42 1993. 4. 정보과학회지 제 11 권 제 2 호

□특 집□

객체지향 데이타베이스 기술†

UniSQL, Inc 9390 Research Blvd. Austin, Texas 78759 김 원

목 차

과거 십년 동안 객체지향(object-oriented) 기술은 프로그래밍 언어, 사용자 인터페이스, 데이타베이스, 운영체제, 전문가 시스템 등의 영역에 적용되어 왔다. 수년 동안 객체지향 데이타베이스라는 이름을 가진 제품이 시장에 나타났으며, 관계형 데이타베이스 시스템을 개발했던 회사들이 그들의 제품에 객체지향 기능을 추가하겠다고 선언하였다. 몇몇 회사는 현재 관계형 기능과 객체지향 기능을 하나의 데이타베이스 시스템에 결합하여 지원하고 있다. 그러나, 이러한 활동에도 불구하고, 사용자, 업계 잡지, 그리고 개발자까지도 객체지향 데이타베이스 시스템, 객체지향 기능을 갖는 관계형 데이타베이스 시스템, 그리고 심지어 이러한 시스템의 필요성에까지 많은 혼동을 갖고 있다. 보고의 목적은 객체지향 데이타베이스 시스템의 원리를 소개하고, 여러 개발자가 채택한 방식을 구분하여 객체지향 데이타베이스 기술을 차세대 데이타베이스 기술로 정착하기 위하여 취해야 할 것이 무엇인지를 보이는 데 있다.

I. 정 의

현재 사용되는 객체지향(object-oriented) 기술은 객체지향 프로그래밍 언어(예, C++, Smaltalk), 객체지향 데이타베이스 시스템(object-oriented database system: OODB), 객체지향 사용자 인터페이스(예, Macintoch와 Microsoft의 윈도우 시스템, Frame, Interleaf의 데스크탑 출판시스템) 등을 포함한다. 객체지향 기술이란 사용자에게 "객체지향 개념"에 기반을 둔 각종 기능을 지원하는 기술이다. "객체지향 개념을 정의하기 위해서는 먼저 "객체(OBJECT)"란 무엇인가를 이해해야 한다.

"객체"는 실세계의 어떤 개체를 나타내는 "자료(data)"와 "프로그램(program)"의 결합을 의미한다. 예를 들어, 나이가 25세이고 연봉이 $25,000인 Tom이란 사원을 객체로 표현하면, "자료"는 (이름 : Tom, 나이 : 25, 연봉 : $25,000), "프로그램"은 (고용, 자료검색, 나이변경, 연봉변경, 해고)이 될 수 있다. 자료부분은 여러 가지 자료형으로 구분될 수 있는데, 예를

† 본 고는 김원박사의 "on object-oriented database technology"의 영문본을 저자의 승인 후, 한국과학기술원의 이윤준, 김형철이 번역 편집한 것입니다

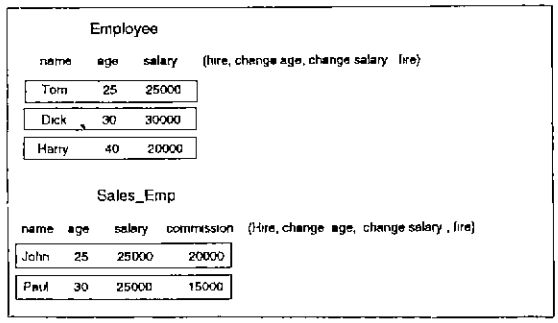
들면, 이름은 스트링(string), 나이는 정수, 연봉은 금액형(monetary) 등을 사용할 수 있다. 그러나, 일반적으로는 사용자가 정의한 임의의 자료형(예를 들면, Employee)도 사용될 수 있다. 위의 예에서 이름, 나이, 연봉 등을 애트리뷰트(attribute)라 한다.

객체는 자료와 프로그램을 캡슐화(encapsulate)한다고 말한다. 이는 사용자가 객체 캡슐 내부를 보지 못하고, 대신 그 객체의 프로그램 부분을 호출함으로써 객체를 사용함을 의미한다. 이는 기존의 프로그래밍 환경에서 입력 인자를 넘겨주고 출력인자를 통해 결과를 받는 프로시듀어(procedure) 호출과 의미가 크게 다르다.

"객체지향"은 대체로 객체의 캡슐화와 계승(inheritance)의 결합을 의미한다. "계승"은 때때로 "재사용(reuse)"이라 부르기도 한다. "계승"은 대체로 새로운 객체가 기존의 객체를 확장하여 생성될 수 있음을 나타낸다. 이제, "계승"이라는 용어를 보다 정확히 설명하면, 각 객체는 자료 부분과 프로그램 부분으로 구성된다. 서로 같은 자료 부분(즉, 같은 애트리뷰트)과 같은 프로그램 부분을 갖는 모든 객체를 묶어서 클라스(class) 혹은 형(type)이라 부른다. 어떤 클라스가 다른 어떤 클라스로부터 애트리뷰트와 프로그램 부분을 계승받도록 클라스들을 배치할 수 있다.

Tom, Dick, 그리고 Harry는 각각 Employee 객체이다. 이들 객체의 자료 부분은 이름, 나이, 연봉으로 이루어지고, 모두 같은 프로그램 부분을 갖는다(즉, 고용, 자료검색, 나이변경, 연봉변경, 해고). 프로그램 부분의 각 프로그램을 "메소드(method)"라 부른다. "클라스"는 같은 애트리뷰트와 메소드를 갖는 객체의 집합을 칭한다. 즉, 위의 예에서 Tom, Dick, Harry는 Employee라는 클라스에 속한다. 이 클라스는 다시 임의의 객체의 에트리뷰트 자료형으로 쓰일 수 있다.

이제 두 명의 판매사원, John과 Paul을 생성하는 경우를 생각하여 보자. 그런데, 판매사원은 커미션이라는 새로운 애트리뷰트를 갖는다. 판매사원은 Employee 클라스에 속할 수 없으나, 기존의 Employee로부터 모든 애트리뷰트와 프로그램을 계승받고 커미션이라는 에트리뷰트를 추가하여 Sales_Employee라는 새로운 클라스를 생성할 수 있다. 이때, Sales_Employee는 Employee의 "하위클라스(subclass)"가 된다. 사용자는 이제 Sales_Employee 클라스에 속하는 두 판매사원을 생성할 수 있다. 그림 1은 객체를 모아 클라스를 구성하고 애트리뷰트와 메소드를 계승하는



(그림 1) 객체, 클라스, 계승

예를 보인 것이다. 사용자는 기존의 클라스의 하위클라스로서 새로운 클라스를 생성할 수 있다. 일반적으로, 하나의 클라스는 하나 혹은 그 이상의 클라스로부터 계승받고, 이러한 계승으로 인한 구조는 DAG(directed acyclic graph)를 형성한다. 이 계승구조를 "계승 계층(inheritance hierarchy)" 혹은 "클라스 계층(class hierarchy)"이라 부른다.

객체지향 개념은 캡슐화와 계승이 함께 적용될 때 제기능을 발휘한다.

- 계승을 통하여 서로 다른 클라스가 애트리뷰트나 메소드를 공유할 수 있기 때문에, 서로 다른 클라스에 속하는 객체에 대해 같은 프로그램을 수행할 수 있다. 이는 데스크탑 출판 시스템이나 윈도우 관리 시스템이 지원하는 사용자 인터페이스의 기초가 된다. 같은 종류의 프로그램(예, 열기, 닫기, 삭제, 생성, 이동, 기타)을 서로 다른 자료형(이미지, 텍스트, 음성, 디렉토리, 기타)에 적용할 수 있다.
- 많은 클라스가 정의되어 있고, 각 클라스가 많은 애트리뷰트와 메소드를 갖고 있으면, 애트리뷰트와 프로그램의 공유 효과는 더욱 커진다. 애트리뷰트나 프로그램이 처음부터 다시 정의되거나 작성될 필요성이 없다. 새로운 클라스를 생성하기 위해 기존 클라스의 애트리뷰트나 메소드를 변경하지 않고, 단지 새로운 애트리뷰트나 메소드를 추가한다. 따라서 기존 클라스에 오류를 발생시킬 기회가 줄어든다.

II. 객체지향 데이타베이스 시스템의 잠재성

객체지향 프로그래밍 언어(OOPL)는 객체들을 조직화하기 위해 클라스를 생성하고, 객체를 생성하며, 계승 계층을 통해 하위클라스가 상위클라스(superclass)로부터 애트리뷰트와 메소드를 계승 받을 수 있

도록 하며, 특정 객체를 접근하는 메소드를 호출하는 기능을 지원한다. OODB는 이와같은 기능을 지원함과 동시에 오늘날의 관계형 데이타베이스 시스템(RDB)이 지원하는 표준기능을 지원해야 한다. 이러한 표준 기능으로는 자료검색을 위한 비절차식(nonprocedural) 질의 기능, 자동 질의 최적화, 동적 스키마 변경(클라스 정의 변경 및 계승 구조 변경), 질의처리 성능 향상을 위한 접근기법의 관리(예, B^+-트리, 확장해쉬, 정렬 등), 트랜잭션 관리, 동시성 제어, 시스템 고장으로부터 회복, 보안 및 권한부여 등이 있다. OOPL을 포함한 프로그램 언어들은 단일 사용자와 비교적 작은 규모의 데이타베이스를 염두에 두고 설계되는 반면, 데이타베이스 시스템은 많은 사용자와 매우 큰 규모의 데이타베이스를 염두에 두고 설계된다. 따라서, 데이타베이스 시스템에서는 성능, 보안 및 권한부여, 동시성제어, 동적 스키마 변경 등이 중요한 문제가 된다. 더구나, 데이타베이스 시스템은 매우 중요한 자료를 정확히 관리해야 하기 때문에 트랜잭션 관리, 동시성 제어, 회복 기능 등이 중요하다.

데이타베이스 시스템의 기능이 주 프로그램(host program)으로 작성된 응용프로그램에 의해 호출되기 때문에 OODB를 설계하는 데 두 가지 접근 방식이 있다. 하나는 OOPL로 작성된 프로그램에서 생성된 자료를 저장, 관리하는 방식으로, 현재 몇 개의 OODB가 C^{++}나 Smaltalk 프로그램의 자료를 저장, 관리하여 준다. 물론 RDB를 이용하여 이러한 객체를 관리할 수 있으나, RDB는 객체를 이해하지 못하며, 특히, 메소드와 계승을 지원하지 못한다. 그러므로, 메소드와 계승을 지원하고, 객체를 튜플로 변환할 수 있는 소위, "객체 관리자(object manager)", 혹은, "객체지향층(object-oriented layer)"이라 불릴 수 있는 소프트웨어가 구현되어야 한다. 그러나, 이와 같이, 객체관리자와 RDB가 결합되면 이것이 사실상 하나의 OODB이다(성능의 문제가 있다).

또 한 가지 접근 방식은 OOPL을 사용하지 않는 것이다. 사용자가 클라스, 객체, 계승 계층 등을 생성하면, 데이타베이스 시스템은 이를 저장 관리한다. 이 방식을 이용하면 실제로 C, FORTRAN, COBOL과 같은 OOPL이 아닌 언어를 사실상 객체지향 언어로 바꾸게 된다. 실제로, C^{++}는 C를 OOPL로 바꾼 것이며, CLOS는 CommonLISP에 객체지향 기능을 추가한 것이다. 이 접근 방식을 이용하여 설계한 OODB를 OOPL로 생성된 객체를 저장하는 데에도 물론 사용할 수 있다. 비록, OOPL 객체을 데이타베이스 시스템의 객체로 전환하는 단계가 필요하겠지만 이는 RDB에 객체관리자를 올리는 것보다 한층 쉬울 것이다.

