「經營情報硏究」 第3號
大韓經營情報學會
1999年 6月

객체지향 데이터베이스와 새로운 Unified Relation 객체지향 데이터베이스의 운용에 관한 연구*

박 창 민**

〈목 차〉

I. 서론
II. 객체지향 패러다임
1. 객체개념과 객체모델 특성
III. 객체지향 데이터베이스
1. 현존하는 객체지향 데이터베이스
2. 객체지향 데이터베이스와 관계형 데이터 베이스의 상호관계
IV. URODB의 제안
1. URODB의 필요성
2. URODB에서 고려되어야 할 사항
3. URODB의 제안
V. 결론
Abstract

I. 서 론

어떤 종류의 조직에서든지 고유의 목표를 달성하기 위하여 잘 마련된 절차를 토대로 데이터를 수집, 축적, 처리하고 교환하는 데에는 상당한 자원과 활동을 투자해야 한다. 예를 들면, 은행에서는 투자정보서비스를 제공할 목적으로 데이터관리시스템을 설치하는 반면, 병원에서는 공공의료를 제공하는 것을 기초로 데이터를 구성한다.

비록 관계모델과 같은 공유의 데이터 모델이나 이론적 근거가 부족하지만 차세대 데이터베이스(DB)와 통합개발환경에서 가장 유망한 객체지향데이터베이스(object-oriented database: OODB)가 있다.

* 이 논문은 1999년 성심외국어대학 연구비지원에 의해 연구됨.
** 성심외국어대학 경영정보시스템전공 전임강사

이와 같은 객체지향 문제 해결방법 및 객체지향 패러다임은 종래의 구조화 개념에 기초한 방법론과는 완전히 다른 사고의 개념을 가진 패러다임이라고 볼 수 있다. 즉, 객체지향적 해결 방법은 보다 인간의 사고에 가까운 패러다임의 문제 해결 방법론이라고 할 수 있다.

따라서, 본 논문에서는 객체지향 개념을 이해하면서 어떻게 객체지향 패러다임으로 접근할 수 있을 것인가에 대해서 논하고, 1985년 전후로 관계형 데이터베이스의 차세대를 연구하던 데이터베이스의 연구가들은 90년도에 접어들면서 차세대 데이터베이스의 방향을 객체지향형 데이터베이스(OODB)로 잡고, 여러 가지 형태의 연구 결과로 선보이게 되었다. 과거에 많이 사용하고 있는 관계데이터베이스를 유용하게 사용하고 미래의 객체지향 개념을 접목시켜 RDB를 사용하고 있는 연구자가 객체지향 개념을 이해하려면 많은 시간과 연구가 필요하다.

본 논문의 목적은 여러 개발자가 채택한 방식을 구분하여 객체지향 데이터베이스 기술을 차세대 데이터베이스 기술로 정착하기 위해 관계형 데이터베이스를 사용하고 있고, 사용했던 개발자들에게 관계형의 장점인 모든 기능을 처리하면서 관계형 모델의 근본적인 결점을 극복하고 RDB와 OODB의 장점을 접목시킨 URODB를 제안함으로써 RDB를 사용했던 사용자가 OODB를 자연스럽게 습득할 수 있도록 했다.

객체지향의 개념을 이해하여 차세대 데이터베이스 기술로 정착하기 위하여 관계형과 객체지향의 혼합형을 제안함으로써 RDB와 호환성을 갖고 객체지향 데이터베이스의 한계성을 최대한 극복하여 차세대 데이터베이스에 더 한층 접근하는데 그 목적이 있다.

본 논문의 구성으로는 제2절에서는 객체지향 패러다임에 대해서 논하고 객체지향의 기본개념들을 정리하여 제3절에서는 객체지향 데이터베이스를 설명하고 결합하는 방법, 관계형과의 상호운영과 데이터 모델의 확장을 하고 UniSQL/M을 소개하며 제4절에서는 URODB를 제안하고 그것의 필요성과 시스템제안을 한다. 제5절에서는 결론을 내리고 향후 연구과제를 논한다.

II. 객체지향 패러다임

1. 객체개념과 객체모델 특성

객체란 어느 데이터들과 그것을 상용하기 위한 기본적인 조작, 절차들을 하나로 한다. 데이터+절차(프로그램)=객체이다. 즉 실세계의 어떤 개체를 나타내는 자료(data)와 프로그램(program)의 결합을 의미한다. 객체는 자료와 프로그램을 캡슐화(eucapsulate)한다고 말한다.

이는 사용자가 객체 캡슐 내부를 보지 못하고 대신 그 객체의 프로그램 부분을 호출함으로써 객체를 사용함을 의미한다.

각 객체는 자료 부분과 프로그램 부분으로 구성된다. 서로 같은 자료 부분, 애트리뷰트(attribute)와 같은 프로그램 부분을 갖는 모든 객체를 묶어서 클래스(class) 혹은 형(type)이라 부른다. 어떤 클래스가 다른 어떤 클래스로부터 attribute와 프로그램 부분을 계승받도록 클래스들을 배치할 수 있다. 사용자는 기존의 클래스의 하위 클래스로서 새로운 클래스를 생성할 수 있다. 일반적으로, 하나의 클래스는 하나 혹은 그 이상의 클래스로부터 계승받고, 이러한 계승으로 인한 구조는 DAG(directed acyclic graph)를 형성한다.

이 계승구조를 "계승계층(class hierarchy)"이라 부른다. 〈표 1〉은 원을 취급하여 객체의 개념을 설명한 것이다. "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. 객체지향(object-oriented)이란 문제와 그 해결방법을 가능한 인간의 사고에게 가까운 형태로 프로그래밍하기 위한 개념으로서 실제와 문제영역에 존재하는 사물이 가지는 기능이나 지식을 모델화한 개념적 객체(conceptual object)이다.

