단 데이터 마트 구조가 구성되면, 이는 다른 시스템 구성 소자들에 대해 사용자 특정 파라미터들을 묻는데 유용하다. 다음의 리스트는 데이터 분배 기술을 위해 요청되는 모든 억세스 방법들을 포함한다.

1)모든 측정치 정의를 얻음

vMeasuresDef=GetMeasureDefinitions()const

측정치 정의 리스트에 대한 일정한 레퍼런스를 얻는다. 벡터는 모든 데이터 마트에 의해 요청되는 모든 개개의 측정치들을 포함한다. 각 측정치 정의는 집합 유형의 정보 및 그의 입력 측정치 칼럼(들)을 포함한다.

numMeasures=GetNumMeasures()const

데이터 마트 구조 내의 측정치들의 전체 개수를 얻는다. 이는 측정치 정의 리스트의 사이즈와 동일하다.

2)측정치 정의 인덱스를 얻음

index=GetMeasureIndex(const measureDefinition& def)const

데이터 마트 구조의 측정치 정의 리스트 내에서 지정된 측정치 정의를 찾는다. 리스트 내에서 발견된 측정치들의 0-베이스드 인덱스를 복귀한다. 지정된 측정치 정의가 발견되지 않으면 이 방법은 -1로 복귀한다.

3)모든 디멘션 정의들을 얻음

vDimensionsDef=GetDimensionsDefinitions()const

디멘션 정의 리스트에 대한 일정한 레퍼런스를 얻는다. 벡터는 모든 디멘션 정의들을 포함한다. 각 디멘션 정의는 디멘션 출력 필터링 플래그와 레벨 코드 및 조건들을 포함한다.

numDimensions=GetNumDimensions()const

데이터 마트 구조 내의 디멘션들의 전체 개수를 얻는다. 이는 디멘션 정의 리스트의 사이즈와 동일하다.

3)디멘션 정의 인덱스를 얻음

index=GetDimensionIndex(const dimensionDef& def)const

데이터 마트 구조의 디멘션 정의 리스트 내에서 특정 디멘션 정의를 찾는다. 리스트 내에서 발견된 디멘션의 0- 베이스드 인덱스를 복귀한다. 특정 디멘션 정의가 발견되지 않으면 이 방법은 -1로 복귀한다.

4)모든 데이터 마트 정의를 얻음

vDataMartsDef=GetDataMartDefinitions()const

데이터 마트 정의 리스트에 대한 일정한 레퍼런스를 얻는다. 벡터는 모든 데이터 마트들의 정의를 포함한다. 각 데이터 마트 정의는 섹션 1.1.4.내에 기재된 바와 같이 디멘션 및 팩트 테이블 정의들을 포함한다.

5)크로스 프로덕트를 위한 액티브 데이터 마트들을 얻음

vMartBits=GetActiveMarts (const crossProductLevelCode& xprod)const

크로스 프로덕트 레벨 조합이 주어지면, 데이터 마트들이 팩트 출력에서 지정된 크로스 프로덕트를 생성할 필요가 있는지를 결정한다. 기본적으로 크로스 프로덕트 테이블을 검색하여 특정 크로스 프로덕트를 찾는다. 비트 벡터들의 복귀는 데이터 마트 정의 리스트와 동일한 방식으로 인덱스된다. 1로의 비트 설정은 대응하는 데이터 마트가 특정 크로스 프로덕트를 생성할/출력할 필요가 있는지를 나타낸다.

6)데이터 마트를 위한 액티브 측정치들을 얻음

vMeasBits=GetActiveMeasues(const dataMartDefinition& rMart)const

vMeasBits=GetActiveMeasures(int iMart)const

데이터 마트 정의 또는 데이터 마트 인덱스가 주어지면, 지정된 데이터 마트에 대한 측정치 비트 벡터를 얻는다. 비트 벡터를 반환하는 것은 측정치 정의 리스트와 동일한 방식으로 인덱스된다. 상응하는 측정치를 지시하는 1로 설정된 비트는 데이터 마트에 요구된다.

7)레벨 코드에 대한 액티브 데이터 마트의 취득

vMartBits=GetActiveMarts(const dimensionDefinition& rDim, const levelCode& code)const

vMartBits=GetActiveMarts(int iDim, const levelCode& code)const

데이터 마트를 취득하는 것은 지정된 디멘젼에서 지정된 레벨의 디멘젼 레코드를 출력하는데 필요하다. 이 함수는 기본적으로 크로스 프로덕트 테이블을 통해 루프(loop)하고, 각각의 크로스 프로덕트 조합에서 지정된 디멘젼의 지정된 레벨 코드를 매치시킨다. 그리고, 각각의 발견된 엔트리로부터 데이터 마트 비트 벡터를 비트와이즈(bitwise) ORs한다. 반환된 비트 벡터는 데이터 마트 정의 리스트와 동일한 방법으로 인덱스된다. 상응하는 데이터 마트를 지시하는 1로 설정된 비트는 입력 레벨 크로스 프로덕트를 생성/출력할 필요가 있다.

8)디멘젼 및 데이터 마트가 주어진 액티브 레벨 코드 취득

vLevelCodes = GetActiveLevels (const dimensionDefinition& rDim, const

dataMartDefinition& rMart) const

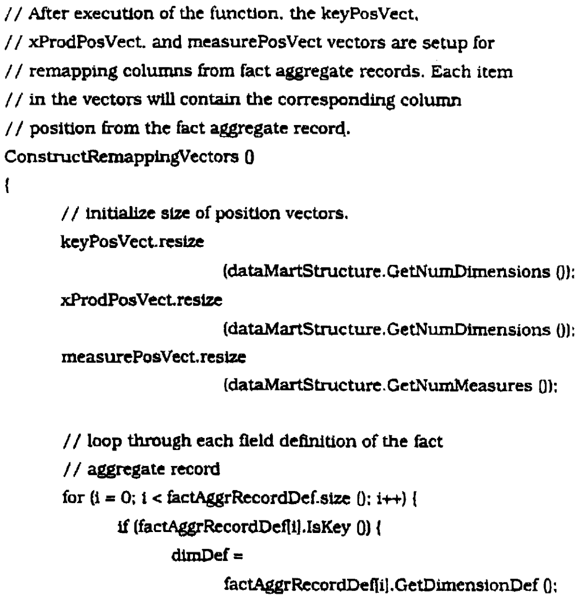
vLevelCodes = GetActiveLevels(int iDim, int iMart) const

디멘젼 및 데이터 마트 정의 또는 이에 상응하는 인덱스가 주어지면, 이 방법들은 지정된 디멘젼으로부터 개별 레벨 코드의 리스트를 얻고, 이 레벨들은 지정된 데이터 마트의 크로스 프로덕트의 리스트에 있다. 특정한 데이터 마트에 대해서, 디멘젼 레코드가 타겟 테이블에 출력되었는지는 그 레벨이 팩트 테이블과 디멘젼 출력 필터링 플래그 값의 어떤 크로스 프로덕트 조합에 관련되는지에 따라서 결정된다.

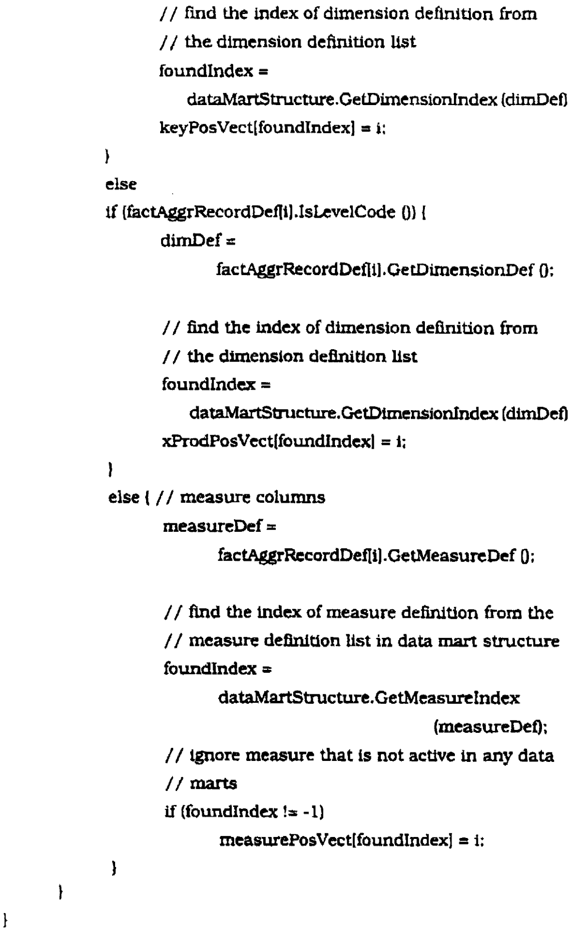
" 팩트 라이터(writer)" 는 모든 데이터 마트에 대한 팩트 테이블을 생성해야 하는 함수적 구성 성분이다. 팩트 라이터는 팩트 집합체의 리스트 및 레코드 정의를 입력으로 하고, 다른 사용자 지정 파라미터에 대한 데이터 마트 구조를 조회하고, 각각의 데이터 마트에 대한 팩트 테이블을 출력한다. 다수의 데이터 마트가 동일한 팩트 테이블에 그 레코드를 병합하도록 지정된다면, 팩트 테이블은 2개 이상의 데이터 마트와 상응할 수 있다.