C^{++}의 인기가 증가하고 있기는 하나, 유일한 데이타베이스 응용 프로그래밍 언어가 아니고, 또한 데이타베이스 시스템과 프로그래밍 언어 사이에는 많은 차이점이 있다는 점에서 두번째 접근 방식이 보다 실제적이라 할 수 있다. 그러나, 어떤 접근 방식으로든 OODB가 제대로 구현된다면 이는 응용 프로그래머의 생산성과 개발된 응용 프로그램의 성능 향상에 큰 도약을 가져올 것이다.

이러한 기술적 도약의 한 원인은 객체지향 개념을 통해 데이타베이스 진화 역사상 처음으로 데이타베이스 설계와 프로그램을 재사용할 수 있게 되었다는 점이다. 객체지향 개념은 원래 복잡한 소프트웨어 시스템이나 각종 설계를 관리하기 위하여 설계되었다. 캡슐화 및 계승을 통해 애트리뷰트(즉, 데이타베이스 설계)와 프로그램을 재사용하여 보다 복잡한 데이타베이스와 프로그램을 구축하는 발판이 되었다. 이것이 바로 과거 30년 동안 화일 시스템에서 관계형 데이타베이스에 이르기까지 자료관리 기술이 추구한 목표였다. OODB는 매우 복잡한 데이타베이스를 설계하고 관리하는 어려움을 줄일 수 있는 잠재성을 갖고 있다.

또 한 가지 원인은 캡슐화와 계승이라는 객체지향 개념에 내재되어 있는 강력한 자료형 기능이다. 이 자료형 기능이 실제로 다음과 같은 RDB의 중요한 세 가지 결점을 보완할 수 있는 열쇠이다. 이는 위에서 더 자세히 거론하겠다.

—RDB에서는 계층형 자료(hierarchical data), 복합중첩 자료(composite nested data) 등이 모두 릴레이션과 튜플로 표현되어야 한다. 게다가 여러 릴레이션에 걸쳐 있는 자료를 검색하기 위해서는 비용이 많이 드는 결합연산(join operation)을 수행해야 된다. OOPL의 한 애트리뷰트의 자료형은 시스템이 지원하는 원시적 자료형과 사용자가 정의한 자료형(클라스)을 가질 수도 있다. 이는 한 객체가 애트리뷰트의 값으로 다른 객체를 가질 수 있다는 뜻이며, 따라서 복합중첩 객체와 계층형 자료를 자연스럽게 표현할 수 있다.

—RDB는 릴레이션의 열(column)에 사용할 수 있는 자료형으로 시스템이 이미 정의한 원시적인 자료형만을 지원하고, 사용자가 새로이 정의한 자료형을 추가할 수 없다. 원시적인 자료형은 주로 숫자나 간

단한 기호들이다. RDB는 새로운 자료형의 추가에 대비하여 구축되어 있지 않기 때문에, 새로운 자료형을 추가하려면 시스템의 구조부터 수정해야 한다. 데이타베이스 시스템에 새로운 자료형을 추가한다는 의미는 이를 애트리뷰트의 자료형으로 쓸 수 있고, 이러한 자료를 저장, 검색, 변경할 수 있음을 의미한다. OOPL의 객체 캡슐화는 객체의 자료부분이 어떠한 자료형도 가질 수 있도록 허용한다. 더구나, 새로운 자료형이 새로운 클라스로서 생성될 수도 있으며, 심지어 이미 존재하는 클라스의 애트리뷰트와 메소드를 계승받아 하위클라스로서 생성될 수도 있다.

—객체 캡슐화는 데이타베이스의 자료뿐만 아니라 프로그램의 저장, 관리에도 기초가 된다. RDB도 요즈음에는 프로그램이 데이타베이스에 저장되었다가 후에 적재되어 수행될 수 있는 "저장 프로시듀어(stored procedure)"를 지원한다. 그러나, 이는 다른 일반 자료와 함께 캡슐화될 수 없다. 즉, 다른 릴레이션이나 튜플과 결합되지 않는다. 게다가, RDB는 계승 개념이 없어 한번 작성된 프로그램을 자동적으로 다시 사용할 수 없다.

III. 현존하는 객체지향 데이타베이스 제품의 상황과 향후 추세

현재 객체지향 데이타베이스 시스템으로는 Servio의 GemStone, ONTOS의 ONTOS, Object Design의 ObjectStore, Objectivity의 Objectivity/DB, Versant Object Technology의 Versant, Object Database의 Object Database, Itasca의 Itasca, O2 Technology의 O2 등이 있다. 이들은 모두 객체지향 모델을 지원한다. 이들이 사용자에게 제공하고 있는 기능으로는 애트리뷰트와 메소드를 갖는 새로운 클라스의 생성, 상위 클라스로부터 애트리뷰트와 메소드의 계승, 각 클라스의 인스턴스 생성, 객체 식별자를 이용한 인스턴스의 추출, 메소드의 수행 등이 있다.

이들 객체지향 데이타베이스 시스템들은 1987년초부터 판매되기 시작했다. 그러나, 이들 중 대부분은 아직 평가중이거나 시제품의 수준이다. 즉 아직 중대한 응용분야에 실각하게 사용된 적이 없다. 게다가, 꽤 많은 제품이 인위적으로 제품설치 건수를 올리기 위해 무상으로 보급되기도 했다. 과거 5년 동안은 일반적인 객체지향 기술, 특히 객체지향 데이타베이스 기술이 태동하는 시기라 할 수 있다. 그러나, OODB가 성숙하지 않은 관계로 아직 중대한 응용분야에서는 적용되지 않고 있다.

3.1 한계

3.1.1 지속성(persistency)을 지원하는 저장 시스템으로서의 한계

현재 객체지향 데이타베이스 시스템의 중요한 목적 중 하나는 프로그래밍을 할 수 있고 동시에 데이타베이스를 관리할 수 있는 통합된 언어(예를 들어, C++, Smaltalk)를 지원하는 것이다. 이 목적은 응용프로그램이 COBOL, FORTRAN, PL/1, C 등과 같은 범용 프로그래밍 언어와 여기에 SQL과 같은 데이타베이스 언어를 결합하여 작성되는 현재 상태에서 기인한다. 범용 프로그래밍 언어와 데이타베이스 언어는 문법이나 자료 모델 측면에서 크게 다르다. 또한 응용프로그램을 작성하기 위해 이렇게 서로 크게 다른 두 가지의 언어를 습득하고 사용한다는 것은 상당한 부담이 된다. C++나 Smaltalk은 이미 클라스와 클라스 계층을 정의하는 기능을 지원한다. 실제로 이들은 범용 프로그래밍 언어와 데이타베이스 언어를 통합하는 기초로 사용될 수 있다. 초기의 대부분의 OODB는 첫 단계로 클라스나 클라스의 인스턴스에 지속성을 지원하려고 했다. 즉, 클라스나 인스턴스를 보조 기억장치에 저장함으로써 클라스를 정의하고 생성한 프로그램이 종료되더라도 이들을 사용할 수 있도록 했다.

현존하는 대부분의 OODB는 RDB에 비교할 만한 복잡한 질의 기능을 제공하지 못한다. 이들은 단지 객체를 검색하는 단순한 방법만을 지원한다. 또한 객체지향 언어에 지속적인 저장 기능을 제공하는 객체지향 데이타베이스 시스템은 자료에 여러가지 제약을 주기도 한다. 대부분의 시스템이 비지속적인 자료가 OID를 가지지 못하게 함으로써 지속적인 자료와 비지속적인 자료를 구별해야 하며, 이 때문에 모든 객체를 지속적인 자료인지 비지속적인 자료인지를 구별하여 선언해야 된다.

3.1.2 데이타베이스 시스템으로서의 한계

현존하는 OODB의 또 다른 문제점은 일반 데이타베이스 시스템의 사용자들이 많이 사용하는 기능이 결여되어 있다는 점이다. 즉, 비절차적인(non-procedural) 질의어, 자동적인 질의처리와 최적화, 동시성 제어, 권한 부여, 동적인 스키마 변경, 시스템 데이타 구조의 변경 등의 기능이 충분히 제공되지 못하고

있다.

－현존하는 대부분 OODB는 중첩 질의(nested query), 집합 질의 연산(예를 들어, union, intersection, difference), 누적 함수(예를 들어, MAX, MIN, AVG)와, GROUP BY 기능, 그리고 클라스들 간의 결합 등과 같은 RDB의 기능을 지원하지 못한다. 현재의 OODB는 데이타베이스의 스키마를 생성하고 데이타베이스에 인스턴스를 저장하는 기능은 제공하지만, 데이타베이스에서 객체를 추출하거나 여러 사용자가 객체를 공유하도록 하는 기법들은 지원하지 못하고 있다.

－RDB에서는 질의를 처리할 때 자동적으로 로크(lock)을 걸지만 OODB에서는 사용자가 명시적으로 잠금을 요구해야 하는 경우도 있다.

－RDB는 생성한 자료나 스키마를 다른 사용자가 사용하거나 변경할 수 있도록 하는 권한 부여 기능을 제공하지만 대부분의 OODB는 그렇지 못하다.

－RDB는 릴레이션에 새로운 애트리뷰트를 추가하거나 기존의 릴레이션을 삭제하는 등의 데이타베이스 스키마에 관련된 기능을 제공한다. 그러나 OODB는 새로운 클라스의 추가를 허용하면서, 이미 존재하는 클라스에 새로운 애트리뷰트나 메소드의 추가, 새로운 상위 클라스의 추가 및 삭제, 클라스의 삭제 등과 같은 데이타베이스 스키마 변경을 지원하지 않는다.

－RDB에서는 시스템 관리자가 시스템 인자를 변경함으로써 성능을 조정할 수 있다. 이와 같은 시스템 인자로는 메모리 버퍼의 크기, 자료의 추가를 위한 잉여 공간의 크기 등을 들 수 있다. 그러나 대부분의 OODB는 이와 같은 시스템 성능조정을 위한 충분한 기능을 제공하지 못한다.

3.2 향후 개선되어야 할 점

이상과 같이 열거한 문제점 때문에 현존하는 OODB는 RDB의 기능을 지원할 수 있도록 수정되어야 한다. 그러나, 현재 중대한 응용 분야에서 사용중인 데이타베이스 시스템 정도의 기능을 지원하도록 OODB를 확장하는 것은 향후 3～4년 내에는 가능하지 않을 것으로 전망된다.

완전한 질의 기능을 위해서 OODB가 갖추어야 할 기능은 다음과 같다.

1. OODB에 사용할 수 있는 질의어를 설계해야 한다

2. 질의어를 처리하기 위해 구문 처리기(parser)를 구현해야 한다.

3. 데이타베이스 시스템의 중요한 부분 가운데 하나인 질의 최적기를 추가해야 한다. 현존하는 대부분의 OODB는 단순한 형태의 질의어만을 제공하며, 질의어를 처리하기 위한 최적의 방법을 찾는 질의 최적기를 갖고 있지 않다.

4. 결합, 집합 질의, 중첩 질의 등과 같은 복잡한 질의를 처리하기 위해서 정렬병합결합(sort-merge-join), 질의 평가, 중첩 질의 처리 등과 같은 다양한 알고리즘을 구현해야 한다.