객체지향 특성은 소프트웨어 공학 분야에서 객체지향 설계개념을 프로그램 언어에 접목시킨 것으로서 소프트웨어 시스템의 설계와 구현에 사용되는 새로운 방법이다. 이것의 주된 목적은 확장성과 재사용성을 증가시켜 프로그래머의 생산성을 향상시키고 소프트웨어의 복잡성과 관리비용을 줄인다.

〈표 1〉 객체의 개념

절차	그린다
(메소드)	이동한다
	소리한다 객체 "원"
데이터	중심좌표
	반 경

객체지향이라는 용어는 오버젝트(object), 클래스(class), 자료추상화(data abstraction), 캡슐화(encapsulation), 에그리게이션(aggregation), 상속(inheritance)을 내포하고 있다.

객체지향 개념은 캡슐화와 계승이 함께 적용될 때 제기능을 발휘한다. 계승을 통하여 서로 다른 클래스가 attribute나 메소드를 공유할 수 있기 때문에, 서로 다른 클래스에 속하는 객체에 대해 같은 프로그램을 수행할 수 있다.

새로운 attribute나 메소드를 추가한다. 따라서 기존클래스에 오류를 발생시킬 기회가 줄어든다. 이런 객체지향 개념을 바탕으로 발달한 것이 객체지향기술인데, 이것은 현재 사용되는 객체지향(object-oriented)기술은 객체지향 프로그래밍 언어(예: C++, Smalltalk), 객체지향 데이터베이스시스템(object-oriented database system: OOBD), 객체지향 사용자 인터페이스(예, Macintosh,와 Microsoft의 윈도우 시스템, Frame, Interleaf의 데스크탑 출판시스템) 등을 포함한다. 본 논문에서는 객체지향 데이터베이스를 중점으로 다룬다.

III. 객체지향 데이터베이스

1. 현존하는 객체지향 데이터베이스

현재 객체지향 데이터베이스(OODB)시스템으로는 Servio의 GemStone, ONTOS의 ONTOS, Object Design의 ObjectStore, Objectivity의 Objectivity/DB 등이 있다. 이들은 모두 객체지향 모델을 지원한다. 이들이 사용자에게 제공하고 있는 기능으로는 attribute와 메소드를 갖는 새로운 클래스의 생성, 상위 클래스로부터 attribute와 메소드의 계승, 각 클래스의 인스턴스 생성, 객체 식별자(OID)를 이용한 인스턴스의 추출, 메소드의 수행 등이 있다.

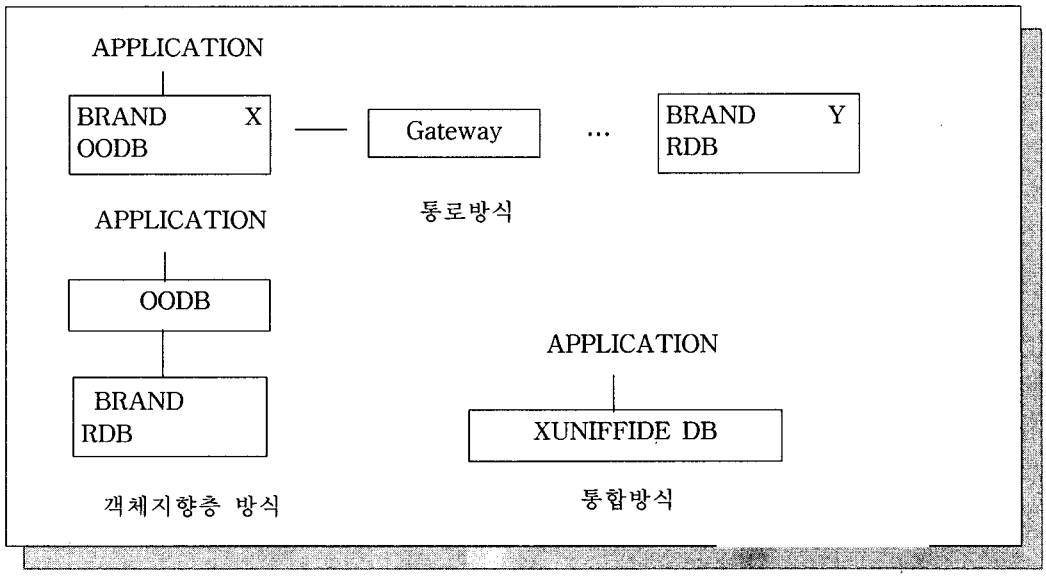
과거 5년 동안은 일반적인 객체지향 기술, 특히 객체지향 데이터베이스 기술이 태동하는 시기라 할 수 있다. 그러나, OODB가 성숙하지 않은 관계로 아직 중대한 응용분야에서는 적용되지 않고 있다. 최근에 SQL을 확장한 UniSQL의 UniSQL/X와 Hewlett Packard의 Open ODB가 데이터베이스 시장에 출현하였다. UniSQL/X는 단일 데이터베이스 시스템으로 밑바닥부터 새로이 구현된 반면 Open ODB는 HP의 관계형 데이터베이스인 ALLBase 위에서 구현되었다(즉, OODB가 RDB의 상부에 위치한다).

1.1 객체지향형 데이터베이스(OODB)와 관계형데이터베이스(RDB)를 결합하는 방법

OODB와 RDB를 결합하는 방법은 (1) OODB와 RDB 사이에 통로(gateway)를 두는 방법,

(2) RDB위에 객체지향 계층(OO-layer)을 두는 방법, (3) OODB와 RDB를 하나로 통합하여 단일시스템으로 구축하는 방법과 같이 크게 세 가지로 분류할 수 있다.

〈그림 1〉은 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 데이터베이스의 일반 사용자와 같다.