상술된 바와 같이, 각각의 팩트 집합체 레코드는 키이, 레벨 코드, 및 측정치들의 리스트를 포함한다. 그러나, 키이, 레벨 코드, 및 측정치들은 데이터 마트 구조내의 디멘젼 정의 리스트 및 측정치 정의 리스트와 동일한 차수에 있도록 요구되지는 않는다. 이 차수들은 팩트 집합체 레코드 정의내에 지정된다. 따라서, " 팩트 라이터" 는 데이터 마트 구조내의 디멘젼 및 측정치 차수에 기초해서 집합체 레코드로부터 칼럼을 재배치하기 위해서 맵핑(mapping) 방식을 셋업할 필요가 있을 것이다.

내부적으로, 팩트 라이터는 키이 위치 벡터, 레벨 코드 위치 벡터, 및 측정치 위치 벡터를 셋업할 필요가 있을 것이다. 키이 및 레벨 코드 위치 벡터의 크기는 디멘젼의 개수와 동일하다. 측정치 위치 벡터의 크기는 데이터 마트 구조내의 측정치 정의 리스트의 크기와 동일하다. 이 벡터들은 기본적으로 팩트 집합체 레코드로부터의 키이, 레벨 코드, 및 측정치의 칼럼 위치를 포함한다. 이 벡터들은 팩트 라이터가 데이터 마트 구조내에 지정된 것과 동일한 디멘젼 또는 측정치 차수에서 집합체 레코드를 프로세스할 수 있도록, 칼럼을 재매핑(re-mapping)하는데 사용된다. 다음의 의사-코드는 서로 다른 위치 벡터가 어떻게 셋업되는지에 대한 주요 단계를 기술한다. 여기에 기술된 함수에 대한 입력은 데이터 마트 구조이다.

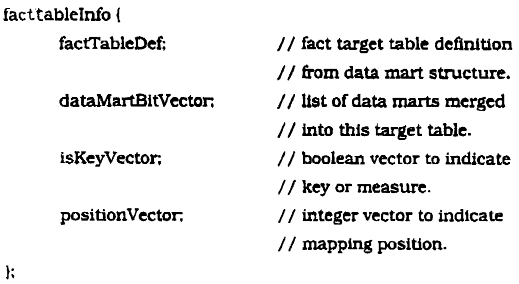


```
// After execution of the function. the keyPosVect.
// xProdPosVect. and measurePosVect vectors are setup for
// remapping columns from fact aggregate records. Each item
// in the vectors will contain the corresponding column
// position from the fact aggregate record.
ConstructRemappingVectors ()
{
        // initialize size of position vectors.
        keyPosVect.resize
                        (dataMartStructure.GetNumDimensions ());
        xProdPosVect.resize
                        (dataMartStructure.GetNumDimensions ());
        measurePosVect.resize
                        (dataMartStructure.GetNumMeasures ());

        // loop through each field definition of the fact
        // aggregate record
        for (i = 0; i < factAggrRecordDef.size (); i++) {
                if (factAggrRecordDef[i].IsKey ()) {
                        dimDef =
                                factAggrRecordDef[i].GetDimensionDef ();
```



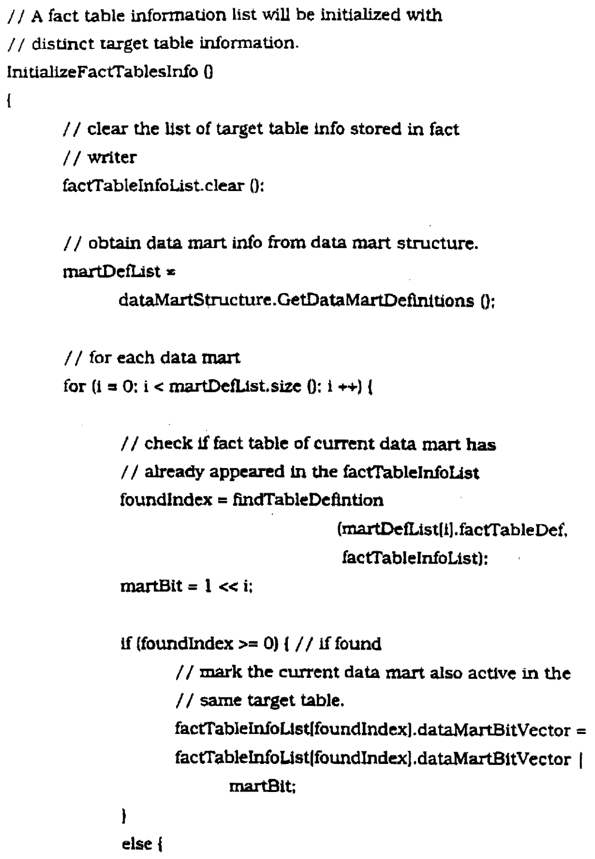
```
                // find the index of dimension definition from
                // the dimension definition list
                foundIndex =
                    dataMartStructure.GetDimensionIndex (dimDef);
                keyPosVect[foundIndex] = i;
        }
        else
        if (factAggrRecordDef[i].IsLevelCode ()) {
                dimDef =
                        factAggrRecordDef[i].GetDimensionDef ();

                // find the index of dimension definition from
                // the dimension definition list
                foundIndex =
                    dataMartStructure.GetDimensionIndex (dimDef);
                xProdPosVect[foundIndex] = i;
        }
        else { // measure columns
                measureDef =
                        factAggrRecordDef[i].GetMeasureDef ();

                // find the index of measure definition from the
                // measure definition list in data mart structure
                foundIndex =
                        dataMartStructure.GetMeasureIndex
                                                        (measureDef);
                // ignore measure that is not active in any data
                // marts
                if (foundIndex != -1)
                        measurePosVect[foundIndex] = i;
        }
    }
}
```

" 팩트 라이터" 는 또한 각각의 데이터 마트 정의에서 지정된 팩트 테이블 정의를 분석해야 할 필요가 있다. 모든 데이터 마트들에 의해 요구되는 개별 타겟 테이블 정의 리스트를 유지해야한다. 각각의 타겟 테이블 정의와 연관된 것은 키이 및 측정치를 타겟 테이블에 의해 바람직한 위치로 재배치하기 위한 위치 벡터, 데이터 마트 비트 벡터, 불린(boolean) 벡터이다. 각각의 개별 팩트 타겟 테이블에 필요한 정보는 다음의 의사-코드에 제시된 구조에서 리스트된다.



```
facttableInfo {
        factTableDef;           // fact target table definition
                                // from data mart structure.
        dataMartBitVector;      // list of data marts merged
                                // into this target table.
        isKeyVector;            // boolean vector to indicate
                                // key or measure.
        positionVector;         // integer vector to indicate
                                // mapping position.
};
```