5. 질의 처리를 위하여 정렬 패키지, B^+-트리 인덱스 패키지 등과 같이 데이타베이스에 자료를 효율적으로 저장 또는 추출할 수 있는 접근 기법을 구현해야 한다.

6. 데이타베이스에 대한 통계를 제공하는 카탈로그(catalog)를 개발해야 하며, 질의 처리를 위해서 이러한 통계 자료를 관리하는 카탈로그 관리기를 구현해야 한다. 이 카탈로그는 클라스 내 객체의 갯수, 애트리뷰트의 최대값과 최소값, 클라스 내 서로 다른 값의 갯수 등을 저장하여야 한다. 또한, 이들 외에도 클라스의 객체가 차지하는 자료 페이지의 갯수, 클라스의 애트리뷰트에 설치된 인덱스의 종류 및 크기 등과 같은 정보도 질의 처리기를 위하여 적당한 시스템 카탈로그에 저장해야 한다.

7. 다수의 사용자가 동시에 데이타베이스를 사용하거나, 시스템의 고장 또는 트랜잭션의 철회가 발생하는 경우에도 시스템의 자료구조 및 데이타베이스를 일관성있게 유지할 수 있도록 자료접근 기법을 설계하여야 한다. 즉, 접근 기법은 데이타베이스 시스템이 동시성 제어 및 회복 기법과 통합되어 구현되어야 한다.

8. 질의 처리 알고리즘도 다수의 사용자가 동시에 데이타베이스를 사용하거나, 시스템의 고장 또는 트랜잭션의 철회가 발생하는 경우, 시스템의 자료구조 및 데이타베이스를 일관성있게 유지할 수 있도록 설계하여야 한다. 즉, 질의 처리 알고리즘은 데이타베이스 시스템의 동시성 제어 및 회복 기법과 통합되어 구현되어야 한다.

9. 질의 처리를 위해 작업영역 관리기가 수정되어야 한다. 작업영역은 사용자가 필요로 하는 객체를 주기억 장치에 관리하기 때문에 객체가 수정되면, 데이타베이스에 저장된 값과 서로 다를 수 있다. 따라서, 질의 처리기는 작업영역에서 수정된 객체를 고려해야 한다. 예들들어, 작업 영역에서 수정된 모든 객체를

질의를 처리하기 전에 데이타베이스에 반영한다면 질의 처리기는 가장 최근의 값을 사용할 수 있을 것이다.

사용자가 스키마를 변경할 수 있도록 하기 위해서는 다음과 같은 기능이 필요하다.

1. 스키마를 동적으로 변경시킬 수 있는 기능을 설계해야 한다.

2. 스키마 관리기를 시스템에 추가하고 다른 관리기들과 통합할 필요가 있다. 스키마 관리기는 데이타베이스 스키마를 접근하고 변경할 수 있어야 한다.

3. 작업영역 관리기를 주기억 장치의 작업영역에서 처리되는 객체에 대한 스키마를 변경할 수 있도록 확장해야 한다. 예를 들어, 어떤 클라스에서 한 에트리뷰트를 삭제하면 작업 관리기는 작업영역 내의 객체에 대해 해당 애트리뷰트를 삭제된 것으로 표시해 두거나, 작업영역의 모든 객체를 데이타베이스에 저장한 다음 삭제된 애트리뷰트를 제외하고 다시 작업영역으로 가져와야 한다.

4. 하나의 클라스가 상위 클라스로부터 애트리뷰트와 메소드를 계승받기 때문에 그 상위 클라스의 스키마 변경도 계승되어야 한다. 예를 들어, 상위 클라스에서 한 애트리뷰트가 삭제되면 그 애트리뷰트는 모든 하위 클라스에서도 삭제되어야 한다. 즉, 클라스의 변경을 위해서는 클라스 자신뿐 아니라 하위 클라스까지도 로크해야됨을 의미한다. 따라서 스키마 관리자는 변경된 스키마를 계승하기 위해서 동시성 제어 기능도 포함해야 한다.

OODB의 개발은 기술적인 어려움뿐 아니라 객체지향 개념에서 유래된 어려움도 있다. 대부분의 OODB는 데이타베이스 시스템이라기 보다는 객체지향 프로그래밍 언어에 지속성을 지원하는 저장 시스템이라고 할 수 있다. 이들은 객체 지향 프로그램에서 생성된 자료를 관리하기 위해서 개발된 것이다. 그러나, 지난 20년 동안 데이타베이스 시스템의 사용자들은 데이타베이스 시스템을 매우 다양한 기능을 수행하는 소프트웨어로 인식하고 있다. 즉, 데이타베이스 시스템은 대량의 데이타베이스에서 소량의 유용한 자료를 검색하고, 이를 위한 질의 처리 기능을 갖고 있으며, 다수의 사용자가 동시에 검색 및 변경할 수 있고, 시스템 고장으로부터 데이타베이스의 일관성을 유지시켜 준다. 또한, 다수의 사용자 환경에서 데이타베이스 일부에 대한 사용 권한을 제어할 수 있고 시스템의 인자를 통해 환경에 맞게 성능을 조절할 수 있다. 이러한 이유 때문에 현존하는 OODB에 OODB라는 이름은 잘못 붙여진 것이다.

현존하는 OODB의 데이타베이스 언어로는 데이타베이스를 위한 내장 함수를 포함하도록 확장시킨 객체지향 프로그래밍 언어가 사용되고 있다. 이와 같은 함수들은 적당한 입출력을 위한 인수와 함께 응용 프로그램에서 호출되며, 호출하는 문법은 응용 프로그래밍 언어와 동일하다. OODB가 일반 데이타베이스 시스템의 기능을 완전하게 제공하기 위해서는 데이타베이스를 위한 내장 함수에 질의 처리를 위한 함수도 포함되어야 한다. 그러나 범용 프로그래밍 언어는 데이타베이스 질의어가 포함될 수 있도록 설계되어 있지 않다. 데이타베이스 질의의 결과에 포함되는 레코드나 객체의 갯수를 미리 예측하기 힘들기 때문에 응용 프로그램은 더 이상 만족하는 레코드나 객체가 없을때까지 검색하도록 설계되어야 한다. 따라서, 응용 프로그램에 데이타베이스 시스템의 커서(cursor) 기능을 도입해야 하며, 예측할 수 없는 갯수의 결과를 저장하기 위한 자료 구조와 알고리즘이 필요하다. 또한, 중첩 질의, GROUP BY, 누적 합수, 집합 질의 등을 표현하기 위한 기능을 응용 프로그래밍 언어의 문법과 동일하도록 제공해야 한다. 아울러 프로그래밍 언어로서 사용되는 주 프로그래밍 언어(host programing language)의 문법은 일반적으로 절차적 언어 수준에 있기 때문에 일반 사용자가 습득하기에는 상당한 어려움이 있다. 그러므로, 응용 프로그래밍 언어와 데이타베이스 언어를 동일한 언어로 사용하는 것이 주 프로그래밍 언어에 데이타베이스 언어를 포함시키는 것보다 항상 좋다고 할 수 없다.

IV. OODB에 관한 잘못 인식된 장점

OODB에 관한 여러가지 장점이 있다. 그러나 이들 가운데 많은 것은 전혀 장점이 없으며, 현재 대부분의 OODB가 RDB에 비해 완전한 데이타베이스 기능을 갖추지도 않았으면서 "데이타베이스 시스템"이라는 불행한 꼬리표를 달고 있는 것에 기인한다. 몇몇 장점은 기술의 진화적 속성에서 발생하는 자연적인 결과이다. 그리고 나머지는 아직 실제적으로 유용하지 않은, 순수 객체 추종자들의 관심을 표명하고 있다.

OODB가 RDB에 비해 10에서 1000배 빠르다. : OODB 판매자들은 성능을 나타내는 수치를 기반으로 OODB가 RDB에 비해 10에서 1000배 빠르다고 주장한다. 이러한 주장은 주의깊게 검증을 하지 않으면

잘못 이해할 수 있다 OODB는 RDB에 대한 성능 향상을 얻을 수 있는 두 가지 근거를 가지고 있다. OODB에서는 객체 X의 애트리뷰트가 다른 객체 Y의 객체 식별자(OID)를 가질 수 있다. 따라서 응용 프로그램이 객체 X를 검색한 후 객체 Y를 검색하고 싶으면 데이타베이스 시스템은 객체 X의 애트리뷰트인 OID를 통해 바로 검색할 수 있다. 그림 2a는 클라스 Person의 두 인스턴스와 클라스 Company의 두 인스턴스를 나타내며 클라스 Person의 한 애트리뷰트인 Workfor 애트리뷰트에 저장된 값은 객체인 클라스 Company의 OID이다. 만약 OID가 객체의 물리적 주소라면 객체를 데이타베이스로부터 직접 호출할 수 있다. 또한 논리적인 주소라면 해쉬 테이블 내용(OID를 물리적 주소로 사상할 수 있는 해쉬 테이블을 시스템이 유지한다고 가정하면)을 검색함으로써 객체를 호출 할 수 있다.

현재 RDB는 릴레이션의 애트리뷰트 도메인으로 단지 원시적인 자료형 밖에 허용하지 않는다. 즉, 튜플의 애트리뷰트 값은 숫자와 스트링과 같은 원시적 자료형만 가능하며 다른 형태는 허용되지 않는다. 만약 릴레이션 R2의 튜플 Y가 논리적으로 R1의 튜플 X의 애트리뷰트 A의 값이라면 튜플 X의 애트리뷰트 A에 실제 저장된 값은 릴레이션 R2의 튜플 Y의 애트리뷰트 B의 값이다. 응용프로그램이 튜플 X를 검색한 후 튜플 Y를 검색하고자 한다면, 시스템은 튜플 X의 애트리뷰트 A의 값을 이용하여 릴레이션 R2를 검색하는 질의어를 효과적으로 수행해야 할 것이다. 그림 2b는 OODB를 나타내는 그림 2a에 대한 RDB의 동등한 표현이다. 릴레이션 Person의 애트리뷰트 도메인은 원시적 자료형인 String이다. 만약 응용프로그램이 "John"을 갖는 Person튜플을 검색한 후, "UniSQL"을 갖는 Company튜플을 검색하고 싶다면 Company 릴레이션 전체를 검색하는 질의를 이용할 필요가 있다. 만약 Company 릴레이션이 수천 혹은 수만 개의 튜플을 가지고 있고 릴레이션 R2(Company)의 애트리뷰트 B(Name)에 색인이 없다면 튜플 Y를 찾기 위해 전체 릴레이션 R2를 순차적으로 검색하여야 한다. 애트리뷰트 B에 대한 색인이 있다고 하더라도, 해쉬 테이블을 이용한 OODB 정도의 효과는 얻을 수 있으나 물리적 주소로 OID를 구현한 OODB보다는 빠르지 못할 것이다(해쉬테이블 검색을 필요로 하지 않기 때문에).