〈그림 1〉 객체지향층 방식, 통합방식, 통로방식

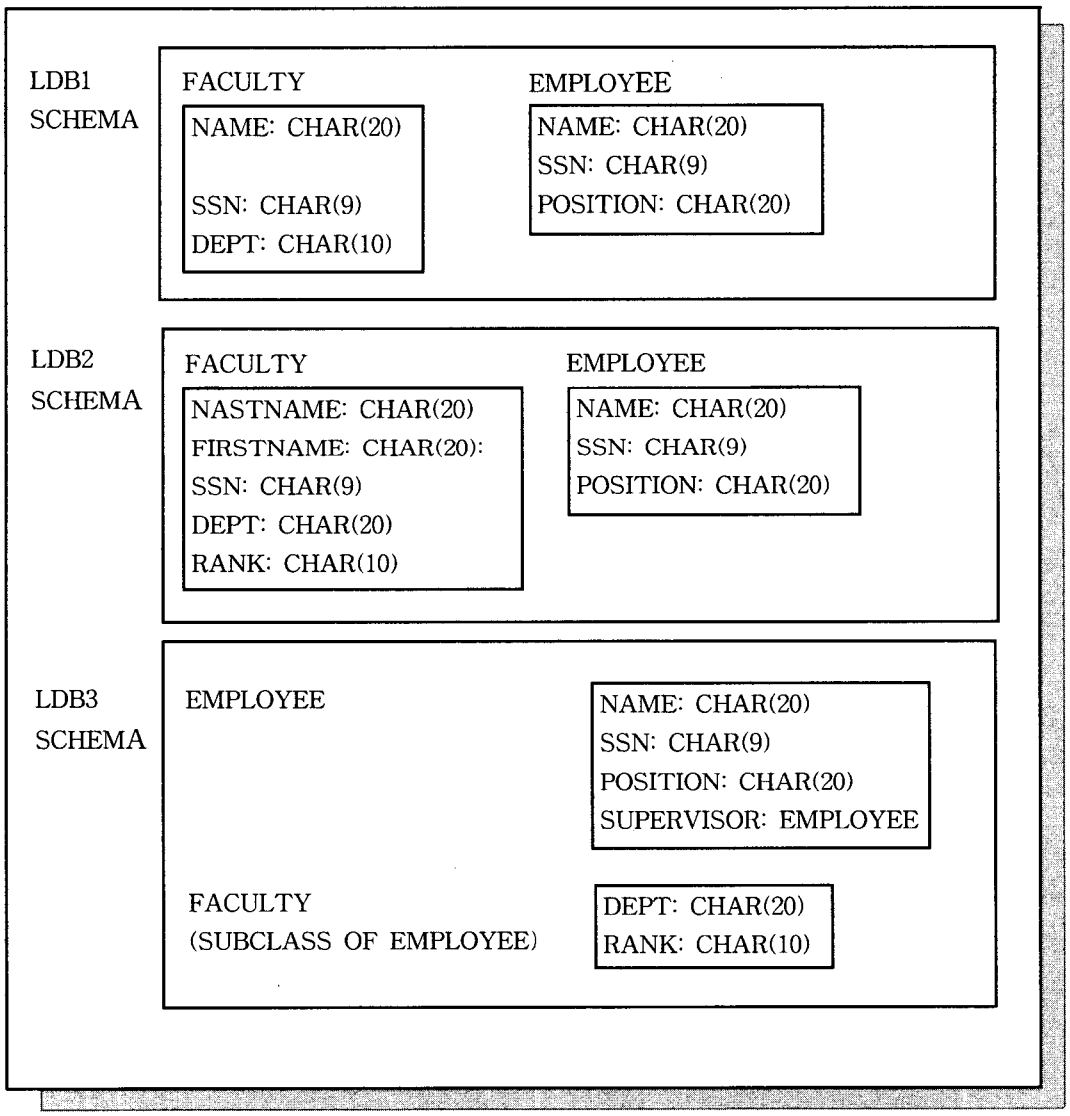
1.2 관계형과의 상호운영에 관한 지역, 전역 데이터베이스

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, 응용프로그램이 하나의 OODB와 하나 이상의 RDB로부터 얻어진 자료를 이용하여 작업할 수 있다. 그러나 앞에서 언급했듯이 현재의 OODB-RDB 통로는 하나의 RDB에 대한 자료연구만을 처리할 수 있으며, OODB와 RDB에 적용되는 여러 개의 자료요구를 하나의 트랜잭션(단위적으로 처리되는 자료연구의 집합)개념으로 처리해 주지는 못한다.

다중 데이터베이스 시스템(Multi-Database System : MDBS)은 이러한 통로를 논리적으로 완전히 일반화시킨 것이다. 하나의 MDBS는 사실상 여러 개의 통로를 통제하는 하나의 데