다음의 의사 - 코드는 상술된 정보를 어떻게 구성하는지를 나타낸다.



```
// A fact table information list will be initialized with
// distinct target table information.
InitializeFactTablesInfo ()
{
        // clear the list of target table info stored in fact
        // writer
        factTableInfoList.clear ();

        // obtain data mart info from data mart structure.
        martDefList =
                dataMartStructure.GetDataMartDefinitions ();

        // for each data mart
        for (i = 0; i < martDefList.size (); i ++) {

                // check if fact table of current data mart has
                // already appeared in the factTableInfoList
                foundIndex = findTableDefintion
                                                (martDefList[i].factTableDef,
                                                 factTableInfoList);
                martBit = 1 << i;

                if (foundIndex >= 0) { // if found
                        // mark the current data mart also active in the
                        // same target table.
                        factTableInfoList[foundIndex].dataMartBitVector =
                        factTableInfoList[foundIndex].dataMartBitVector |
                                martBit;
                }
                else {
```

```
            // construct a new target table entry
            factTableInfo.factTableDef =

 martDefList[i].factTableDef;
            factTableInfo.dataMartBitVector = martBit;

            // get the list of column definitions from
            // the table
            vColumns =
                martDefList[i].factTableDef.GetColumnDefs ();

            // set the size of key vector to no. of columns
            factTableInfo.isKeyVector.resize
                                    (vColumns.size ());
            factTableInfo.positionVector.resize
                                    (vColumns.size ());

            // loop through each column in target table
            for (j = 0; j < vColumns.size (); j ++) {
                    if (vColumns[j].IsKey ()) {
                        factTableInfo.isKeyVector[j] = true;

                        // find index of dimDef in data mart
                        // structure
                        dimIndex =
                            dataMartStructure.GetDimensionIndex
                            (vColumns[j].dimensionDef);
                        factTableInfo.positionVector[j] =
                            dimIndex;









                     }
                     else { // if column is a measure
                            factTableInfo.isKeyVector[j] = false;

                            // find index of vColumns[j].measureDef
                            // in the data mart structure
                            measureIndex =
                                  dataMartStructure.GetMeasureIndex
                                  (vColumns[j].measureDef);
                            factTableInfo.positionVector[j] =
                                  measureIndex;
                    }
            }
            factTableInfoList.AppendEntry (factTableInfo);
        } // else
    } // for
}
```

이 의사 - 코드는 데이터 마트 구조로부터의 정보에 기초하여 팩트 라이터 내부의 개별 팩트 타겟 테이블 정보의 리스트를 분포시킨다. 의사 - 코드는 각 데이터 마트를 통해 루프하고, 팩트 타겟 테이블 정의를 점검한다. 먼저, 타겟 테이블 정의가 이미 현재의 타겟 테이블 리스트에 있는지를 알아낸다. 만약 있다면, 이는 또 다른 데이터 마트가 이미 동일한 팩트 타겟 테이블에 그 결과를 출력했다는 것을 의미한다. 따라서, 현재의 데이터 마트의 비트는 발견된 타겟 테이블 정보 엔트리의 데이터 마트 비트 벡터로 설정될 것이다. 데이터 마트가 새로운 팩트 타겟 테이블로 출력된다면, 새로운 타겟 테이블 정보 엔트리는 리스트에 추가될 것이다. 현재의 데이터 마트의 비트는 빈 데이터 마트 비트 벡터내에 설정될 것이다. 또한, 불리안 및 위치 벡터는 타겟 칼럼을 재배치하는 목적으로 분포될 것이다.

불리안 및 위치 벡터의 크기는 타겟 테이블내의 칼럼의 개수와 동일하다. 불리안 벡터내의 각각의 엔트리는 출력 칼럼이 키이 또는 측정치인지를 나타낸다. 칼럼이 키이라면, 위치 벡터내의 상응하는 값은 키이가 발생된 디멘젼 인덱스를 나타낸다. 칼럼이 측정치라면, 위치 벡터내의 상응하는 값은 데이터 마트 구조내의 측정치 정의 리스트로부터의 측정치 인덱스를 나타낸다. " 팩트 라이터" 가 디멘젼 및 측정치 정의 리스트와 각각 동일한 차수내의 키이 및 측정치의 리스트로 주어질 때, 이 인덱스들은 칼럼을 재배치하는데 사용된다. 도 4는 전체 맵핑 메카니즘이 어떻게 동작하는지의 한 예를 나타낸다.

팩트 라이터의 내부 데이터 구조가 초기화된 후에, " 팩트 라이터" 는 팩트 집합체 리스트의 판독을 시작할 수 있고, 서로 다른 타겟 팩트 테이블에 집합체를 분배시킬 수 있다.

" 팩트 라이터" 는 집합체의 리스트를 한 번만 판독하면 된다. 각각의 레코드에 대해서, 팩트 라이터는 각각의 관련된 타겟 테이블에 매핑된 레코드를 재매핑하고 분배한다. 다음의 의사 - 코드 함수는 하나의 집합체 레코드에 주어진 다수의 타겟 테이블 분배의 프로세스를 기술한다.

```
WriteAggregateRecord (aggrRecord)
{
        // fill up pre-allocated (or data members) keys, xprod,
        // and measures lists based on re-mapping columns from
        // aggrRecord.
        for (i = 0; i < dataMartStructure.GetNumDimensions ();
                i ++) {

                keys[i] = aggrRecord[keyPosVect[i]];
                xProd[i] = aggrRecord[xProdPosVect[i]];
        }

        for (i = 0; i < dataMartStructure.GetNumMeasures ();
                i ++)
                measures[i] = aggrRecord[measurePosVect[i]];

        // get active data marts for the current cross product
        activeMarts = dataMartStructure.GetActiveMarts (xProd);

        // loop through each distinct target table
        for (i = 0; i < factTableInfoList.size (); i++) {

                // if the record is active in the current
                // target table
                if (activeMarts &
                        factTableInfoList[i].dataMartBitVector) {

                        // loop through each column
                        for (j = 0; j <
                                factTableInfoList[i].isKeyVector.size ();
                                j ++) {
                                position =
                                        factTableInfoList[i].positionVector[j];

                                // re-position key and measure columns in the
                                // pre-allocated outputRecord
                                if (factTableInfoList[i].isKeyVector[j]
                                        == true)
                                        outputRecord[j] = keys[position];
                                else
                                        outputRecord[j] = measures[position];
                        }

                        // output record to the target table
                        writeRecordToTable (outputRecord);
                }
        }
}
```

각각의 집합체 레코드를 판독하기 위한 외부 루프가 있어야 하고, 이 함수를 각각의 관련된 타겟 테이블에 대한 레코드를 기입하는 함수라고 부른다. 따라서, 상기 함수는 집합체 레코드의 입력 리스트내의 레코드의 개수만큼의 횟수로 실행된다.

이 함수는 우선 입력 집합체 레코드를 세 개의 리스트(키이, 크로스 프로덕트, 및 측정치 리스트)로 재배치한다. 매핑된 키이 및 크로스 프로덕트 리스트는 데이터 마트 구조내의 디멘젼 정의 리스트와 동일한 차수를 가져야 한다. 매핑된 측정치 리스트는 측정치 정의 리스트와 동일한 차수를 가져야 한다. 그리고, 함수는 현재의 집합체 레코드의 크로스 프로덕트를 위한 액티브 데이터 마트를 결정한다. 액티브 데이터 마트중의 적어도 하나에 관련된 각각의 타겟 테이블에 대해, 키이 및 측정치 리스트는 타겟 테이블의 바람직한 칼럼 위치로 재매핑될 것이다.

레코드가 데이터 마트 구조에 의해 예상된 것과 같이 먼저 차수가 정해진다면, 입력 집합체 레코드로부터의 칼럼을 재배치하는 제1 단계는 필요없다. 이는 이 명세서의 끝에 설명되는 것처럼 집합체 생성 동안에 데이터 마트 구조를 활용함으로써 얻어질 수 있다. 이는 데이터 마트 분포 기술에 대한 요구가 아니기 때문에, 더 일반적인 접근 방식이 적용되어 왔고, 이 가정은 전제 조건으로 놓이지 않는다.

" 디멘젼 라이터" 는 모든 데이터 마트에 대해 하나의 디멘젼의 디멘젼 타겟 테이블을 생성해야하는 함수적 구성 성분이다. 디멘젼 라이터는 입력으로 디멘젼 레코드의 리스트를 사용하고, 사용자 지정 파라미터에 대한 데이터 마트 구조를 조회하여, 각 데이터 마트에 대한 디멘젼 테이블을 출력한다. 다수의 데이터 마트가 동일한 디멘젼 테이블에 그 레코드를 병합하도록 지정된다면, 디멘젼 테이블은 2개 이상의 데이터 마트와 상응할 수 있다.