OODB가 RDB에 비해 성능 향상을 얻는 두번째 근거는 객체가 주기억장치로 적재될 때 대부분의

Person

oid	name	age	salary	worksfor
115	John	25	25000	002
267	Chen	30	25000	001

Company

oid	name	age	president	location
001	Acme	15	Cohen	NY
002	UniSQL	3	Kim	Austin

(그림 2a) OODB의 객체 표현

Person

name	age	salary	worksfor
John	25	25000	UniSQL
Chen	30	25000	Acme

Company

name	age	president	location
Acme	15	Cohen	NY
UniSQL	3	Kim	Austin

(그림 2b) RDB2.의 튜플 표현

Person

oid	name	age	salary	worksfor
115	John	25	25000	002
267	Chen	30	25000	001

Company

oid	name	age	president	location
001	Acme	15	Cohen	NY
002	UniSQL	3	Kim	Austin

(그림 3a) 데이타베이스에서의 객체표현

Person

	name	age	salary	worksfor
040	John	25	25000	020
080	Chen	30	25000	004

Company

	name	age	president	location
004	Acme	15	Cohen	NY
020	UniSQL	3	Kim	Austin

(그림 3b) 메모리에서의 객체 표현

OODB는 객체 내의 OID를 주기억장치 포인터로 변환한다는 것이다. 만약 객체 X와 Y가 주기억장치로 적재된다면 객체 X의 애트리뷰트 A 값으로 저장된 OID는 주기억장치에서 객체 Y를 가르키는 가상 주기억장치 포인터로 변환된다. 그러므로 객체 X에서 객체 Y로의 탐색은 즉, 객체 X의 애트리뷰트 A값을 통한 객체 Y의 접근은 기본적으로 주기억장치 포인터에 의한 것이고, 그림 3a는 클라스 Person과 Company의 객체를 데이타베이스 표현 형태로 나타낸 것이고, 그림 3b는 이를 주기억장치 표현 형태로 나타낸 것이다. Person 객체의 Worksfor 애트리뷰트 내에 저장된 OID들은 주기억장치 주소로 변환된다. 수백 개 아니 수천 개의 객체를 주기억장치로 적재하고, 그리고 주기억장치 내에서 각 객체가 다른 객체 하나 이상을 가르키는 주기억 장치 포인터들을 갖고 있다고 하자. 더우기 한 객체에서 다른 객체로의 탐색 항해를 반복적으로 수행한다면, RDB는 OID들을 저장하지 못한다 그들은 하나의 튜플 내에 다른 튜플의 주기

억장치 포인터를 저장할 수 없다. 주기억장치 상주 객체를 통한 탐색항해 기능은 근본적으로 RDB에는 없는 특성이다. 그로 인한 성능 저하는 주기억장치 내에 대용량의 버퍼 공간을 갖는다고 단순히 해결될 수 없다. 그러므로 주기억장치 내에 적재된 연결 객체를 통한 반복되는 탐색항해를 요구하는 응용 프로그램에 대해서는 OODB가 RDB를 단연 앞지를 수 있다.

만약 모든 데이타베이스 응용이 단지 주기억장치 내에서 객체간에 연결된 포인터를 따라 검색하거나 객체의 OID 검색만을 요구한다면 OODB가 RDB에 비해 수백 내지 수천 배 빠른 것은 열마든지 가능하다. 그러나 OID의 검색을 이용한 대부분의 응용은 RDB가 이미 잘 지원하는 데이타베이스 접근과 갱신을 또한 요구한다. 이러한 요구는 대량 데이타베이스 적재와 생성, 갱신, 객체의 제거(한번에 하나씩), 임의의 클라스로부터 검색 조건을 만족하는 객체의 검색, 한 클라스 이상의 결합(나중에 간단히 살펴봄), 트랜잭션 완료 등을 포함한다. 그러한 응용에 대해 OODB는 어떤 성능 잇점도 제공하지 못한다. 사실, 그림 2의 예제 데이타베이스에서 보면 만약 응용 프로그램의 목적이 주어진 Person객체에 대한 특정한 Company 객체를 찾는 것이 아니라(즉, 단순한 탐색항해) 어떤 조건을 만족하는 Company 객체와 연관된 Person 객체를 찾는 것이라면 (예 : 연봉이 40000보다 작은 25세 이상의 모든 사람을 찾아라-즉, 일반적인 질의) OODB는 OID가 어떻게 구현되어 있는지 그리고 질의어 최적기가 질의를 처리하는 과정에서 OID의 장점을 이용하도록 설계되어 있는지에 따라 아무런 성능 이익도 얻지 못할 수도 있다.

OODB는 데이타베이스 언어가 필요없다. : OODB는 OOPL이 아닌 별도의 데이타베이스 언어가 필요없다는 잘못된 인식이 있다. 이러한 잘못된 인식은 현재의 OODB가 완전한 데이타베이스 시스템이 아니기 때문에, 그리고 데이타베이스에서 제공되는 기능이 함수의 호출로서 표현되었기 때문에 발생하였다. 현재의 대부분의 OODB를 진정한 의미의 데이타베이스 시스템으로 향상시키고, 특히 RDB가 제공하는 모든 질의 기능을 지원하기 위해서는 비절차 질의 언어의 지원이 꼭 필요할 것이며, 이는 필수 불가결한 것이다.

OODB에서는 결합이 필요없다. : RDB의 릴레이션의 결합과 비교할 때 OODB에서는 클라스의 결합이 많이 필요하지 않다. 그러나 필요성을 완전히 제거하지는 못한다. OODB에서 클라스 C의 애트리뷰트의 도메인은 다른 클라스가 될 수 있다. 그러나 RDB에서 릴레이션 R1의 애트리뷰트 도메인은 다른 릴레이션 R2가 될 수 없다. 그러므로 RDB에서는 사용자가 한 릴레이션의 튜플을 다른 릴레이션의 튜플과 서로 연관시키기 위해서 항상 명시적으로 두 릴레이션에 대해 결합 연산을 수행해야 한다. OODB는 클라스의 애트리뷰트 값으로 저장된 클라스의 객체 OID를 통함으로써 명시적 결합을 암시적 결합으로 대치한다. 그림 2의 예가 이를 보여준다. OODB에서 클라스 D를 클라스 C의 애트리뷰트 도메인으로 명시하면 이는 곧 클라스 C와 D의 결합을 정적으로 표시한 것이 된다.

관계형 결합은 두 릴레이션에서 서로 대응된 애트리뷰트 쌍의 값을 기반으로 릴레이션을 서로 관계시키는 일반적인 방식이다. OODB에서도 일반적으로 두 클라스가 대응된 애트리뷰트 쌍을 가질 수 있으므로 관계형 결합은 여전히 유용하다. 예를 들면 그림 2에서 클라스 Person과 Company가 동시에 애트리뷰트 Name, Age를 가지고 있다. 비록 클라스 Company의 애트리뷰트 Name과 Age의 도메인이 클라스 Person의 애트리뷰트 Name, Age의 도메인과 다르지만 사용자는 이러한 애트리뷰트 값을 기반으로 두 클라스를 서로 관련시키고자 할 수 있다(예, 자신이 일하는 회사의 나이보다 적은 나이를 가진 사람을 찾아라).

OID는 키의 필요성을 없앤다. : OID는 자신이 갖는 장점보다 더 많은 관심을 받아왔다. OID는 단지 객체를 표현하는 수단이며, 개개의 객체의 유일성을 보장한다는 것외에는 더 이상의 의미를 갖지 않는다. 비록 OID가 각 객체에게 유일성을 주지만, 이는 시스템에 의해 자동적으로 주어지며 사용자에게는 보이지 않는다. 그러므로 대량의 데이타베이스로부터 특정 객체를 찾을 수 있는 편리한 수단을 제공하지는 않는다(즉, 사용자가 특정 객체의 OID를 모를 때). 사용자가 자신이 정의한 키를 사용하여 객체를 찾을 수 있다면 더욱 편리할 것이다. 예를 들면 그림 2의 예제 데이타베이스에서 만약 Name 애트리뷰트가 키라면 사용자는 특정 이름을 검색하는 질의어를 통해 한 Person객체를 찾을 수 있을 것이다.

OODB는 비절차 언어를 가질 수 없다. : 이는 현재 대부분의 OODB가 제한된 질의 기능을 제공하는데 기인하는 것이다. OODB의 판매자들은 데이타베이스 탐색항해 성능과 객체의 영속성에 개발 노력을 집중

한다. 최근의 일반적으로 Object SQL이라고 이름지워진 비절차 언어의 제안은 객체 지향 파라다임과 비절차 질의어 사이의 기본적인 이론의 일치가 있다는 것을 보여준다.

질의 처리는 캡슐화를 위반한다. : OOPL에서 자료와 프로그램을 캡슐화하는 한 가지 목적은 프로그래머가 객체에 접근하기 위해 그 객체의 프로그램 부분만 사용하도록 하여 프로그래머가 객체를 저장하는데 필요한 자료구조와 프로그램 부분의 구현에 대한 자세한 내용을 알 필요 없도록 하는 것이다. 질의어를 처리하는 과정에서 데이타베이스 시스템은 객체의 내용을 읽어야만 하며 객체의 애트리뷰트에 저장되어 있는 OID를 얻어 그 OID에 대응되는 객체를 검색해야 한다. 이때 데이타베이스 시스템이 객체의 내용을 조사하기 때문에 순수 객체 추종자들은 이것을 객체 캡슐화의 위반이라 한다. 이러한 관점은 실제적이지도 않으며 유용하지도 않다. 첫째, 각 객체의 내용을 조사하는 것은 일반 사용자가 아니라 데이타베이스 시스템이며, 둘째, 객체의 애트리뷰트에 저장된 값을 조사하는 행위는 모든 클라스의 모든 애트리뷰트에 암시적으로 결부된 "get(또는 read)" 메소드를 통해서라고 간주될 수 있기 때문이다. 만약 객체의 순수성이 항상 지켜져야 한다면 사용되는 모든 개개의 숫자나 문자 상수에 반드시 명시적으로 하나의 OID가 할당되어야 한다. 그러나, 알려진 OOPL이나 OO 응용중 어느 것도 이것을 지키지 않는다.

OODB는 단지 과거의 계층적 혹은 망 데이타베이스 시스템의 재생이다. : OODB는 계층적 자료를 자연스럽게 표현한다. 그리고 이러한 자료는 탐색항해를 통해 검색된다. 이러한 OODB의 특성은 계층형 데이타베이스 시스템(예 : IMS)과 망 데이타베이스 시스템(예 : TOTAL)이 갖는 특성이다. 그러나, 계층형 그리고 망 데이타베이스 시스템은 캡슐화와 계승을 이해하지 못한다. 캡슐화와 계승은 OODB에서 중요한 역할을 하기 때문에 OODB를 계층형 혹은 망 데이타베이스 시스템과 동등하게 취급해서는 안된다. 보다 정확히 말하면 OODB가 잠재적으로 계층형 그리고 망 데이타베이스 시스템을 포함한다고 할 수 있다.

OODB는 버전과 장기 트랜잭션을 제공할 수 있다. : OODB가 버전과 장기 트랜잭션을 어떻게 제공할 수 있는지에 대해 일반적으로 잘못 인식하고 있다. RDB는 버전과 장기 트랜잭션을 암시적으로 지원할 수 없다. 비록 릴레이션에서 객체로 가는 파라다임의 변화가 RDB의 키의 결함을 해결하지만 버전과 장기 트랜잭션의 문제를 설명하지는 못한다. 객체 지향 파라다임도 관계형 자료 모델과 마찬가지로 버전과 장기 트랜잭션을 포함하지 않는다. 간단히 말하면 C++이나 Smaltalk에는 어떠한 버전 기능이나 장기 트랜잭션 기능이 포함되어 있지 않다.