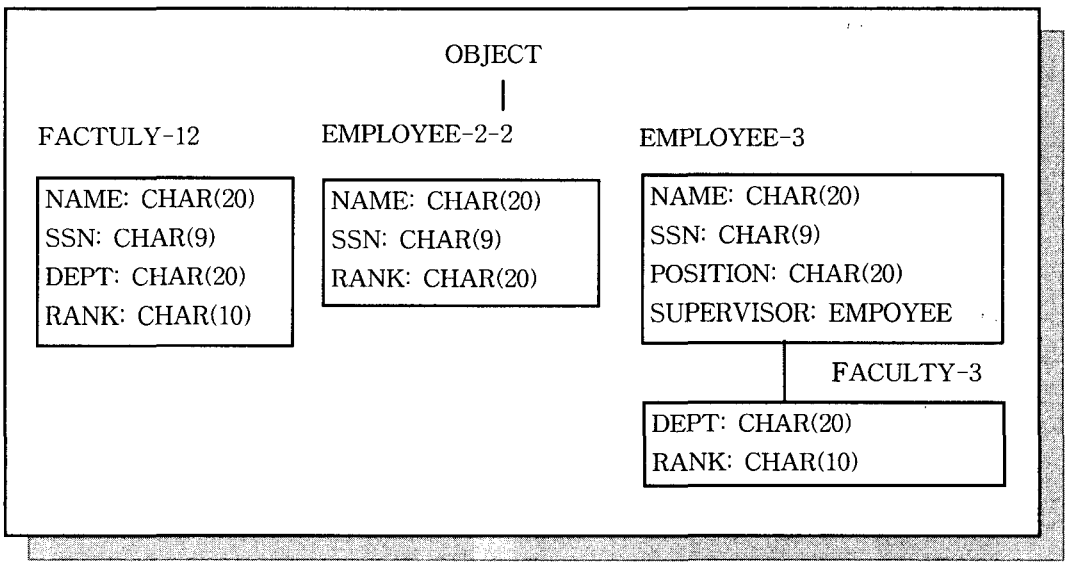
이터베이스 시스템이라고 할 수 있다. MDBS는 자기 자신의 데이터베이스를 가지는 것이 아니고 여러 개의 원격 데이터베이스들을 각각에 하나씩 할당된 통로를 이용하여 관리하게 된다.

MDBS는 여러 개의 원격 데이터베이스들을 마치 하나의 가상 데이터베이스인 것처럼 보이게 한다. MDBS가 실제 자기자신의 데이터베이스를 가지고 있지 않기 때문에 접근방법의 관리(B+ 트리트색인, 확장해쉬 생성, 삭제)나, 인수화된 성능 조정 같은 기능들은 필요 없게 된다.



〈그림 2〉 지역 데이터베이스 스키마의 예

MDBS의 질의나 변경요구를 원격 데이터베이스가 처리할 수 있는 것으로 변환하기 위하여 각 원격 데이터베이스에 대한 통로가 필요하다. 이러한 통로를 주로 "구동기"라고 부르고, 원격 데이터베이스를 "지역 데이터베이스"라고 부르며, 사용자에게 보이는 하나의 가상 데이터베이스를 "전역 데이터베이스"라 부른다. 즉, MDBS는 여러 개의 지역 데이터베이스를 하나의 전역 데이터베이스로 집약하게 되는 것이다.



〈그림 3〉 전역 데이터베이스 스키마의 예

1.3 객체지향 데이터베이스 시스템의 잠재성

객체지향 프로그래밍 언어(OOPL)는 객체들을 조직화하기 위해 클래스를 생성하고, 객체를 생성하며, 계승계층을 통해 하위클래스가 상위클래스(superclass)로부터 attribute와 메소드를 계승받을 수 있도록 하며, 특정 개체를 접근하는 메소드를 호출하는 기능을 지원한다. OODB는 이와 같은 기능을 지원함과 동시에 오늘날의 관계형 데이터베이스 시스템(RDB)이 지원하는 표준기능을 지원해야 한다. 이러한 표준기능으로는 자료검색을 위한 비절차식(non-procedural) 질의기능, 자동 질의 최적화, 동적 스키마 변경(클래스 정의 변경 및 계층 구조 변경), 질의 처리 성능 향상을 위한 접근기법의 관리(예: B+트리, 확장해쉬, 정렬 등), 트랜잭션 관리, 동시성 제어, 시스템 고장으로부터 회복, 보안 및 권한 부여 등이 있다.

1.4 데이터 모델의 확장

관계형 데이터베이스는 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 제한조건을 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. 더욱이 릴레이션들의 열의 값을 상호 비교하는 것을 기본으로 하는 결합질의를 함으로써 두 개 이상의 릴레이션을 상호 관련시킬 수 있다.

이러한 간단한 자료모델을 세 가지 방법으로 일반화 및 확장할 수 있다. 각각의 방법은 주요 객체지향 개념을 반영한다. 객체지향 시스템 또는 객체지향 프로그래밍 언어의 기본적인 원리는 객체의 값 또한 개체라는 것이다. 첫 번째 확장은 릴레이션의 열값이 시스템에서 정의한 자료형(숫자, 스트링 등) 뿐만이 아닌 사용자가 정의한 임의의 릴레이션의 튜플이 될 수 있도록 허용함으로써 객체지향 개념을 반영한다.

이는 사용자가 정의한 임의의 릴레이션을 다른 릴레이션의 열값의 범주로 지정할 수 있음을 의미한다. <표 2>에서 첫 CREATE TABLE문을 관계형 모델에서 Employee 릴레이션을 정의하는 것을 나타낸다. Hobby의 값은 더 이상 문자열로 제한되지 않고 사용자가 정의한 릴레이션 Activity의 튜플이 될 수 있다. 유사하게 Emloyee의 Manager attribute의 자료형은 Employee 릴레이션 그 자체가 될 수 있다.

두 번째 확장은 객체지향 개념인 캡슐화이다. 캡슐화는 자료를 조작하기 위한 프로그램과 자료를 결합하는 것이다. 이것을 한 릴레이션에 속하는 튜플의 열에 대한 연산을 수행하는 프로시듀어(procedure)를 그 릴레이션에 부착시킴으로써 가능하다.

세 번째 CREATE TABLE문은 주어진 사원의 퇴직금을 계산하는 Retirement Benefits Procedure를 구체화한 procedure절을 나타낸다. 각 열의 값을 읽고 갱신하는 procedure는 묵시적으로 각 릴레이션에 부착되어 있다.