" 디멘젼 라이터" 는 데이터 분배를 시작하기 전에, 일부 내부 정보를 셋업할 필요가 있다. 우선, 디멘젼 라이터는 입력 디멘젼 레코드 정의에 기초해서 키이, 레벨 코드, 및 에버-액티브(ever-active) 스위치 위치를 발견할 필요가 있다. 이는 아래에 기술된 의사-코드에서처럼, 디멘젼 레코드 정의내의 각 필드를 통해 루핑함으로써 수행될 수 있다. 디멘젼 라이터는 발견된 키이, 레벨 코드, 및 액티브 스위치 위치를 디멘젼 라이터의 내부 변수로 저장한다.

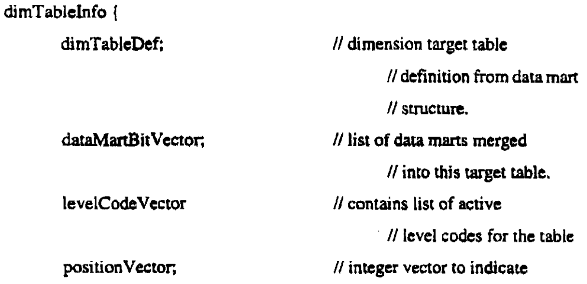
```
// dimKeyPos - key position from input dimension record.
// dimCodePos - level code position from input dimension
// record.
// dimSwitchPos - active switch position from input
// dimension record.
GetDimPositions ()
{
        dimKeyPos = -1;
        dimCodePos = -1;
        dimSwitchPos = -1:

        for (int i =0 ; i < dimRecordDef.size (): i++) {
                if (dimRecordDef[i].IsKey ())
                        dimKeyPos = i;
                else if (dimRecordDef[i].IsLevelCode ())
                        dimCodePos = i:
                else if (dimRecordDef[i].IsActiveSwitch ())
                        dimSwitchPos = i;
        }
}
```

다수의 디멘젼 테이블 분배를 하기에 앞서, 각 디멘져널 레코드에 대한 액티브 데이터 마트 비트 벡터를 얻기위해 전 - 처리(pre - processing) 단계가 요구된다. 비트 벡터는 입력 요구들에 관하여 기술된 출력 필터링 옵션을 돕기 위해 사용된다. 따라서, 출력 필터링이 실현되지 않는다면, 전 - 처리 단계 또한 제거될 수 있다. 전 - 처리 단계의 출력은 각 데이터 마트 비트 벡터가 1 디멘져널 레코드에 대응하는 각 디멘젼에 대한 데이터 마트 비트 벡터들의 리스트이다.

전 - 처리 단계는 팩트 집합체 레코드들의 리스트를 한 차례 판독함으로써 모든 디멘젼들에 대한 데이터 마트 비트 벡터들의 리스트들을 동시에 분포하게 할 수 있다. 최초에, 모든 디멘젼들의 데이터 마트 비트 벡터들의 리스트들은 0으로 설정된다. 그후에, 각 팩트 집합체 레코드에 대하여, 각 디멘젼마다 키이 및 레벨 코드 쌍이 검색된다. 각 디멘젼에 대한 레벨들의 크로스 프로덕트들과 연관된 데이터 마트 비트 벡터는 " 크로스 프로덕트에 대한 액티브 마트들 획득(Get active marts for a cross product)" 방법을 사용함으로써 검색된다. 각 키이에 대하여, 디멘져널 레코드는 키이를 조화시킴(matching)으로써 대응하는 디멘져널 레코드 리스트로부터 검색된다. 마지막으로, 검색된 디멘져널 레코드의 데이터 마트 비트 벡터는 데이터 마트 구조로부터 바로 얻어진 데이터 마트 비트 벡터와 함께 OR될 것이다. 따라서, 검색된 디멘져널 레코드는 다른 크로스 프로덕트들 안에서의 출현으로 인하여 이미 연관되어 있는, 어떤 데이터 마트들뿐만 아니라 데이터 마트 비트 벡터에 지정된 데이터 마트들 안에서 액티브로 간주된다.

따라서, 팩트 집합체들의 리스트를 한번 판독함으로써, 모든 디멘젼들의 데이터 마트 비트 벡터들은 분포될 수 있다. 전 - 처리 단계는 집합 생성동안 수행되도록 최적화될 수 있다. 그러나, 데이터 마트 비트 벡터 리스트의 초기설정이 양호한 데이터 마트 분포 기술의 일부분이기 때문에, 전 - 처리 단계는 보다 일반적인 접근 방식을 제공하기 위하여 분할된다.

" 디멘젼 라이터" 는 또한 데이터 마트 구조를 조회하고, 각 데이터 마트 안에 지정된 현재의 디멘젼에 대한 디멘젼 테이블 정의 정보를 분석할 것을 요구한다. 내부적으로, 디멘젼 라이터는 모든 데이터 마트들에 의해 요구되는 별개의 디멘젼 타겟 테이블 정의들의 리스트를 유지할 것을 요구한다. 각 디멘젼 테이블 정의와 연관된 것은 액티브 데이터 마트 비트 벡터, 액티브 레벨 코드들의 리스트 및 키이들과 속성들을 타겟 테이블에 의해 바람직한 위치로 재배치하기 위한 위치 벡터이다. 각 디멘젼 타겟 테이블을 위해 요구되는 정보는 다음의 구조안에 나열된다.



```
dimTableInfo {
        dimTableDef;                    // dimension target table
                                                // definition from data mart
                                                // structure.
        dataMartBitVector;              // list of data marts merged
                                                // into this target table.
        levelCodeVector                 // contains list of active
                                                // level codes for the table
        positionVector;                 // integer vector to indicate
```

```
                                        // mapping position.
};
```

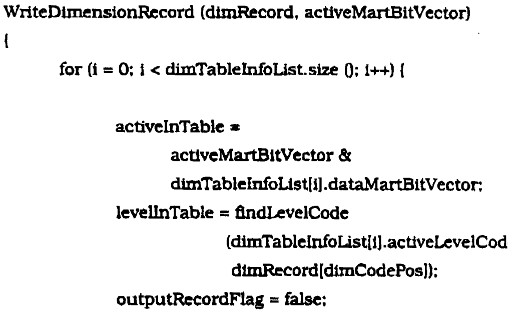
The following pseudo-code indicates how to construct the above described information:

```
// A dimension table information list will be initialized
// with distinct target table information.
InitializeDimTablesInfo ()
{
    // clear the list of target table info stored in
    // dimension writer
dimTableInfoList.clear ();

    martDefList =
        dataMartStructure.GetDataMartDefinitions ();

    // curDimDef is the current dimension definition
    curDimIndex =
        dataMartStructure.GetDimensionIndex (curDimDef);

    for (i = 0; i < martDefList.size (); i ++) {
        // check if dimension table of current data mart has
        // already appeared in the dimTableInfoList

        foundIndex = findTableDefintion
            (martDefList[i].dimTableDef[curDimIndex].
            dimTableInfoList);

        martBit = 1 << i;
        if (foundIndex >= 0) { // if found

            // add current data mart to data mart bit vector
            dimTableInfoList[foundIndex].dataMartBitVector =
            dimTableInfoList[foundIndex].dataMartBitVector |
                martBit;

            // merge current list of active level codes
            // into the found entry
            vCodes = GetActiveLevels (curDimIndex, i);
            mergeDistinctLevelCodes
            (dimTableInfoList[foundIndex].levelCodeVector,
             vCodes);
        }
        else {
            // construct a new target table entry
            dimTableInfo.dimTableDef =
                martDefList[i].dimTableDef[curDimIndex];
            dimTableInfo.dataMartBitVector = martBit;
            dimTableInfo.levelCodeVector =
                GetActiveLevels (curDimIndex, i);

            // get the list of columns from the table
            vColumns =
                martDefList[i].dimTableDef.GetColumnDefs ();
            for (j = 0; j < vColumns.size (); j++) {
                if (vColumns[j].IsKey ()) {







                    // key position from input dimension
                    // record
                    dimTableInfo.positionVector[j] =
                        dimKeyPos;
                }
                else { // attribute column

                // key position from input dimension
                    // record
                    dimTableInfo.positionVector[j] =
                        dimRecordDef.FindInputColumn
                        (vColumns[j].GetInputColumn());
                }
            } // for
        }
    } // for
}
```