버전과 장기 트랜잭션이 OODB와 연관되어 왔던 이유는 단순히 그들이 RDB에서는 빠져있던 데이타베이스 기능이며, OODB가 강력한 자료 모델 기능과 객체 탐색항해 기능을 통해 RDB보다 훨씬 잘 지원할 수 있는 응용(예 : computer-aided engineering system, computer aided authoring system, 등)의 요구사항의 하나로 인식되어 왔기 때문이다. 사실 대부분의 OODB는 버전과 장기 트랜잭션을 아예 제공하지도 않는다. 버전과 장기 트랜잭션을 제공하는 몇몇 OODB도 단지 원시적인 기능만을 지원한다.

버전과 장기 트랜잭션을 지원하는 난이도는 OODB나 RDB나 똑같다. 먼저 버전 측면을 살펴보자. 객체가 버전화되기 위해서는 시간표지(timestamp) 또는 버전 식별자를 유지해야 된다. 이것은 시스템이 정의한 애트리뷰트를 새롭게 정의함으로써 구현할 수 있다. 이는 OODB에서는 클라스의 버전화된 객체에, RDB에서 릴레이션의 버전화된 튜블 각각에 대해 적용할 수 있다. 더우기 버전유도, 버전 제거, 버전 검색 등의 기능은 OODB와 RDB의 데이타베이스 언어를 확장함으로써 표현할 수 있다.

다음은 장기 트랜잭션을 고려해 보자. 트랜잭션이란 간단히 말하면 하나의 단위로 취급되는 데이타베이스 연산의 집합이다. RDB는 트랜잭션이 데이타베이스와 단지 몇분 몇초 동안 수행한다고 가정하였다. 이런 가정은 무효가 되었으며, 사용자가 데이타베이스를 오랫동안(몇시간 아니면 몇일) 접근하는 환경에서 장기 트랜잭션이 필수적으로 되었다. 트랜잭션의 수행시간과 관계없이 트랜잭션은 시스템 오류가 존재하고 여러 사용자가 데이타베이스를 동시에 접근할 때 데이타베이스의 일관성을 유지하는 기법이다. OODB와 RDB가 다른 점은 자료 모델이다. 즉 자료가 어떻게 표현되느냐 하는 것이다(즉, OODB에서는 애트리뷰트, 메소드, 클라스, 클라스계층, RDB에서는 애트리뷰트와 릴레이션). RBD와 OODB의 파라다임의 차이가 트랜잭션의 설계 문제를 해결할 수 없다는 것은 분명하다.

OODB는 다중매체 자료를 지원할 수 있다. : OODB는 다중매체 자료를 관리하는 데 필요한 함수를 구현할 수 있다. 다중매체 자료는 임의의 자료 형태

(숫자, 짧은 스트링, 고용인, 회사, 이미지, 오디오, 텍스트, 그래픽스, 영화, 이미지와 텍스트를 포함하는 문서 등)와 임의의 크기(1 바이트, 10 K 바이트, 1 기가 바이트 등)로 광범위하게 정의된다. OODB는 다중매체 자료를 관리하는 첫번째 요구사항인 임의의 자료형을 만들고 사용할 수 있도록 하여 준다.

그러나, 객체지향 파라다임(즉, 캡슐화, 계승, 메소드, 임의의 자료형-집합적으로 또는 개별적으로)은 매우 큰 다중매체 객체(예. 이미지, 오디오, 텍스트 문서, 영화 등)를 저장하고 검색, 갱신하는 문제를 해결하지는 않는다. RDB가 매우 큰 객체의 검색(일반적으로 페이지 버커가 객체 전체를 저장할 수 없다), 부분 갱신(한 객체의 조그만한 갱신 때문에 객체전체가 다시 복사되어서는 안된다), 동시성 제어(여러 사용자가 동시에 동일한 객체를 접근할 수 있어야 한다), 회복(로깅할 때 객체 전체를 복사해서는 안된다)을 포함하여, 릴레이션 내의 한 열의 도메인 BLOB (binary large object)이 될 수 있도록 해야 하듯이 OODB도 이와 똑같은 공학적인 문제들을 해결해야 한다.

OODB에는 이론적인 기초가 없다. : 나중에 기술하겠지만, 객체 지향 자료 모델은 확장된 관계형 자료 모델의 한 유형이다. 객체 지향 자료 모델은 다음과 같이 기존의 관계형 자료 모델을 확장한다. 확장의 시작점은 바로 릴레이션과 클라스의 유사성이다.

1. 릴레이션의 애트리뷰트 도메인이 시스템에 이미 정의된 원시적 자료형이 아닌 임의의 자료형으로 일반화된다. 이를 통해 클라스의 통합화(aggregation)계층과 계층의 인스턴스로서 중첩된 객체가 유도된다.

2. 메소드를 첨가함으로써 릴레이션의 특성을 애트리뷰트와 애트리뷰트에 대한 제약조건 범위 이상으로 확장한다.

3. 각 릴레이션이 시스템에서 정의한 객체 식별자 애트리뷰트를 갖게 된다. 객체 식별자는 도메인이 임의의 클라스인 애트리뷰트의 값으로 저장된다.

4. 클라스는 클라스들 사이의 일반화(generalization)/세분화(specialization) 관계를 통해 클라스 계층을 구성한다. 기존의 관계형 모델은 릴레이션을 계층을 조직하지는 않는다.

5. 클라스의 성질은 그것의 하위형(하위 클라스)에 의해 재귀적으로 계승될 수 있다. 하위 클라스는 계승된 성질들을 교정할 수 있으며 새로운 애트리뷰트와 메소드를 정의할 수도 있다.

6. 클라스의 애트리뷰트와 메소드는 그 클라스에 정의된 인터페이스를 이용한 메시지에 의해서만 접근될 수 있다. 어떤 클라스의 인스턴스에 보낸 메시지가 그 클라스에 정의되어 있지 않다면, 그것이 정의되어 있는 클라스를 찾기 위해 클라스 계층을 따라 위로 메시지를 보낸다. 이것을 객체 대한 메시지의 늦은 바인딩(binding)이라고 부른다.

위에서 알 수 있듯이 RDB의 많은 이론적 기초가 OODB에 사용될 수 있다는 것은 분명하다. 게다가 지난 15년간 통합화, 릴레이션의 프로시쥬어, 일반화, 대리(surrogate)와 같은 개념으로 관계형모델을 확장하려는 연구가 광범위하게 진행되어 왔다. 이러한 개념은 정도의 차이는 있으나 각각 중첩된 객체, 메소드, 계승, 객체 식별자 등에 해당한다.

메소드의 특성이나 다중 계승(multiple inheritance)에서의 충돌 해결과 같이 좀 더 많은 수식화가 필요한 영역이 있다. 그러나, 중요한 점은 RDB 조차도 E.F. Codd가 제안한 최소의 관계형 자료 모델과 관계해석(calculus), 그리고 관계 대수(algebra)를 기반으로하여 개발되었으나, 트랜잭션 관리, 동시성 제어, 회복, 뷰, 질의 최적화, 권한부여, 접근 방법의 이론은 Codd의 관계 대수나 관계 해석과 무관하게 개발되어야 했다.

V. 관계형과 객체지향 기법의 통합

오늘날 관계형 데이타베이스의 결함과 객체지향 데이타베이스에 대한 기대는 널리 알려져 있다. 그러나 객체지향 데이타베이스가 아직 데이타베이스 시장에 큰 영향을 미치지는 못하고 있다. 그 이유는 현재 대부분의 OODB가 데이타베이스 시스템으로서 성숙되지 않았기 때문이다(즉 이들은 RDB가 이미 지원하는 많은 주요 기능을 아직 지원하지 못한다). 또한 현재 객체지향 데이타베이스는 관계형 데이타베이스와 충분히 호환성을 갖지 못한다(즉 이들은 아직 관계형 데이타베이스 언어 SQL을 포함할 수 있는 언어를 지원하지 않는다). 객체지향 기법이 데이타베이스 기술에 획기적인 도약을 가져올 수 있으며 객체지향 데이타 모델을 포함하는 새로운 데이타베이스 시스템은 관계형 데이타베이스와 호환성이 있어야 한다(즉 SQL을 포함하여야 한다)는 것이 일반적으로 일치된 의견이다.

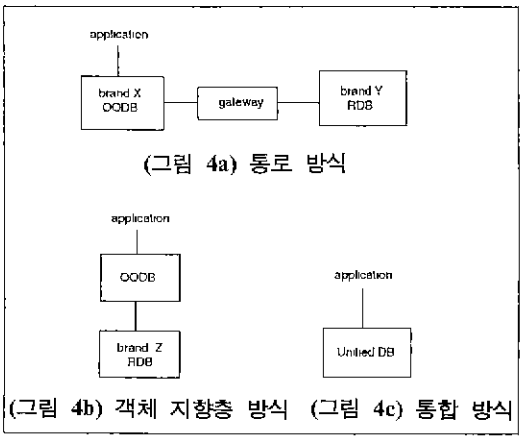
5.1 통합 구조

일부 객체지향 데이타베이스 개발회사들이 일반적

으로 Object SQL이라는 이름의 "SQL-같은" 언어(SQL을 포함하는 언어일 필요는 없다)를 준비하고 있다. Object SQL은 질의 기능을 전혀 제공하지 않거나 단순한 질의만을 제공하던 기존의 객체지향 데이타베이스 위에 데이타베이스를 정의하고 이에 대하여 질의를 수행할 수 있는 기능을 덧붙여 지원한다. 이것은 상품 전략에 대한 주요 방향 전환을 나타낸다. 불과 몇년 전까지만 해도 이들 회사들은 그들의 객체지향 데이타베이스와 일부 관계형 데이타베이스 사이의 통로(gateway)만을 제공하려고 하였다. 최근에 SQL을 확장한 UniSQL의 UniSQL/X와 Hewlett Packard의 OpenODB가 데이타베이스 시장에 출현하였다. UniSQL/X는 단일 데이타베이스 시스템으로 밑바닥부터 새로이 구현된 반면 OpenODB는 HP의 관계형 데이타베이스인 AllBase 위에서 구현되었다(즉 OODB가 RDB의 상부에 위치한다).

OODB와 RDB를 결합하는 방법은 OODB와 RDB 사이에 통로(gateway)를 두는 방법, RDB 위에 객체지향 계층(OO-layer)을 두는 방법, OODB와 RDB를 하나로 통합하여 단일 시스템으로 구축하는 방법과 같이 크게 세 가지로 분류할 수 있다. 그림 4는 이들 세 가지 방법을 보인 것이다. 통로를 두는 방법에서 OODB는 사용자의 질의를 단순히 변환하여 RDB로 전송하고, RDB는 이를 수행하여 그 결과를 다시 OODB에 전송한다. RDB에서 볼 때 통로는 관계형 데이타베이스의 일반 사용자와 같다. 현재 구현된 통로 방식은 OODB에 대한 질의에 여러가지 제약을 갖고 있다. 즉, 검색만을 허용하거나, 여러 질의어를 갖는 트랜잭션을 지원하지 못하거나, 또는 RDB의 처리 능력에 비교할 때 단지 단순한 질의만을 허용한다. 통로 방식을 이용하여 응용 프로그램이 RDB와 OODB로부터 검색된 데이타를 모두 사용할 수는 있지만 객체지향과 관계형 기법을 통합하는 좋은 방법은 아니다. 통로를 이용하는 방법의 성능은 질의와 결과의 변환, RDB와의 추가적인 통신비용 때문에 받아들일 수 없다. 또한 응용 프로그램을 만드는 사람이나 일반 사용자들이 두 개의 다른 데이타베이스가 있다는 것을 알아야 하므로 사용면에서도 적합하지 않다.