한 릴레이션은 이제 그 릴레이션의 튜플의 상태와 작동을 캡슐화 한다. 상태는 열값들의 집합이고 동작은 그 열값에 대하여 연산하는 procedure들의 집합이고 동작은 그 열값에 대하여 연산하는 procedure들의 집합이다. 사용자는 릴레이션의 튜플에 대한 연산을 수행하는 어떤 procedure도 만들 수 있으며, 이들의 응용에는 가상적으로 제한이 없다. 세 번째로 객체지향 개념인 계승계층의 개념을 지원한다.

<표 2>의 네 번째 CREATE TABLE 문에서 Employee 릴레이션은 다른 릴레이션 Person

의 자식으로 정의된다. Employee 릴레이션은 자동으로 Person 릴레이션의 세 열을 계승받는다. 즉, Employee 릴레이션은 자기에게 정의되지 않았지만 Name, SSN, Age열을 갖게 된다.

〈표 2〉 관계형 모델의 확장

```
1.CREATE TABLE EMPLOYEE
(Name char(20), job char(20), salary float, 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), salaryfloat, hobbyactivity, manager empoyee)
PROCCEDURE Retirementrenefits 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(2), age INTEGER);
5. CREATE TABLE EMPLOYEE
(NAME CHAR(20), JOB CHAR(20), salary FLOAT, HOBBY set-of activity, Manager
Employee)
```

릴레이션의 계층은 관계형 모델의 독립적인 릴레이션들의 집합에 비해 두 가지 장점이 있다.

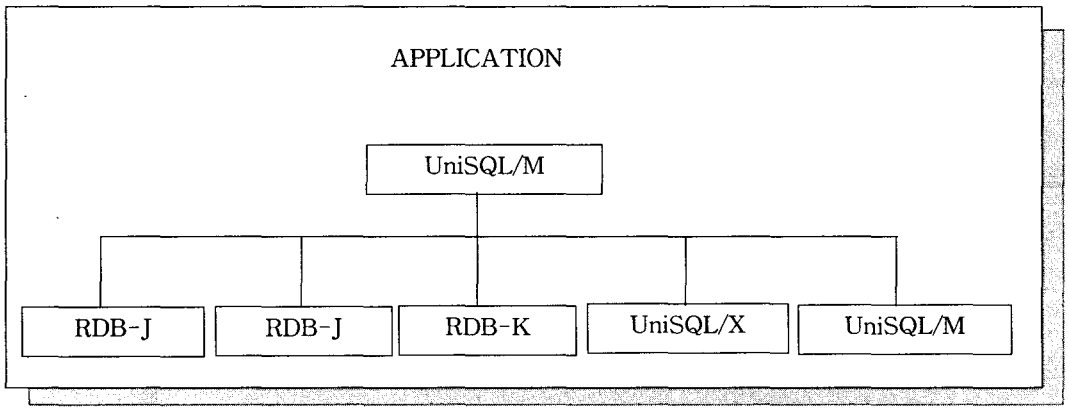
첫 번째, 릴레이션 계층을 통해 사용자는 이미 존재하는 릴레이션의 자식으로 새로운 릴레이션을 생성할 수 있다. 이때 새로운 릴레이션은 이미 존재하는 릴레이션들과 그들의 조상 릴레이션의 모든 열과 procedure를 계승받는다.

두 번째, 릴레이션의 계층을 통해 시스템은 두 릴레이션의 관계를 IS-A 관계로 유지할 수 있다. 〈표 2〉의 다섯 번째 CREATE TABLE문은 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.

1.5 UniSQL/M 소개

UniSQL에서 개발한 UniSQL/M은 MDBS 시스템으로서 〈그림 4〉에서 보는 바와 같이 여러 개의 UniSQL/X 데이터베이스와 여러 개의 관계형 데이터베이스를 통합한다. UniSQL/M은 UniSQL/M 사용자는 강력한 SQL/X 질의문을 이용하여 전역 데이터베이스에 대한 질의, 자료변경을 할 수 있다. UniSQL/X의 클래스로부터 정의되는 뷰의 모임으로 관리한다. UniSQL/M은 전역 데이터베이스에 참가하는 각 지역 데이터베이스의 릴레이션, 클래스, attribute, 자료형 메소드 등을 디렉토리에 관리한다. 디렉토리에 저장된 정보를 이용하여 전역 데이터베이스에 대한 질의, 변경연산을 실제 해당되는 자료가 저장된 지역 데이터베이스에 대한 것으로 변환하게 된다.



〈그림 4〉 UniSQl/L/M 다중 데이터베이스 시스템

각 지역 데이터베이스에 부착된 구동기는 이렇게 변환된 질의를 자신이 부착되어 있는 지역 데이터베이스를 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 데이터베이스의 자료를 수정하였다 하더라도 모든 변경이 동시에 완료(COMMIT)되는지 아니면 취소(ABORT)되도록 한다는 것이다.

현재 RDB 판매사들은 다양한 수준의 통로를 공급하고 있다. 몇 명 통로를 IMS와 같은 계

층형 데이터베이스 시스템이나 DEC의 RMS와 같은 파일 시스템에 SQL 질의문을 전달하는 기능을 제공한다. 또 어떤 것들은 질의와 변경연산의 처리는 물론 분산 트랜잭션 처리기능도 갖추도록 보강하고 있다. 그러나 이들 중 SQL 질의문을 OODB에 전달 할 수 있도록 한 것은 아무 것도 없다.

2. 객체지향 데이터베이스(OODB)와 관계형 데이터베이스(RDB)의 상호관계

RDB에서는 계층형 자료(hierarchical data), 복합중첩자료(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 attribute의 자료형은 시스템이 지원하는 원시적 자료형과 사용자가 정의한 자료형(클래스)을 가질 수도 있다 이는 한 객체가 attribute의 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 자료형은 주로 숫자나 간단한 기호들이다.