이 의사 - 코드(pseudo - code)는 별개의 디멘젼 타겟 테이블 정보 리스트를 데이터 마트 구조로부터의 정보에 기초하여 라이터 안에 분포시킨다. 그것은 각 데이터 마트를 통하여 루프를 이루며, 현재 디멘젼의 그 타겟 테이블 정의를 조사한다. 첫번째로, 그것은 타겟 테이블 정의가 이미 타겟 테이블 리스트 안에 있는지를 검색한다. 검색된다면, 그것은 다른 데이터 마트가 이미 그것의 결과들을 같은 디멘젼 타겟 테이블로 출력하고 있다는 것을 의미한다. 따라서, 현재의 데이터 마트의 비트는 검색된 테이블 정보 엔트리의 데이터 마트 비트 벡터 안으로 설정된다. 또한, 현재의 데이터 마트에 대한 액티브 레벨 코드들의 리스트는 겸색된 엔트리의 레벨 코드 벡터로 병합될 것이다. 데이터 마트가 새로운 디멘젼 타겟 테이블로 출력하고 있다면, 새로운 타겟 테이블 정보 엔트리는 그 리스트에 더해질 것이다. 현재의 데이터 마트의 비트는 초기의 빈 데이터 마트 비트 벡터 안에 설정될 것이다. 레벨 코드 벡터 엔트리는 현재의 데이터 마트에 대한 액티브 레벨 코드들의 리스트와 함께 분포될 것이다. 마지막으로, 위치 벡터는 디멘젼 레코드로부터 인덱스들 또는 칼럼 위치들을 포함할 것이다. 벡터는 칼럼들을 출력될 디멘젼 레코드들로부터 타겟 테이블로 재배치하는데 사용된다.

디멘젼 라이터의 내부 데이터 구조가 초기화된 후에, 그것은 디멘져널 레코드들의 리스트를 판독하기 시작하며, 그 레코드들을 별개의 타겟 테이블들에 분배한다.

디멘젼 라이터는 디멘져널 레코드들의 리스트를 한번 판독하는 것만을 요구한다. 각각의 레코드에 대하여, 그것은 맵핑된 레코드(들)을 각각의 관련된 타겟 테이블에 재맵핑하여 분배한다. 다음의 의사 - 코드 함수는 1 디멘져널 레코드와 그것의 대응하는 액티브 데이터 마트 비트 벡터가 주어진 다수의 타겟 테이블 분배 처리 과정을 나타낸다.



```
WriteDimensionRecord (dimRecord, activeMartBitVector)
{
    for (i = 0; i < dimTableInfoList.size (); i++) {

        activeInTable =
            activeMartBitVector &
            dimTableInfoList[i].dataMartBitVector;
        levelInTable = findLevelCode
                (dimTableInfoList[i].activeLevelCodes,
                 dimRecord[dimCodePos]);
        outputRecordFlag = false;
```

```
                // update the ever active switch
                if (activeInTable)
                        dimRecord[dimSwitchPos] = true;

                // determine whether to output the record depending
                // on output filtering option
                if (filterOption == AllRecords) {
                        if (levelInTable)
                                outputRecordFlag = true;
                }
                else
                if (filterOption == ActiveInDataMarts) {
                        if (levelInTable && activeInTable)
                                outputRecordFlag = true;
                }
                else
                if (filterOption == EverActiveInDataMarts) {
                        if (levelInTable &&
                                 dimRecord[dimSwitchPos] == true)
                                outputRecordFlag = true;
                }

                if (outputRecordFlag == true) {
                        for (j = 0; j <
                                dimTableInfoList[i].positionVector.size ();
                                j ++) {

                                position =
                                        dimTableInfoList[i].positionVector[j];
                                outputRecord[j] = dimRecord[position];

                        writeOutputRecord (outputRecord);
                        }
        } // for
}
```

액티브 데이터 마트 비트 벡터는 전-처리 단계의 출력이다. 각 디멘져널 레코드를 판독하고, 각각의 관련된 타겟 테이블에 그 레코드를 기입하는 기능을 호출하기 위하여 외부 루프가 있어야 한다. 따라서, 이 기능은 디멘져널 레코드들의 입력 리스트 안의 레코드들의 수만큼 실행된다.

그 함수는 그 디멘젼 타겟 테이블 정보의 리스트를 통하여 루프를 이룬다. 그것은 첫번째로 두개의 플래그들(flags) - activeInTable과 levelInTable - 을 초기화시킨다. activeInTable은 현재의 디멘젼 레코드가 팩트 안에서 액티브인지를 나타낸다. levelInTable은 현재의 디멘젼 레코드의 레벨 코드가 현재의 타겟 테이블의 레벨 코드 리스트 안에 있는지를 나타낸다. 그후에, 그것은 계속적인 에버 액티브(ever active) 스위치를 갱신한다. activeInTable 플래그가 참(true)이면 스위치는 참으로 설정된다. 필터링 옵션에 기초하여, 로직(logic)은 현재의 디멘젼 레코드들이 타겟 테이블로 출력되어야 하는지를 판정한다. 레코드가 출력되어야 하는 경우에, 키이와 속성 칼럼들은 타겟 테이블의 바람직한 칼럼 위치로 재맵핑될 것이다.

각 입력 디멘젼 레코드와 연관된 " 에버 액티브" 플래그는 " 데이터 마트들 안에서의 에버 액티브(Ever Active in Data Marts)" 비트 벡터로 교체될 수 있다는 것이 주목되어야 한다. 이는 타겟 데이터 마트들이 질문(question) 안에서 데이터 마트에 대응하는 팩트 데이터를 수신한 적이 있다면, 디멘젼 레코드들이 타겟 데이터 마트들로 출력될 수 있게 한다. 그러나, 이는 집합들 사이, 양호하게는 입력 디멘젼 테이블들 안에서, 출력 데이터 마트들에 관련된 계속적인 데이터 유지를 요구할 것이다. 이는 출력 데이터 마트 정의들과 동기화된(synchronized) 입력 팩트 데이터를 유지하는 문제들을 야기하며, 이런 이유때문에 여기 기술된 양호한 실시예에서 실현되지 않는다. 양호한 실시예에서 실현된 " 에버 액티브" 플래그는 출력 데이터 마트들의 구성에 의존적인 어떤 데이터도 포함하지 않는다.

시스템 구성요소들의 동작과 그 사이의 흐름은 이후에 기술될 것이다. 도 5는 이전의 섹션들에서 기술된 입력 요구들과 시스템 구성요소들이 주어진 데이터 흐름의 개관을 도시한다.

도면에 도시된 바와 같이, 데이터 마트 구조는 분배가 시작할 수 있기 전에 구성되어야 한다. 데이터 마트 구조가 초기화되면, 디멘져널 레코드들을 위한 전-처리 단계는 발생한다. 전-처리 단계는 모든 디멘젼 레코드 정의들과 팩트 집합체 레코드 정의들로부터 키이들과 레벨 코드들의 필드 위치들을 확립한다. 그후에 팩트 집합체 레코드들의 리스트의 판독이 개시된다. 각 팩트 집합체 레코드에 대하여, 모든 팩트 집합체 레코드의 대응하는 디멘젼 레코드들의 데이터 마트 비트 벡터들(각 디멘젼으로부터 하나)은 갱신될 것이다. 전-처리 단계의 세부사항들은 " 디멘젼 라이터" 를 참조하여 이미 기술되었다.

전-처리가 완료되면, " 팩트 라이터" 와 " 디멘젼 라이터들" 의 동작들은 병렬로 진행될 수 있다. " 팩트 라이터" 는 팩트 집합체 레코드들의 리스트를 입력으로 취하고, 데이터 마트 구조를 조회하며, 각 데이터 마트에 대한 팩트 테이블을 생성한다. 각 " 디멘젼 라이터" 는 디멘져널 레코드들의 리스트와 대응하는 전-처리 단계로부터 생성된 데이터 마트 비트 벡터들의 리스트를 입력으로 취하고, 데이터 마트 구조를 조회하며, 각 데이터 마트에 대한 디멘젼 테이블을 생성한다. 따라서, 라이터는 어떤 상호 의존성도 가지지 않으며, 그들의 임무를 독립적으로 수행할 수 있다. 그러나, 데이터 마트들이 참조 무결성(referential integrity)이 실행되는 테이블들 내 데이터 베이스에 어떤 특정한 순간에 로드(load)되면, 팩트 데이터안의 모든 키이 값들은 디멘젼 데이터안에 대응하는 값들을 가져야 한다. 모든 대응하는 키이 값들이 디멘젼 라이터에 의해 기입되기 전에, 팩트 라이터가 엔트리를 기입하면, 참조 보전에서 브레이크다운이 있을 것이다. 라이터들이 그런 환경에서 사용되고 있다면, 참조 보전 실행은 프로세스 개시전에 불가능하게 되며, 프로세스 종료시 다시 가능하게 된다. 대안적으로, 팩트 라이터는 디멘젼 라이터들이 프로세싱을 종료한 경우에만 개시하도록 설정될 수 있다.