HP의 OpenODB와 같이 객체지향 계층을 두는 방법에서는 사용자가 OODB 언어(OpenODB의 경우 ObjectSQL)를 이용하여 시스템을 접근한다. 객체지향 계층은 데이타베이스 언어에 나타난 모든 객체지향 측면을 하부의 RDB와 상호작용할 수 있는 동등한



(그림 4a) 통로 방식

(그림 4b) 객체 지향층 방식 (그림 4c) 통합 방식

관계형으로 변환한다. 예를 들어 객체지향 계층은 객체를 릴레이션의 튜플로 변환하고 객체의 OID를 만든다. 그리고 그 OID를 OODB가 이용할 수 있는 인터페이스를 이용하여 변환된 튜플의 한 애트리뷰트로서 RDB에 전달한다. 또한 한 객체의 OID는 RDB 인터페이스를 이용하여 RDB에 저장된 객체로 대응된다. RDB는 자료 관리 계층과 저장 관리 계층으로 구성된다. 자료 관리 계층은 SQL 문을 수행하며 저장 관리 계층은 자료를 데이타베이스에 대응시킨다. 객체지향 계층은 자료 관리 계층 인터페이스를 이용하거나(즉 SQL 문을 이용하여 RDB와 상호작용) 저장 관리 계층 인터페이스를 이용한다(즉 RDB의 하부 기능을 이용하여 RDB와 상호작용), 자료 관리 인터페이스는 저장 관리 인터페이스 보다 느리다. (OpenODB는 자료 관리 인터페이스를 이용한다). 이 방법은 하부의 RDB가 수정되지 않은 것을 가정하고 있기 때문에 복잡한 데이타베이스 기능이 요구될 때 심각한 성능 및 운영상의 문제를 발생시킬 수 있다. 예를 들어 동적 스키마 변환으로 인해 클라스 계층에 있는 많은 클라스들이 로크되어야 한다면 RDB에 클라스 계층을 한번에 잠금할 수 있는 기능이 없으므로 객체지향 계층은 한번에 한 클라스씩 로크를 얻거나(성능을 저하시키고 교착상태(deadlock)의 위험이 있다). 또는 데이타베이스 전체에 잠금을 걸어야 한다(다른 사용자가 데이타베이스를 전혀 사용하지 못할 가능성이 있다). 앞의 어느 방법도 바람직하지 못하다. 게다가 객체를 메모리에서 갱신하고 트랜잭션이 끝났을 때 자동으로 갱신된 객체를 데이타베이스에 기록하는 것을 RDB 인터페이스를 이용하면 개개의 객체를 한번에 하나씩 기록하여야 한다.

객체지향 계층 방법을 이용하는 근본적인 이유는 객체지향 계층을 현재 존재하는 다양한 RDB위에 이식할 수 있기 때문이다. 이러한 유연성은 성능 저하의 댓가로 얻어진다. 객체지향 계층 방법은 다양한 데이타베이스들을 하나의 데이타베이스로 보이게 하는 다중 데이타베이스 시스템(multidatabase system)의 기초이다. 다중 데이타베이스 시스템에서는 응용 프로그램이 OODB와 RDB에서 검색된 자료를 함께 이용할 수 있다. OpenODB는 현재 객체지향 계층을 RDB에만 연결할 수 있기 때문에 다중 데이타베이스 시스템은 아니다. 다중 데이타베이스 시스템에 대하여는 다음에 상세하게 다루도록 하겠다.

단일 시스템으로 통합하는 방법은 RDB의 저장 관리 계층과 자료 관리 계층에 필요한 모든 변환을 하여 OODB와 RDB를 하나의 계층으로 만드는 것이다. 이를 위하여 많은 기술적인 문제를 해결할 필요가 있었다.

-관계형과 객체지향 자료 모델을 하나의 통합된 모델로 결합하여야 한다.

-ANSI SQL 데이타베이스 언어를 포함할 수 있는 언어를 설계하여야 한다. 따라서 데이타베이스언어는 자료 정의, 질의(조인, 집합 연산, 등) 갱신 기능을 포함한다.

-데이타베이스 시스템은 데이타베이스 언어가 허용되는 모든 기능을 지원하여야 한다. 이러한 기능은 동적 스키마 변환, 자동 질의 최적화, 자동 질의 처리, 접근방법(B$^+$ 트리색인, 확장해쉬, 외부정렬을 포함), 동시성 제어, 소프트웨어와 하드웨어의 고장으로부터 복구, 트랜잭션 관리, 권한 부여 및 철회를 포함한다. 통합된 자료 모델의 기능이 다양하기 때문에 구현은 더욱 어렵다.

-통합된 시스템은 RDB에 대한 질의를 기존의 RDB와 비교할만하게 수행할 수 있어야 하고, 동시에 OODB에 대한 질의를 기존의 OODB와 비교할만하게 수행하여야 한다.

-RDB와 OODB의 통합과는 별도로 멀티미디어 자료 관리, 시간적(temporal) 자료 관리를 위한 해결책을 제공한다. 멀티미디어 자료 관리와 시간적 자료 관리는 RDB나 OODB에 별로 관련이 없으므로 여기에서는 다루지 않는다.

다음 절에서 관계형과 객체지향 자료 모델의 통합을 위하여 관계형 모델을 어떻게 확장할 수 있는지를 기술한다. 통합된 자료 모델에서 사용자는 복잡한 자료의 요구를 쉽고 자연스럽게 표현할 수 있다. 다음으로 데이타베이스를 접근하기 위한 질의와 자료 조작 언어에 대하여 설명한다. 그리고 나서 메모리 내의 객체에 대한 탐색 항해 지원과 OOPL을 지원하는 몇몇 OODB의 성능 특성에 대해 논의한다.

5.2 데이타 모델의 통합

관계형 데이타베이스는 릴레이션의 집합으로 구성되며 릴레이션은 행(튜플)과 열로 이루어진다. 릴레이션의 행/열 엔트리는 하나의 값을 가지며, 그 값은 시스템에서 정의한 자료형(예, 정수, 스트링, 실수, 날짜, 시간, 금액형)에 속한다. 그러한 값에 사용자는 무결성 제한 조건을 추가할 수 있다(예, 사원의 나이는 정수형으로 18과 65 사이의 값을 갖는다). 사용자는 릴레이션에서 어떤 조건을 만족하는 튜플을 검색하기 위하여 비순차적인 질의를 할 수 있다. 더우기 릴레이션들의 열의 값을 상호 비교하는 것을 기본으로 하는 결합 질의를 함으로써 두 개 이상의 릴레이션을 상호 관련시킬 수 있다.

이러한 간단한 자료 모델을 세 가지 방법으로 일반화 및 확장할 수 있다. 각각의 방법은 주요 객체지향 개념을 반영한다. 객체지향 시스템 또는 객체지향 프로그래밍 언어의 기본적인 원리는 객체의 값 뜨한 개체라는 것이다. 첫번째 확장은 릴레이션의 열값이 시스템에서 정의한 자료형(숫자, 스트링 등) 뿐만이 아닌 사용자가 정의한 릴레이션의 튜플이 될 수 있도록 허용함으로써 객체 지향 개념을 반영한다. 이는 사용자가 정의한 임의의 일레이션을 다른 릴레이션의 열값의 범주로 지정할 수 있음을 의미한다. 그림 5에서 첫번째 CREATE TABLE 문을 관계형 모델에서 Employee 릴레이션을 정의하는 것을 나타낸다. Hobby와 Manager 열의 값은 문자열로 제한되었다. 그림 5에서 두번째 CREATE TABLE은 릴레이션의 열에 대한 자료형의 확장을 반영한다. Hobby의 값은 더 이상 문자열로 제한되지 않고 사용자가 정의한 릴레이션 Activity의 튜플이 될 수 있다. 유사하게 Employee의 Manager 애트리뷰트의 자료형은 Employee 릴레이션 그 자체가 될 수 있다.

릴레이션의 열이 다른 릴레이션의 튜플(즉, 임의의 자료형)을 갖게 되면 자연스럽게 중첩 릴레이션이 발생한다. 다시 말하면 릴레이션의 행/열 엔트리의 값이 다른 릴레이션의 튜플을 가지며 그 튜플은 또한 순환적으로 또 다른 릴레이션의 튜플을 값으로 가질 수 있다. 그림 2에서 이렇게 개념적으로 간단한 확

장이 어떻게 자료검색의 성능향상을 가져오는지 이미 알아보았다. 이를 통해 데이타베이스 시스템은 멀티미디어 시스템(화상, 음성, 그래픽, 텍스트와 이러한 자료로 이루어진 복합문서를 관리한다), 과학 자료처리시스템(벡터, 행열 등을 다룬다), 공학 및 설계 시스템(복잡한 객체를 다룬다) 등과 같은 응용을 지원할 수 있는 잠재성을 갖는다. 이러한 이유가 오늘날의 프로그래밍 언어와 데이타베이스 시스템이 많은 자료형을 지원하는 근본이 되고 있다.

두번째 확장은 객체지향 개념인 캡슐화이다. 캡슐화는 자료를 조작하기 위한 프로그램과 자료를 결합하는 것이다. 이것은 한 릴레이션에 속하는 튜플의 열에 대한 연산을 수행하는 프로시듀어(procedure)를 그 릴레이션에 부착시킴으로써 가능하다. 그림 5의 세번째 CREATE TABLE 문은 주어진 사원의 퇴직금을 계산하는 RetirementBenefits 프로시듀어를 구체화한 PROCEDURE 절을 나타낸다. 각 열의 값을 읽고 갱신하는 프로시듀어는 묵시적으로 각 릴레이션에 부착되어 있다.

한 릴레이션은 이제 그 릴레이션의 튜플의 상태와 작동을 캡슐화한다. 상태는 열값들의 집합이고 동작은 그 열값에 대하여 연산하는 프로시듀어들의 집합이다. 사용자는 릴레이션의 튜플에 대한 연산을 수행하는 어떤 프로시듀어도 만들 수 있으며, 이들의 응용에는 가상적으로 제한이 없다.

세번째로, 객체지향 개념인 계승 계층의 개념을 지원한다. 사용자는 데이타베이스에 있는 모든 릴레이션을 계층으로 구성할 수 있다. 예를 들어 릴레이션 P와 C에 대해서 C가 C에 정의된 열과 프로시듀어 외에 P의 모든 열과 프러시듀어를 계승 받는다면 P는 C의 부모가 된다. 또한, 하나의 릴레이션이 두 개의 부모 릴레이션에서 열과 프로시듀어를 계승 받을 수 있다. 이를 다중계승(multiple inheritance)이라 한다. 릴레이션의 계층은 시스템에 정의된 루트를 갖는 비순환 방향 그래프(directed acyclic graph)이다. IS-A (일반화와 세분화) 관계는 자식과 그의 부모 릴레이션 사이에 성립된다. 그림 5의 네번째 CREATE TABLE 문에서 Employee 릴레이션은 다른 릴레이션 Person의 자식으로 정의된다. Employee 릴레이션은 자동으로 Person 릴레이션의 세 열을 계승 받는다. 즉, Employee 릴레이션은 자기에게 정의되지 않았지만 Name, SSN, Age열을 갖게 된다.