RDB는 새로운 자료형의 추가에 대비하여 구축되어 있지 않기 때문에, 새로운 자료형을 추가하려면 시스템의 구조부터 수정해야 한다. 데이터베이스 시스템에 새로운 자료형을 추가한다는 의미는 이를 attribute의 자료형으로 쓸 수 있고, 이러한 자료를 저장, 검색, 변경할 수 있음을 의미한다.

OOPL의 객체 캡슐화는 객체의 자료부분이 어떠한 자료형도 가질 수 있도록 허용하다. 더구나, 새로운 자료형이 새로운 클래스로서 생성될 수도 있으며, 심지어 이미 존재하는 클래스의 attribute와 메소드를 계승받아 하위 클래스로서 생성될 수도 있다.

객체 캡슐화는 데이터베이스의 자료뿐만 아니라 프로그램의 저장, 관리에도 기초가 된다.

RDB도 요즈음에는 프로그램이 데이터베이스에 저장되었다가 후에 적재되어 수행될 수 있는 "저장 프로시듀어(store procedure)"를 지원한다. 그러나, 이는 다른 일반 자료와 함께 캡슐화 될 수 없다. 즉, 다른 릴레이션이나 튜플과 결합되지 않는다. 게다가, RDB는 계승 개념이 없어 한번 작성된 프로그램을 자동적으로 다시 상용할 수 없다.

OODB는 RDB에 대한 성능향상을 얻을 수 있는 두 가지 근거를 가지고 있다. OODB에서는 객체 X의 attribute가 다른 객체 Y의 객체식별자(OID)를 가질 수 있다. 따라서 응용프로그램이 객체 X를 검색한 후 객체 Y를 검색하고 싶으면 데이터베이스 시스템은 객체 X의 attribute인 OID를 통해 바로 검색할 수 있다.

IV. URODB의 제안

1. URODB의 필요성

현존하고 있는 관계형 데이터베이스는 많은 기능들이 있고 관계형 DB의 관념에 사로잡혀 있다. 이런 시점에서 객체지향 데이터베이스를 바로 사용한다면 많은 혼란을 야기하고 객체지향의 개념의 교육에 많은 시간을 들여야 한다. 본 논문에서 제시한 URODB은 이런 혼란을 없애고 현존하는 RDB를 사용하면서 객체지향 개념을 가미한 혼합형 데이터베이스를 제안함으로써 객체지향 개념을 습득하여 차기 세대 데이터베이스인 객체지향 데이터베이스를 완전히 이해하여 대체해 나가는 데 그 목적이 있다.

과거 30년 동안 파일 시스템에서 관계형 데이터베이스에 이르기까지 많은 추구가 있지만 매우 복잡한 데이터베이스를 설계하고 관리하는 어려움이 있다는 것을 부인할 수 없다. 또한 OODB와 RDB를 통합시키는 방법의 하나인 통로(gateway)방식은 하나의 RDB에 대한 자료요구만을 처리할 수 있으며, OODB와 RDB에 적용되는 여러 개의 자료요구를 하나의 트랜잭션(단위적으로 처리되는 자료요구의 집합)개념으로 처리해 주지는 못한다.

또한 기존의 객체지향시스템은 관계 데이터베이스와 호환성이 없고 객체지향 데이터베이스의 한계성이 있었다. 이런 어려움을 해결하기 위해 URODB(Unified Relational Object-oriented Database)의 제안이 필요하게 되는데 이것은 관계형의 호환성을 극복했으며 객체지향 데이터베이스의 한계성을 최대한 해결했다.

URODB의 제안은 객체지향 개념을 통해 데이터베이스 진화 역사상 처음으로 데이터베이스 설계와 프로그램을 재 사용할 수 있다는 점이다. 또 한 가지 원인은 캡슐화와 계승이라는 개체지향 개념에 내재되어 있는 강력한 자료형 기능이다.

2. URODB에서 고려되어야 할 사항

완전한 질의 향상을 위해서 URODB가 갖추어야 할 기능은 다음과 같다.

(1) OODB에 사용할 수 있는 질의어를 설계해야 한다.

(2) UniSQL/M기능에서 RDB의 호환성을 갖는 질의어를 처리하기 위해 구문처리기(parser)를 구현해야 한다.

(3) 데이터베이스 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, 집합질의, 중첩질의 등과 같은 복잡한 질의를 처리하기 위해서 정렬병합결합(sort-merge-join), 질의평가, 중첩질의 처리 등과 같은 다양한 알고리즘을 구현해야 한다.

(5) 질의 처리를 위하여 정렬 패키지, B+트리 인덱스 패키지 등과 같은 다양한 알고리즘을 구현해야 한다.

(6) 데이터베이스에 대한 통계를 제공하는 카탈로그(catalog)를 개발해야 하며, 질의 처리를 위해서 이러한 통계자료를 관리하는 카탈로그 관리기를 구현해야 한다. 이 카탈로그는 클래스내에 개체의 개수 등을 저장해야 한다. 또한, 이들 외에도 클래스의 객체가 차지하는 자료 페이지의 개수, attribute의 개수 등을 저장하여야 한다. 또한 이들 외에도 클래스의 객체가 차지하는 자료 페이지의 개수, 클래스의 attribute에 설치된 인덱스의 종류 및 크기 등과 같은 정보도 질의 처리기를 위하여 적당한 시스템 카탈로그에 저장해야 한다.

(7) 다수의 사용자가 동시에 데이터베이스를 사용하거나, 시스템의 고장 또는 트랜잭션 철회가 발생하는 경우에도 시스템의 자료구조 및 데이터베이스를 일관성 있게 유지할 수 있도록 자료 접근기법을 설계하여야 한다. 즉, 접근기법은 데이터베이스 시스템이 동시성 제어 및 회복기법과 통합되어 구현되어야 한다.