다음은 본 발명의 데이터 마트 분포 기술의 주요한 이익과 장점들의 요약이다.

1) 단일 집합 생성 프로세스의 다중 데이터 마트들의 분포를 허락한다.

2) 데이터 마트들이 집합체들의 요구들과 중복될 수 있거나 심지어는 집합들의 레벨들과 같기 때문에, 다중 데이터 마트들을 한번에 생성하기에 효과적인 보다 많은 리소스(resource). 이는 입력 팩트 파일 판독 및 동일한 집합 버킷들(buckets)의 수회 생성의 부담(overhead)을 제거한다.

3) 데이터 마트들이 다른 스타 스키마 구성들과 다른 집합체들을 가지도록 허락한다. 각 데이터 마트의 구성은 손질되거나(tailored) 다른 사용자 그룹들의 요구들에 맞추어 진다.

4) 디멘젼 테이블의 레코드들은 팩트 테이블의 집합된 레코드들에 근접하게 결합된다. 레코드는 레코드 레벨이 대응하는 팩트 테이블에 대한 출력 크로스 프로덕트들의 리스트에 관여하는 경우에만, 디멘젼 테이블 안에 있다. 생성된 데이터 마트들은 보다 적은 오차를 내는 경향이 있다.

5) 큰 데이터 세트들에 대한 데이터 분배(partition)를 허락한다. 논리적으로 분리된 데이터 마트들은 다른 팩트 테이블들을 가지고 있는 동안에 불필요한 디멘져널 데이터의 복제(duplication)를 방지하기 위하여 디멘젼 테이블들을 결합 및 공유할 수 있다. 데이터 마트 분배의 플렉시블 스킴으로, 데이터는 다른 디멘젼이나 팩트 테이블에서 나누어지거나 합쳐질 수 있다.

6) 플렉시블 디멘젼 출력 필터링 시킴에 따라 사용자는 디멘젼 테이블로 출력되는 디멘젼 레코드들의 올바른 세트를 선택할 수 있게 된다.

본 명세서에서 제시되는 이러한 기술은 더욱 일반적인 시나리오를 커버하도록 의도된다. 최적화를 위하여 이러한 기술에 추가적인 적절한 가정이 설정될 수 있다. 이러한 부분에서는 가능한 최적화의 약간의 변형을 기술한다.

앞서 설명한 바와 같이, 디멘젼 출력 필터링 플래그가 본 발명의 데이터 마트 분포 기술에 요구되는 것은 아니다. 디멘젼 기록 분배의 성능을 향상시키는 것이 제공된다. 그 로직은, 네개의 출력 필터링 옵션인, 디멘젼 출력의 없음, 모든 디멘젼의 기록, 데이터 마트에서의 활성화, 및 데이터 마트에서의 에버 엑티브를 인에이블하기 위하여 제공된다. 추가적인 출력 필터링 옵션은 레벨 코드 기준이나 활성 데이터 마트의 다른 조합들을 근거로 추가될 수 있다. " 데이터 마트에서의 에버 엑티브" 출력 필터링 성능은, " 에버 엑티브" 플래그 오버 타임을 유지하는데 필요하기 때문에, 지속적으로 유지되도록 현재까지 보여지는 디멘젼 레코드를 필요로 한다. 만약 " 데이터 마트에서의 에버 엑티브" 필터링 성능이 지원될 필요가 없다면, 디멘젼 레코드 또한 지속적으로 요구되지는 않는다.

상술한 기술에서, 팩트 결합 레코드에서의 측정치와 관련되어, 모든 측정치의 슈퍼 세트(superset)가 필요하다는 것을 제외한 다른 가정은 없었다. 또한 팩트 집합 레코드와 디멘젼 레코드의 필드 위치에 관해서 설정된 가정은 없었다. 바람직한 실시예에서는, 데이터 마트 구조가 집합 발생 이전에 형성되었기 때문에 어떤 적절한 가정이 설정되었다. 집합 발생 이전에 사용자가 모든 디멘젼과 데이터 마트 정의를 사용할 수 있다는 것은 상당히 일반적이다.

데이터 마트 구조가 집합 발생이전에 주어지면, 필요한 측정치의 세트가 알려진다. 또한 데이터 마트 구조 안에서의 디멘젼과 측정치의 차수가 알려진다. 이 경우, 단지 요구된 측정치는 집합 발생 도중에 발생된다. 각 팩트 집합 레코드에 등록된 키이, 레벨 코드, 및 측정치 또한 미리 가정된 임의의 차수로 있을 수 있다. 예를들어, 레코드에서 제1 N 필드는 데이터 마트 구조에서 지정되는 것과 같은 동일한 디멘젼의 차수에서 키이 필드가 될 수 있으며, 다음 N 필드는 데이터 마트 구조에서 지정되는 것과 같이 동일한 디멘젼의 차수에서 또한 레벨 코드 필드가 될 수 있으며, 마지막 M 필드는 데이터 마트 구조에서 지정되는 것과 같이 동일 차수에서 측정치가 될 수 있다. 이 경우, 팩트 집합 레코드로부터의 초기의 재-위치설정(positioning) 단계는 삭제될 수 있다. 디멘젼 레코드에 대하여, 키이, 레벨 코드, 및 에버 액티브 활성 스위치는 최적화를 위하여 첫번째 3개의 필드에 위치하도록 가정될 수 있다.

집합체 발생이전에 데이터 마트 구조가 작성되면, 데이터 마트 비트 벡터를 분포시키는 전-처리 단계 역시 삭제될 수 있다. 집합체 발생과 동일한 시간동안 작업이 이루어질 수 있다. 그러면, 단지 데이터 마트 비트 벡터를 분포시키기 위한 목적으로, 팩트 집합체의 리스트를 읽을 필요는 없게 된다.

바람직한 실시예에서, 입력 팩트 레코드가 모든 관련 집합체 버켓으로 aggregate되면, 입력 팩트 레코드에 관련하는 모든 디멘젼 레코드의 데이터 마트 비트 벡터와 그 집합체 버켓은 갱신될 것이다. 비트 벡터를 갱신하는 로직은 " 디멘젼 라이터" 부분에서 설명된 바와 동일하다. 그러나, 이 경우, 한번더 팩트 집합체 레코드를 읽을 필요가 없어진다. 각 디멘젼에서 데이터 마트 비트 벡터가 출력 필터링을 위하여, 다시 사용된다. 만약 출력 필터링이 제공되지 않거나 제공된 필터링 옵션이 데이터 마트 비트 벡터를 필요로 하지 않는다면, 데이터 마트 비트 벡터를 분포시키는 단계는 삭제될 수 있다.

이상과 같이, 본 발명의 바람직한 실시예가 나타나고 설명되는 한면, 당업자는 본 발명 사상을 벗어나지 않는 범위 내에서 변경 및 변형이 이루어질 수 있다는 것을 분명히 알 것이며, 그 범위는 첨부하는 청구항들에 의해 정의된다.

(57) 청구의 범위

청구항 1.

입력 팩트(fact) 데이터와 적어도 하나의 디멘젼(dimension) 테이블을 포함하는 제1 데이터 마트(mart)로부터 다수의 출력 데이터 마트를 발생하는 방법에 있어서,

a) 입력 팩트 데이터의 레벨과는 다른 하나 이상의 레벨로 가산된(summarized) 상기 팩트 데이터를 나타내는 다수의 레코드를 포함하는 집합된 팩트 데이터를 발생하는 단계와;

b) 상기 집합된 팩트 데이터를 상기 다수의 출력 데이터 마트로 분산하는 단계를 포함하며, 임의의 집합된 팩트 데이터 레코드는 상기 데이터 마트들 중 2개 이상의 데이터 마트로 분산되며, 상기 적어도 소정의 출력 데이터 마트로 분산된 팩트 데이터 레코드의 세트는 동일하지 않은 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 2.