릴레이션 계층은 관계형 모델의 독립적인 릴레이션들의 집합에 비해 두 가지 장점이 있다. 첫째, 릴

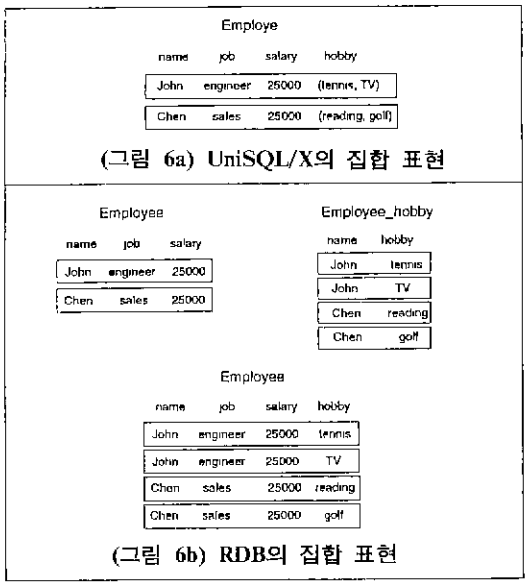
```
1 CREATE TABLE Employee
    (Name CHAR(20) Job CHAR(20) Salary FLOAT, Hobby CHAR(20), Manager CHAR(20)),

2 CREATE TABLE Employee
    (Name CHAR(20) Job CHAR(20), Salary FLOAT, HOBBY Activity, Manager Employee),

      CREATE TABLE Activity
          (Name CHAR(20), NumPlayers INTEGER, Origin CHAR(20)),

3 CREATE TABLE Employee
    (Name CHAR(20) Job CHAR(20) Salary FLOAT, HOBBY Activity, Manager Employee)
     PROCEDURE RetirementBenefits FLOAT,

4 CREATE TABLE Employee
    (Job CHAR(20) Salary FLOAT, HOBBY Activity, Manager Employee)
     PROCEDURE RetirementBenefits FLOAT
     AS CHILD OF Person
        CREATE TABLE Person
            (Name CHAR(20), SSN CHAR(9) Age INTEGER),

5 CREATE TABLE Employee
    (Name CHAR(20), Job CHAR(20), Salary FLOAT, HOBBY set-of Activity, Manager Employee),
```

(그림 5) 관계형 모델의 확장

레이션 계층을 통해 사용자는 이미 존재하는 릴레이션의 자식으로 새로운 릴레이션을 생성할 수 있다. 이때 새로운 릴레이션은 이미 존재하는 릴레이션들과 그들의 조상 릴레이션의 모든 열과 프로시듀어를 계승받는다. 다음으로 릴레이션 계층을 통해 시스템은 두 릴레이션의 관계를 IS-A 관계로 유지할 수 있다. 관계형 데이타베이스에서는 사용자가 이러한 관계를 관리해야 한다.

객체지향 개념은 아니지만 관계형 모델의 근본적인 결점을 극복할 수 있는 더 하나의 확장이 가능하다. 릴레이션의 행/열 엔트리가 단지 하나의 값이 아닌 여러 개의 값을 가질 수 있도록 한다. 게다가, 그 여러 개의 값들은 각각 임의의 자료형이 될 수 있다. 그림 5의 다섯번째 CREATE TABLE 문은 Hobby 열의 자료형이 Activity의 집합임을 나타낸다. 즉 Hobby 열의 값은 사용자가 정의한 릴레이션 Activity의 튜플들의 집합이 될 수 있다. 관계형 모델에서는 릴레이션의 행/열 엔트리는 하나의 값만 가질 수 있기 때문에 사용자는 릴레이션의 열이 하나 이상의 값을 가져야 할 경우에 추가로 중복 릴레이션을 만들어야 한다. 예를 들어 한 사원이 하나 이상의 취미를 갖는다고 가정하면, Hobby 열의 서로 다른 값을 위하여 릴레이션 Employee의 튜플을 중복으로 두거나, 추가적인 릴레이션, 이를테면 Employee_Hobby와 같은 릴레이션을 만들어야 한다. 그림 6a는 집합의 표현을 나타내고, 그림 6b는 관계형 데이타베이스에서 두가지 집합 표현을 나타내고 있다(Manager 애트리뷰트는 보이지 않는다).

이상을 정리하면 네 가지 중요한 방법으로 관계형 모델을 확장할 수 있다. 비록 각각의 확장이 사소하게 보일 수 있지만, 확장의 결과는 전체적으로 데이타 모델링을 용이하게 하고, 질의 성능을 증가시킨다는



(그림 6a) UniSQL/X의 집합 표현

(그림 6b) RDB의 집합 표현

면에서 중요하다. 중첩 릴레이션과 열에 집합을 허용하는 확장은 관계형 데이타 베이스의 사용자들을 성가시게 하였던 일을 제거한다. 프로시듀어와 릴레이션 계층의 확장은 데이타 모델링과 응용 프로그램에 새롭고 중요한 가능성을 제시하고 있다. 더우기, 중첩 릴레이션과 릴레이션 계층의 확장은 OOPL의 강력한 자료형 기능을 반영한다.

이제 릴레이션에 관련된 용어를 다음과 같이 바꾸어 보자. 릴레이션을 클라스로, 릴레이션의 튜플을 클라스의 인스탄스로, 열을 애트리뷰트로, 프로시듀어를 메소드로, 릴레이션 계층을 클라스 계층으로, 자식 릴레이션을 하위 클라스로, 부모 릴레이션을 상위 클라스로 바꾼다. 위에서 설명한 데이타 모델은 객체지향 데이타 모델이다. 객체지향 데이타 모델은 관계형 모델을 확장하여 얻을 수 있다. 객체지향 데이타 모델, 확장된 관계형 데이타 모델, 통합된 관계형 및 객체지향 모델이라는 용어는 그 데이타 모델이 앞에서 설명한 처음 세개의 확장 방법으로 확장되었다면 동의어이다. 그러나 확장된 관계형 모델이 앞에서 설명한 처음 세 가지를 모두 확장하지 않았다면 객체지향 모델이라고 할 수 없다. 더우기, 이러한 확장에 의한 객체지향 모델은 그것이 관계형 모델에 기초를 두고 있기 때문에 과거 20년동안 개발된 관계형 데이타베이스 기술을 적용하여 구축할 수 있다는 점에 주목해야 한다.

5.3 질의와 자료 조작

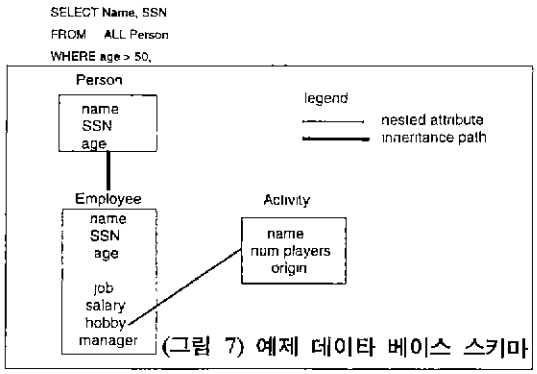
물론 사용자가 복잡한 자료를 표현할 수 있도록 데이타모델을 정의하는 것으로는 충분하지 못하다. 일단 데이타베이스 스키마가 데이타 정의 기능을 사용하여 정의되고 나면, 데이타베이스는 많은 객체로 채워지게 될 것이다. 데이타베이스 시스템은 사용자가 데이타베이스의 원하는 일부를 효율적으로 검색하고 변경할 수 있을 때 그 진가를 발휘한다. 이것을 허용하기 위하여 데이타베이스 시스템은 질의와 자료 조작(삽입, 변경, 삭제)을 위한 기능을 제공한다.

일련의 클라스들이 관계형 데이타베이스의 릴레이션으로 정의된다면, 상요자는 결합과 중첩 부속 질의, 결과를 그룹으로 나타내거나 정렬하는 질의, 뷰에 관련된 질의 등을 포함하는 ANSI SQL구문에 있는 모든 질의를 작성할 수 있어야 한다. 그림 7을 사용하여 두 개의 간단한 예를 생각해 보자. 그림에서 클라스 Employee는 클라스 Person의 하위 클라스로 정의되고 클라스 Activity는 클라스 Employee의 애트리뷰트인 Hobby의 도메인이다. 첫번째 질의는 연봉이 50000을 넘으면서 나이가 30보다 많은 고용인을 찾아 담당업무별로 분류하고 평균 연봉을 출력한다. 두번째 질의는 관리자보다 많은 연봉을 받는 고용인들의 이름을 찾는 결합 질의이다.

```
SELECT  Job, Avg(Salary)
FROM    Employee
WHERE   Salary>50000 AND Age>30
GROUP   BY Job;

SELECT  Employee, Name
FROM    Employee
WHERE   Employee.Salary>Employee.Manager.Sa-
        lary;
```

또한, 통합된 데이타 모델에 꼭 필요한 많은 새로운 질의 형태(관계형 모델에 적용할 수 없는 질의)를 표현할 수 있어야 한다. 통합된 데이타 모델은 훨씬 풍부한 표현 기능이 있어서 RDB에서 나타나지 않는 질의들을 표현할 수 있다. 특히 중첩 클라스에 대한 질의, 탐색 조건이 메소드를 포함하는 질의, 중첩 객체에 대한 질의, 클라스 계층의 클라스 집합에 대한 경로 질의(path query)를 지원한다.

클라스 계층에 대한 질의의 예로 한 클라스와 이



(그림 7) 예제 데이타 베이스 스키마

것의 하위 클라스의 인스턴스들을 검색하는 것이 있다. 다음의 질의는 키워드 ALL의 영향으로 클라스 Person과 하위클라스 Employee의 모든 인스턴스에 적용되어 결과를 산출한다.

그림 7을 사용하면 중첩 객체를 검색하는 경로 질의의 예로 다음과 같은 질의가 있다. "50,000달러보다 더 많은 연봉을 받고 취미가 테니스인 고용인들과 이들의 고용주 이름을 찾으시오". 이 질의는 클라스 Employee와 Activity에 의해서 정의되는 중첩 객체들에 적용된다. 이 질의는 술어 Name="tennis"를 클라스 Activity에, 술어 Salary>50000를 클라스 Employee에 연관지은 것이다. 이 질의는 조건을 만족하는 중첩 Employee 객체의 모든 애트리뷰트를 넘겨준다.

```
SELECT*
FROM Employee
WHERE Salary>5000 AND Hobby.Name="Tennis";
```

술어(Hobby.Name="Tennis")에서 점(.) 표기는 임의의 자료형을 통한 애트리뷰트의 중첩을 설명하기 위하여 표준 술어 표현을 확장한 것이다.

5.4 메모리 상주 객체를 위한 탐색 항해 지원

OOPL 객체에 지속성(persistency)를 지원하기 위해 OODB는 많은 객체를 메모리(작업공간. 혹은, 객체 버피 풀이라 부름)에 자동으로 관리하기 위한 작업공간 관리 기능을 제공하여야 한다. 특히 데이타베이스와 메모리 간의 저장 객체 형식을 자동적으로 전환하고, 객체가 데이타베이스에서 메모리로 적재될 때 객체에 저장된 OID를 메모리 포인터로 자동적으로 변환하며, 트랜잭션이 끝날 때 메모리에서 변경된 객체를 데이타베이스에 자동적으로 저장한다.

이러한 작업공간 관리 기능을 통하여 데이타베이스 응용 프로그램은 메모리 포인터를 통하여 메모리 상주 객체를 탐색할 수 있고, 변경된 객체를 데이타베이스에 집단적으로 반영할 수 있다. RDB의 경우에는 이를 위해 두 릴레이션을 결합하거나 적어도 하나의 릴레이션을 검색하는 질의를 이용해야 한다. 게다가 RDB에서는 변경된 튜플을 RDB 인터페이스(데이타 관리자 계층이나 저장 관리자 계층)를 통하여 한번에 한개씩 저장해야 한다. 트랜잭션이 끝날 때 변경된 객체들을 자동적으로 데이타베이스에 보내 이들의 지속성을 유지하여야 한다.