(8) 질의 처리 알고리즘도 다수의 사용자가 동시에 데이터베이스를 사용하거나, 시스템의 고장 또는 트랜잭션의 철회가 발생하는 경우, 시스템의 자료구조 및 데이터베이스를 일관성 있게 유지할 수 있도록 설계하여야 한다. 즉, 질의 처리 알고리즘은 데이터베이스 시스템의 동시성 제어 및 회복기법과 통합되어 구현되어야 한다.

(9) 질의 처리를 위해 작업영역 관리기가 수정되어야 한다. 작업영역은 사용자가 필요로 하는 객체를 주기억장치에 관리하기 때문에 객체가 수정되면, 데이터베이스에 수정된 값과 서로 다를 수 있다. 따라서 질의 처리기는 작업영역에서 수정된 객체를 고려해야 한다.

3. URODB의 제안

오늘날 관계형 데이터베이스의 결합과 객체지향 데이터베이스에 대한 기대는 널리 알려져 있다. 그러나 객체지향 데이터베이스가 아직 데이터베이스 시장에 큰 영향을 미치지는 못하고 있다. 그 이유는 현재 대부분의 OODB가 데이터베이스 시스템으로서 성숙되지 않았기 때문이다(즉, 이들은 RDB가 이미 지원하는 많은 주요 기능을 아직 지원하지 못한다). 또한 현재 객

체지향 데이터베이스는 관계형 데이터베이스와 충분히 호환성을 갖지 못한다(즉, 이들은 아직 관계형 데이터베이스 언어 SQL을 포함할 수 있는 언어를 지원하지 않는다).

객체지향기법이 데이터베이스 기술에 획기적인 도약을 가져올 수 있으며 객체지향 데이터 모델을 포함하는 새로운 데이터베이스 시스템은 관계형 데이터베이스와 호환성이 있어야 한다. 따라서 URODB는 관계형 데이터베이스와 호환성이 필요하게 됐고 앞에서 언급한 RDB 판매자들이 다양한 수준의 통로를 공급하고 있고 몇 명 통로는 IMS와 같은 계층형 데이터베이스 시스템이나 DEC의 RMS와 같은 파일 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 아무 것도 없다. 본 논문에서 제안한 URODB는 이러한 결점을 보안하려고 했으며 현존하는 UniSQL/M은 여러 개의 UniSQL/X와 관계형 데이터베이스를 통합한 것에다 호환성을 추가하고 객체지향 데이터베이스의 한계성을 최대한 극복함으로써 차세대 데이터베이스로 가는 중간 단계로 URODB시스템을 제안한 것이다.

URODB의 제안은 단일시스템으로 통합하는 방법으로 RDB의 저장관리 계층과 자료관리 계층에 필요한 모든 변환을 하여 OODB와 RDB를 하나의 계층으로 만드는 것이다. 기존방법과 다른 점은 다음과 같다.

(1) 관계형과 객체지향자료 모델을 하나의 통합된 모델로 결합한다.
(2) ANSI SQL 데이터베이스 시스템은 데이터베이스 언어를 포함할 수 있는 언어를 설계하여야 한다. 따라서 데이터베이스 언어는 자료 정의, 질의(조인, 집합, 연산, 등) 갱신기능을 포함한다.
(3) 데이터베이스 시스템은 데이터베이스 언어가 허용되는 모든 기능을 지원한다. 이러한 기능은 동적 스키마 변환, 자동 질의 최적화, 자동질의 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.
(4) 통합된 시스템은 RDB에 대한 질의를 기존의 RDB와 비교할만 하게 수행할 수 있어야 하고, 동시에 OODB에 대한 질의를 기존의 OODB와 비교할만 하게 수행한다.
(5) RDB와 OODB의 통합과는 별도로 멀티미디어 자료관리, 시간적(temporal) 자료관리를 위한 해결책을 제공한다.

(6) 최적의 질의 최적기가 있다.

(7) PARSER가 있다

(8) 호환성 기능이 있다. 즉, SQL을 포함하는 관계형 데이터베이스와 호환성이 있다. RDB가 이미 지원하는 많은 주요 기능을 완전히 지원함.

(9) 객체지향 데이터모델 확장을 하여 데이터모델확장을 시도했다.

중첩 릴레이션과 열과 집합을 허용하는 확장은 관계형 데이터베이스의 사용자들을 성가시게 하였던 일을 제거한다. procedure와 릴레이션계층의 확장은 데이터 모델링과 응용프로그램에 새롭고 중요한 가능성을 제시하고 있다. 더욱이 중첩 릴레이션과 릴레이션계층의 확장은 OOPL의 강력한 자료형 기능을 반영한다.

이제 릴레이션에 관련된 용어를 다음과 같이 바꾸어 보자. 릴레이션 클래스로, 릴레이션의 튜플을 클래스의 인스탄스로, 열을 attribute로, procedure를 메소드로, 릴레이션계층을 클래스 계층으로, 자식 릴레이션을 하위 클래스로, 부모 릴레이션을 상위 클래스로 바꾼다. 위에서 설명한 데이터 모델은 객체지향 데이터 모델이다. 객체지향 데이터 모델은 관계형 모델을 확장하여 얻을 수 있다.

(10) 지속성을 지원하는 저장시스템으로서의 한계 극복

모든 객체를 지속적인 자료인지 비지속적인 자료인지 자동적으로 선언된다.