제1항에 있어서,

각각의 집합된 팩트 데이터 레코드는 단지 한번 발생되어 상기 집합 팩트 데이터 레코드를 요구하는 상기 출력 데이터 마트로 출력되는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 3.

제1항에 있어서,

각 출력 데이터 마트에 의해 요구되는 레벨들 또는 레벨 크로스 프로덕트들을 지정하는 적어도 하나의 데이터 구조를 제공하는 단계;

상기 입력 팩트 데이터로부터 발생될 필요가 있는 집합 팩트 데이터 레코드를 설정하는 단계; 및

각각의 요구된 집합 팩트 데이터 레코드를 발생하는 단계를 더 포함하며,

각각의 발생된 집합 팩트 데이터 레코드에 대하여는,

1) 상기 출력 데이터 마트들 중에서 상기 집합된 팩트 데이터 레코드를 요청하는 출력 데이터 마트를 상기 적어도 하나의 데이터 구조로부터 설정하는 단계, 및

2) 상기 집합 팩트 데이터 레코드를 요청하는 상기 각 출력 데이터 마트에 상기 집합된 팩트 데이터 레코드를 제공하는 단계를 포함하는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 4.

제3항에 있어서,

상기 데이터 구조는 상기 적어도 하나의 디멘젼으로부터의 레벨들 또는 레벨 크로스 프로덕트의 리스트를 포함하며, 상기 리스트의 각 요소는 하나 이상의 출력 데이터 마트와 연관됨으로써, 각 집합 팩트 데이터 레코드가, 상기 레코드가 연관되는 레벨들 또는 레벨 크로스 프로덕트에 기초하여 적절한 출력 데이터 마트에 연관될 수 있도록 하는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 5.

제1항에 있어서, 집합된 팩트 데이터는 서로 다른 데이터 마트들에 의해 공유되는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 6.

제1항에 있어서,

입력 디멘젼 테이블로부터 상기 출력 데이터 마트의 각각에 대하여 디멘젼 테이블을 발생하는 단계를 더 포함하고, 각각의 발생된 타겟 디멘젼 테이블은 상기 연관된 데이터 마트에 의해 요구되는 레벨에 연관된 디멘젼 레코드만을 포함하는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 7.

제6항에 있어서,

상기 적어도 하나의 발생된 디멘젼 테이블은 두개 이상의 출력 데이터 마트에 의해 공유되는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 8.

제6항에 있어서,

특정 디멘젼에 대하여 타겟 디멘젼 테이블을 발생하는 단계는, 상기 디멘젼 레코드에 연관된 레벨이 상기 타겟 디멘젼 테이블에 연관된 레벨의 리스트에 있을 때에만, 상기 타겟 디멘젼 테이블에 연관된 레벨의 리스트를 발생하고 타겟 디멘젼 테이블로 디멘젼 레코드를 출력함으로써 수행되는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 9.

제8항에 있어서,

특정 타겟 디멘젼 테이블에 대한 상기 레벨의 리스트는 상기 타겟 디멘젼 테이블이 대응하는 상기 데이터 마트 각각에 연관된 상기 적절한 디멘젼에서 레벨의 리스트를 병합함으로써 발생되며,

특정 데이터 마트에 연관된 각각의 레벨의 리스트는 상기 데이터 마트가 연관되는 모든 크로스 프로덕트에 대응하는 디멘젼에서의 레벨의 세트를 포함하는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 10.

제9항에 있어서,

각 데이터 마트에서 발생된 디멘젼 테이블은 해당 데이터 마트로 출력되는 상기 집합된 출력 팩트 데이터에서 나타나는 상기 입력 디멘젼 테이블로부터의 레코드만을 포함하는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

청구항 11.

제10항에 있어서,

특정 디멘젼에서 각 데이터 마트에 대하여 디멘젼 테이블을 발생하는 상기 단계는,

a) 디멘젼 레코드를 각 데이터 마트에 연관시키고, 초기에는 각 디멘젼 레코드를 아무런 데이터 마트에도 연관시키지 않는 저장수단을 제공하는 단계와;

b) 각 디멘젼에서 지정 디멘젼 레코드에 대응하는 디멘젼 엔트리의 세트를 포함하는 각 팩트 데이터 엔트리에 대하여,

1) 각 디멘젼에서 상기 디멘젼 엔트리에 연관된 레벨을 설정하는 과정과,

2) 상기 레벨의 상기 크로스 프로덕트에 기초하여 상기 팩트 데이터 엔트리에 연관된 상기 데이터 마트를 설정하고, 각 디멘젼에 대하여, 상기 데이터 마트가 상기 디멘젼 엔트리에 미리 연관되어 있지 않으면 상기 저장 수단에서 상기 디멘젼 엔트리에 상응하는 상기 디멘젼 레코드에 상기 각 데이터 마트를 연관시키는 과정

을 수행하는 단계와;

c) 상기 디멘젼 레코드가 상기 저장 수단에서 상기 출력 디멘젼 테이블이 연관되는 데이터 마트에 연관되면, 각 디멘젼 레코드를 각 출력 디멘젼 테이블로 출력하는 단계

에 의해 수행되는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

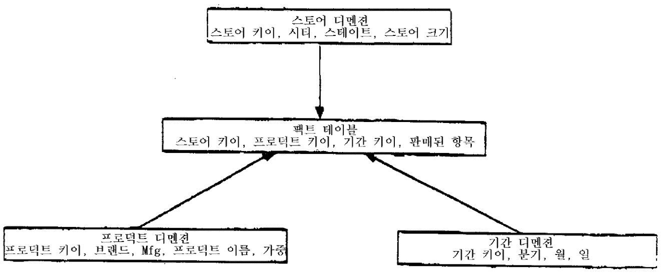
청구항 12.

제6항에 있어서,

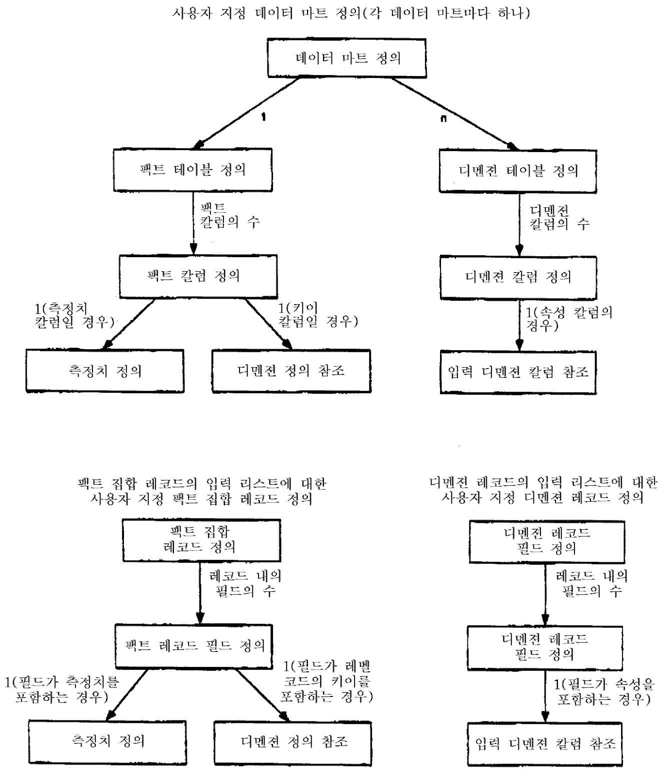
발생된 디멘젼 테이블은 서로 다른 출력 데이터 마트에 의해 공유되는 것을 특징으로 하는 출력 데이터 마트 발생 방법.