또한 완전한 질의 기능과 동적 스키마 진화를 제공하여야 한다. 어떤 시점에서 한 객체가 데이타베이스와 작업공간에 동시에 존재할 수 있고 작업공간에 있는 복사본이 변경될 수 있기 때문에, 질의 처리는 이미 작업공간에 적재된 객체에 대해서는 그 곳의 복사본을 이용하고, 작업공간에 적재되지 않은 객체들에 대해서는 데이타베이스에 있는 객체를 이용해야 한다. 게다가, 사용자가 한 클라스의 스키마를 변경하면(예를 들면, 클라스의 애트리뷰트를 추가하거나 삭제) 작업공간에 있는 객체들의 복사본은 무효화되어야 한다. 작업공간 기능은 객체에 지속성을 지원하고 OOPL로 작성된 응용 프로그램이 객체를 탐색 항해할 때 요구하는 성능을 만족하기 위해서 반드시 필요하다.

VI. 관계형 데이타베이스와의 상호운영 (interoperating)

앞에서 OODB와 RDB를 통합시키는 방법의 하나로서 설명했던 통로(gateway)방식이 OODB와 RDB의 상호운영에 유용하게 사용될 수 있다. 이 방식을 이용하면, OODB와 RDB가 공존할 수 있으며, 응용 프로그램이 하나의 OODB와 하나 이상의 RDB로부터 얻어진 자료를 이용하여 작업할 수 있다. 그러나 앞에서 언급했듯이 현재의 OODB-RDB 통로는 하나의 RDB에 대한 자료요구만을 처리할 수 있으며, OODB와 RDB에 적용되는 여러 개의 자료요구를 하나의 트랜잭션(단위적으로 처리되는 자료요구의 집합) 개념으로 처리해 주지는 못한다.

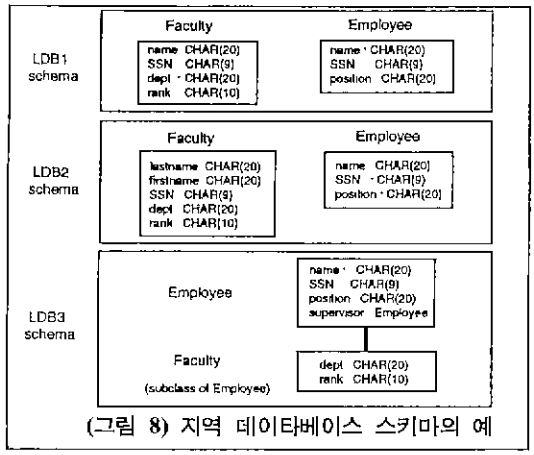
다중 데이타베이스 시스템(multidatabase system: MDBS)은 이러한 통로를 논리적으로 완전히 일반화

시킨 것이다. 하나의 MDBS는 사실상 여러 개의 통고를 통제하는 하나의 데이타베이스 시스템이라고 할 수 있다. MDBS는 자기 자신의 데이타베이스를 가지는 것이 아니고 여러 개의 원격 데이타베이스들을 각각에 하나씩 할당된 통로를 이용하여 관리하게 된다. MDBS는 여러 개의 원격 데이타베이스를 마치 하나의 가상 데이타베이스인 것처럼 보이게 한다. MDBS가 실제 자기 자신의 데이타베이스를 가지고 있지 않기 때문에 접근방법의 관리(B^+-트리색인이나 확장 해쉬의 생성이나 삭제)나, 인수화된 성능조정같은 기능들은 필요가 없게 된다.

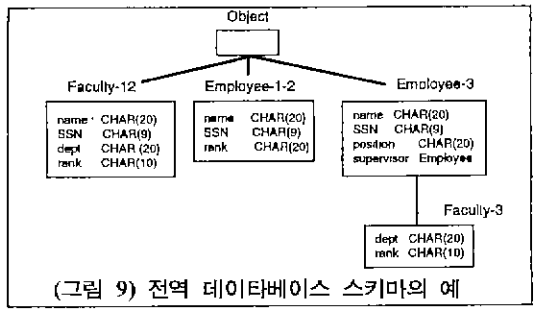
하지만, MDBS가 거의 완전한 데이타베이스 시스템이기 때문에 각각의 원격 데이타베이스를 기반으로 가상의 데이타베이스를 정의할 수 있도록 하는 자료정의 기능을 제공해야 한다. 자료정의 기능은, 여러 개의 원격 데이타베이스에 서로 다른 형태로 표현되어 있지만 동일한 의미를 갖는 자료들을 적절히 조화시킬 수 있는 기능을 포함하고 있어야 한다. MDBS의 사용자는 이러한 가상 데이타베이스에 대하여 자료정의에 관한 질의, 자료에 관한 질의, 자료의 수정 등을 수행할 수 있다(질의 최적화와 질의 처리 기능 지원이 요구됨). 또한, 여러명의 MDBS의 사용자들이 동시에 가상 데이타베이스에 대한 질의, 변경 등을 요구할 수도 있다(동시성 제어 기능이 요구됨). MDBS의 사용자는 가상 데이타베이스에 대한 여러 개의 질의, 변경 연산을 하나의 트랜잭션으로서 요구할 수도 있고 (트랜잭션 관리 기능이 요구됨) 가상 데이타베이스에 대한 사용권한을 다른 사용자에게 부여하거나 회수할 수도 있다(권한부여 관리기능이 요구됨).

MDBS의 질의나 변경요구를 원격 데이타베이스가 처리할 수 있는 것으로 변환하기 위하여 각 원격 데이타베이스에 대한 통로가 필요하다. 이러한 통로를 주로 "구동기"라고 부르고, 원격 데이타베이스를 "지역 데이타베이스"라고 부르며, 사용자에게 보이는 하나의 가상 데이타베이스를 "전역 데이타베이스"라고 부른다. 즉, MDBS는 여러개의 지역 데이타베이스를 하나의 전역 데이타베이스로 집약하게 되는 것이다.

이제 어떻게 여러 지역 데이타베이스를 하나의 전역 데이타베이스로 사상하는 지를 살펴보자. 그림 8은 세개의 지역 데이타베이스 스키마를 보여주고 있는데, LDB1과 LDB2는 RDB이고 LDB3은 OODB이다. 그리고 그림 9는 이들 세 개의 지역 데이타베이스 스키마가 하나의 정역 OODB 스키마로 표현되는 예를



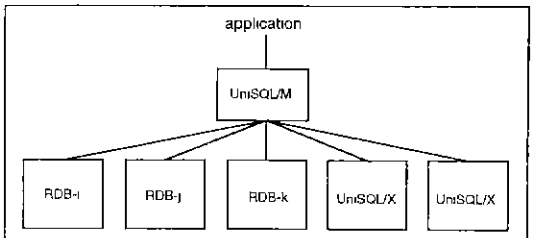
(그림 8) 지역 데이타베이스 스키마의 예



(그림 9) 전역 데이타베이스 스키마의 예

보여준다. 그림 9에서 Faculty-3 클라스는 Employee-3 클라스의 하위 클라스로서 정의된다. 전역 데이타베이스 관리자는 LDB3의 Employee와 Faculty 클라스를 전역 데이타베이스에서는 각각 Employee-3, Faculty-3이라는 이름으로 LDB1과 LDB2의 Employee들을 합하여 Employee-1-2라는 이름으로 전역 데이타베이스에 관리한다. 또한 LDB1과 LDB2의 Faculty들을 합하여 전역 데이타베이스의 Faculty-12로 재명명하되, Faculty-12의 Name이라는 애트리뷰트는 LDB2의 경우, Lastname과 Firstname 애트리뷰트로부터 유도된다. 다시 말하면, UniSQL/M의 데이타베이스 정의 기능을 이용하여 LDB1의 Faculty 릴레이션과 LDB2의 Faculty 릴레이션의 스키마상의 차이를 적절히 조화시켜서 전역 데이타베이스의 Faculty-12 클라스를 정의하게 되는 것이다. 전역 데이타베이스의 클라스는 사실상 가상 클라스(즉, 뷰)이다.

예를 들어 UniSQL에서 개발한 UniSQL/M은 MDBS 시스템으로서 그림 10에서 보는바와 같이 여러개의 UniSQL/X 데이타베이스와 여러개의 관계형 데이타베이스를 통합한다. UniSQL/M은 UniSQL/X



(그림 10) UniSQL/M 다중 데이타베이스 시스템

로부터 파생된 것으로서 하나의 완전한 데이타베이스 시스템이다. 따라서 UniSQL/M 사용자는 강력한 SQL/X 질의문을 이용하여 전역 데이타베이스에 대한 질의, 자료 변경을 할 수 있다. UniSQL/M은 전역 데이타베이스를 각 지역 RDB의 릴레이션과 각 지역 UniSQL/X의 클라스로부터 정의되는 뷰의 모임으로 관리한다. UniSQL/M은 정역 데이타베이스에 참가하는 각 지역 데이타베이스의 릴레이션, 클라스, 애트리뷰트, 자료형, 메소드 등을 디렉토리에 관리한다. 디렉토리에 저장된 정보를 이용하여 전역 데이타베이스에 대한 질의, 변경연산을 실제 해당되는 자료가 저장된 지역 데이타베이스에 대한 것으로 변환하게 된다. 각 지역 데이타베이스에 부착된 구동기는 이렇게 변환된 질의를 자신이 부착되어 있는 지역 데이타베이스를 통해 수행하고, 그 처리 결과 UniSQL/M에게 돌려준다. 그러면 UniSQL/M은 형식변환, 합병, 정렬, 그룹화, 결합 등과 같은 후처리 작업을 하게 된다. 또한 UniSQL/M은 분산 트랜잭션 기능도 제공하는데, 이것은 하나의 트랜잭션이 설사 여러 개의 지역 데이타베이스의 자료를 수정하였다 하더라도 모든 변경이 동시에 완료(commit)되든지 아니면 취소(abort) 되도록 한다는 것이다.

현재 RDB 판매사들은 다양한 수준의 통로를 공급하고 있다. 몇몇 통로는 IMS와 같은 계층형 데이타베이스 시스템이나 DEC의 RMS와 같은 화일 시스템에 SQL 질의문을 전달하는 기능을 제공한다. 또 어떤 것들은 질의와 변경연산의 처리는 물론 분산 트랜잭션 처리 기능도 갖추도록 보강하고 있다. 그러나 이들 중 SQL 질의문을 OODB에 전달할 수 있도록 한 것은 아무것도 없다.

VII. 맺는말

객체지향 개념과 이를 이용한 객체지향 데이타베이스 기술에 대하여 설명하였다. 또한 이 기술을 이용하여 기존의 관계형 데이타베이스와 어떻게 결합하여 사용할 수 있는가를 보였다. 끝으로 이를 확장하여 다중 데이타베이스 시스템을 구현할 수 있음을 예를 통하여 보였다. 다양한 응용이 요구되는 현재의 데이타베이스 응용분야에 대처하기 위하여 기존의 관계형 데이타베이스의 기술를 극복하고자 하는 관점에서 보면 객체지향 데이타베이스 기술을 쉽게 이해할 수 있으리라 믿는다.