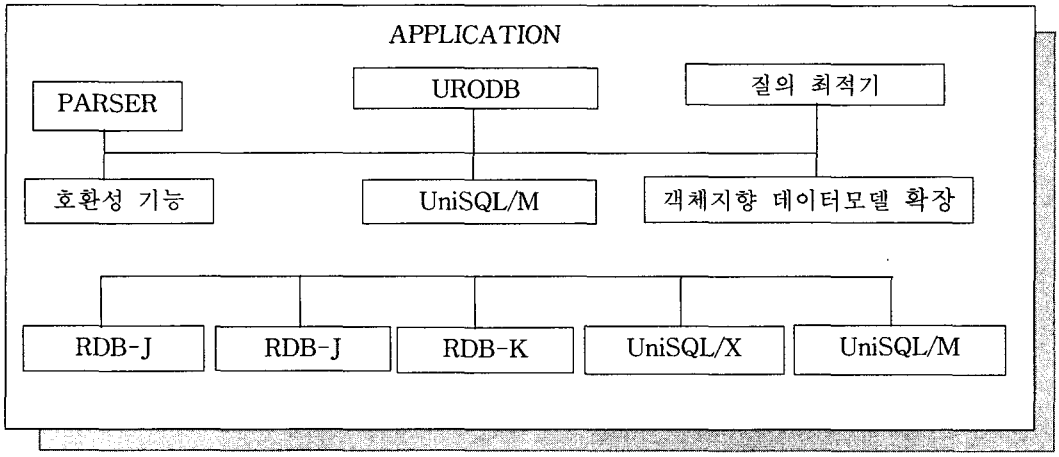
V. 결 론

과거 십년 동안 객체지향(object-oriented)기술은 프로그래밍 언어, 사용자 인터페이스, 데이터베이스, 운영체제, 전문가시스템 등의 영역에 적용되어 있다. 수년 동안 객체지향 데이터베이스라는 이름을 가진 제품이 시장에 나타났으며, 관계형 데이터베이스 시스템을 개발했던 회사들이 그들의 제품에 객체지향 기능을 추가하였다고 선언하였다.

몇몇 회사는 현재 관계형 기능과 객체지향 기능을 하나의 데이터베이스 시스템에 결합하여 지원하고 있다. 그러나, 이러한 파동에도 불구하고, 사용자, 업계 잡지, 그리고 개발자까지도 객체지향 데이터베이스 시스템, 객체지향 기능을 갖는 관계형 데이터베이스 시스템, 그리고 심지어 이러한 시스템의 필요성까지 많은 혼동을 갖고 있다.

따라서 본 연구는 URODB는 UniSQL/X 데이터베이스와 여러 개의 관계형 데이터베이스를 통합하고 거기에서 문제된 관계형 데이터베이스의 호환성 문제와 객체지향 데이터베이스의

한계성을 극복하기 위한 가상적인 시스템을 제안했고 이 가상적인 시스템이 하나의 완전한 데이터 시스템으로 구축되려면 URODB의 제안에 따라 시스템분석 및 설계를 하여 시스템을 구현함으로써 차기 객체지향 데이터베이스로 UniSQL/M 보다 나은 시스템으로 구축하여 재평가 받는 것이다.



〈그림 5〉 URODB 시스템의 구조

URODB의 장점은 관계형 데이터베이스와 호환성(모든 기능이 지원됨)이 있으며 객체지향과 관계형의 중간단계로 관계형으로 객체지향개념을 갖는 URODB는 현존하는 객체지향 데이터베이스와 차기 데이터베이스의 중간단계라고 규정할 수 있다. 왜냐하면 차기 데이터베이스는 완전한 객체지향 데이터베이스며 URODB는 이것을 모방했지만 객체지향분석 및 객체지향적으로 설계하여 객체지향언어로 구현하지 못했기 때문에 완전한 차기 데이터베이스라고 할 수가 없다.

향후 연구과제는 URODB이 가상적인 객체지향 관계형 시스템이기 때문에 이것에 맞게 실제로 설계 및 구현을 해야 한다. 설계 및 구현을 하여 기존의 UniSQL/M과 비교하여 보다 나은 시스템을 만드는 것이고 향후 차기 데이터베이스의 기본이 되고 영향을 주는 것이 최대 연구 과제이다.

참 고 문 헌

1. A.S. Pollitt, *Information storage Retrieval System*, Ellis Horwood, 1989.
2. Eduardo Ostertag, J. Hendler, R.P. Diaz, C. Braun, "Computer Similarity in a Real Library System," *An AI-Based Approach*, ACM, July 1992.
3. G. Booch, *Software Engineering with Ada*," Benjamin Cummings, 1983.
4. I. Sommerville, *Software Engineering*, Third ED., Addison Wesley, 1990.
5. J. Desoi, "A Graphical Environment for user interface design and Development," *Software Engineering Journal*, Sep. 1990.
6. J.E. Sammet, A. Ralston, "The New Computing Reviews Classification System-Final Version," *CACM*, vol. 25, Jan, 1982.
7. R. Lanergan, B.A. Poynton, "Reusable code-The application development technique of the future," *Proc. Joint SHARE/GUIDE/IBM Appl. Develo. Symp.* pp. 127~136 Oct. 1979.
8. R.J. Abbott, "Program Design by Informal English Description," *CACM*, Vol. 26, No. 11, pp. 882~894, Nov. 1983.
9. T.C. Jones, "Reusability in Programming: A Survey of the State of the Art," *IEEE Transactions on Software Engineering*, pp. 488~494, Sep. 1984.
10. W.B. Rauch-Hindin, "Reusable Software," *Electronic Design*, pp. 176~194, Feb. 1983.

Abstract

A Study on Application Object-oriented Database and New Unified Relation Object-oriented Database

Park, Chang-min

This paper proposes to study on the interoperation between Relation Database(RDB) and Object-oriented database(OODB). It also describes some Characteristics of OODB and introduces the UniSQL system. This paper suggests a new systemcalled URODB (Unifide Relation & Object-oriented database) that will overcome the limits of OODB, improve the shortcomings of the present RDB, and increase the interchangeability in particular.