도면

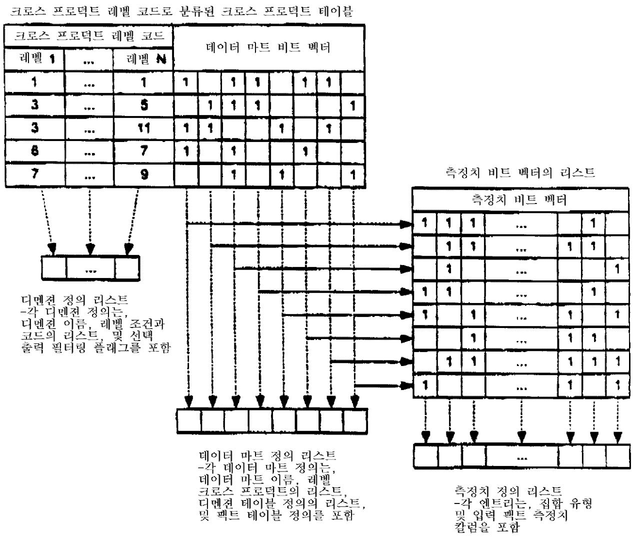
도면 1


스토어 디멘젼
스토어 키이, 시티, 스테이트, 스토어 크기
팩트 테이블
스토어 키이, 프로덕트 키이, 기간 키이, 판매된 항목
프로덕트 디멘젼
프로덕트 키이, 브랜드, Mfg, 프로덕트 이름, 가중
기간 디멘젼
기간 키이, 분기, 월, 일

도면 2


사용자 지정 데이터 마트 정의(각 데이터 마트마다 하나)
데이터 마트 정의
1
n
팩트 테이블 정의
디멘전 테이블 정의
팩트 칼럼의 수
디멘전 칼럼의 수
팩트 칼럼 정의
디멘전 칼럼 정의
1(측정치 칼럼일 경우)
1(키이 칼럼일 경우)
1(속성 칼럼의 경우)
측정치 정의
디멘전 정의 참조
입력 디멘전 칼럼 참조
팩트 집합 레코드의 입력 리스트에 대한 사용자 지정 팩트 집합 레코드 정의
디멘전 레코드의 입력 리스트에 대한 사용자 지정 디멘전 레코드 정의
팩트 집합 레코드 정의
디멘전 레코드 필드 정의
레코드 내의 필드의 수
레코드 내의 필드의 수
팩트 레코드 필드 정의
디멘전 레코드 필드 정의
1(필드가 측정치를 포함하는 경우)
1(필드가 레벨 코드의 키이를 포함하는 경우)
1(필드가 속성을 포함하는 경우)
측정치 정의
디멘전 정의 참조
입력 디멘전 칼럼 참조

도면 3



크로스 프로덕트 레벨 코드로 분류된 크로스 프로덕트 테이블

크로스 프로덕트 레벨 코드			데이터 마트 비트 벡터							
레벨 1	...	레벨 N								
1	...	1	1		1	1		1	1	
3	...	5		1	1	1				1
3	...	11	1	1			1		1	
8	...	7	1		1			1		
7	...	9			1		1			1

측정치 비트 벡터의 리스트

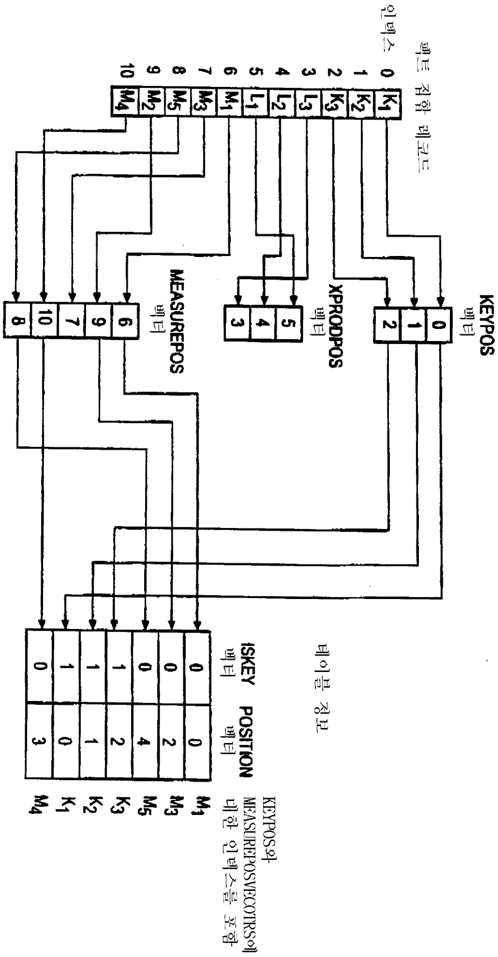
측정치 비트 벡터						
1	1	1	...		1	
	1	1	...	1	1	
	1		...			1
1	1		...		1	
1		1	...	1		1
		1	...	1	1	
	1	1	...	1	1	1
1			...	1		1

디멘젼 정의 리스트
-각 디멘젼 정의는,
디멘젼 이름, 레벨 조건과
코드의 리스트, 및 선택
출력 필터링 플래그를 포함

데이터 마트 정의 리스트
-각 데이터 마트 정의는,
데이터 마트 이름, 레벨
크로스 프로덕트의 리스트,
디멘젼 테이블 정의의 리스트,
및 팩트 테이블 정의를 포함

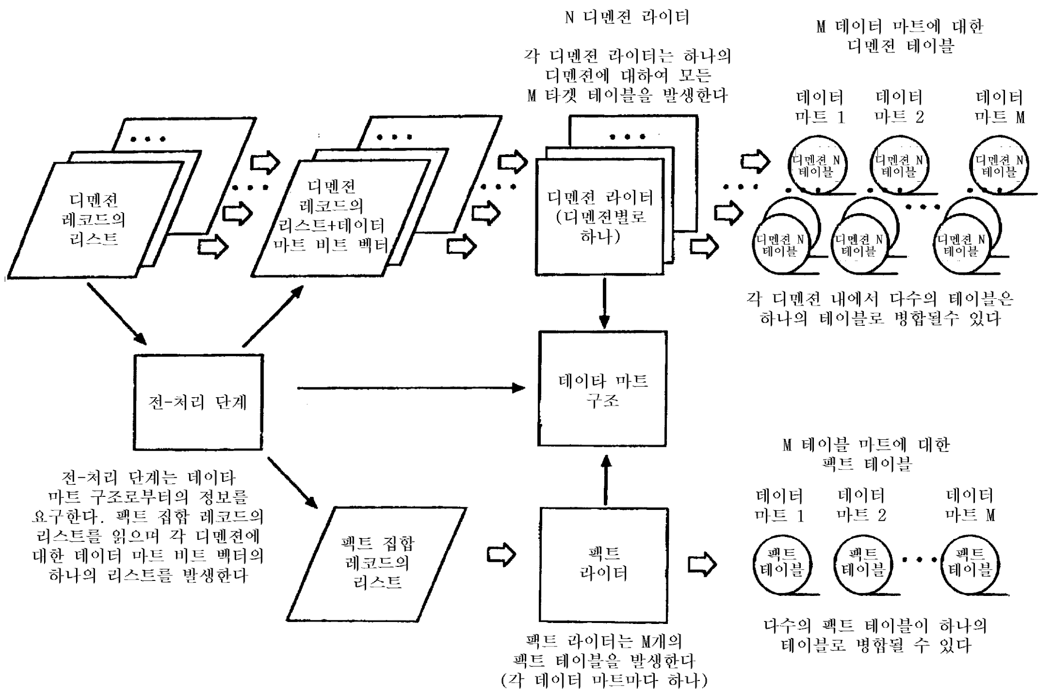
측정치 정의 리스트
-각 엔트리는, 집합 유형
및 입력 팩트 측정치
칼럼을 포함

도면 4



	ISKEY 벡터	POSITION 벡터
M1	0	0
M3	0	2
M5	0	4
K3	1	2
K2	1	1
K1	1	0
M4	0	3

도면 5


디멘전
레코드의
리스트
디멘전
레코드의
리스트+데이터
마트 비트 벡터
N 디멘전 라이터
각 디멘전 라이터는 하나의
디멘전에 대하여 모든
M 타겟 테이블을 발생한다
디멘전 라이터
(디멘전별로
하나)
M 데이터 마트에 대한
디멘전 테이블
데이터
마트 1
데이터
마트 2
데이터
마트 M
디멘전 N
테이블
각 디멘전 내에서 다수의 테이블은
하나의 테이블로 병합될수 있다
전-처리 단계
데이타 마트
구조
전-처리 단계는 데이타
마트 구조로부터의 정보를
요구한다. 팩트 집합 레코드의
리스트를 읽으며 각 디멘전에
대한 데이터 마트 비트 벡터의
하나의 리스트를 발생한다
팩트 집합
레코드의
리스트
팩트
라이터
팩트 라이터는 M개의
팩트 테이블을 발생한다
(각 데이터 마트마다 하나)
M 테이블 마트에 대한
팩트 테이블
데이터
마트 1
데이터
마트 2
데이터
마트 M
팩트
테이블
다수의 팩트 테이블이 하나의
테이블로 병합될 수 있다