객체지향 데이터베이스 시스템의 필요요건과 설계에 관한 연구

유 양 근[†] · 류 해 영[††]

요 약

본 논문은 객체지향 패러다임에 입각한 데이터베이스 시스템의 기본적인 개념과 필요 요건을 제시하고 이에 부응하는 객체지향 데이터베이스 시스템의 개괄적 설계를 보인다. 이 시스템은 C++에 기초하여 크게 세개의 계층으로 구성 된다. 최하층은 저장시스템(storage system)으로 화일 입출력을 수행하고 중간층은 트랜잭션 관리, 스키마관리,주기억장치에서의 버퍼관리 등 사용자 인터페이스를 제외한 여러 기능을 담당한다. 최상위층은 사용자인터페이스를 위하여 설계되어 그래픽 사용자 인터페이스(graphical user interface), 전처리기(preprocessor), 인터프리터 등 여러관련 기능을 제공할 뿐만 아니라 객체지향의 여러 특성을 위해 확장된 SQL을 지원하도록 한다.

A Study on the Design and Requirements of the Object-oriented Database Systems

Yang Keun You[†] · Hae Young Lyu[††]

ABSTRACT

This paper introduces the basic concepts and requirements of database system based on the object-oriented paradigm, and presents the overview of the design of an object-oriented data base system which reflects those concepts and requirements. The system implemented on C++, consists of three structure layers. The inner layer, which is in fact a storage system, performs file I/O, while the intermediate layer is responsible for most of the functions except the ones rerated to the user interface, such as the transaction management, the schema management, and the management of buffers in main memory etc. The outer layer, designed mainly for the user interface, not only provides the functions for graphical user interface, preprocessor, and interpreter etc., but also supports extended SQL for object-oriented features.

1. 서 론

최근들어 객체지향 패러다임(paradigm)은 새로운 응용 시스템을 설계하고 구현하는데 많이 사용되고 있다. 인공지능 분야의 지식표현 기법, 컴퓨터를 이용한 설계/생산(CAD/CAM), 다중 매체(multimedia)로 구성된 문서의 처리를 위한 사무정보 시스템(Office Information System), 데이터베이스 시스템[MAI86] 등의 분야는 물론 운영체제 설계, 분산 시스템 설계까지의 다양한 분야에서 이 개념은 널리 이용되고 있

† 종신회원:강남대학교 문헌정보학과
†† 정 회 원:단국대학교 전산통계학과
논문접수:1996년 10월 14일, 심사완료:1997년 3월 12일

다. 이렇게 객체지향 개념이 새로운 응용분야에서 각광을 받는 이유는 기존의 절차적 언어(procedural language) 개념에 비하여 다음과 같은 장점이 있기 때문이다. 첫째, 실세계의 모든 개념적 개체는 단일 개념의 객체로서 모델링될 수 있고, 데이터 추상화(data abstraction)개념을 잘 지원해 준다. 둘째, 데이터 사이에 존재하는 복잡한 관계를 전문화(specialization)/일반화(generalization)와 복합객체(complex object)를 사용하여 쉽게 표현할 수 있다. 셋째, 객체는 여러 형태의 멀티미디어 데이터를 적절히 다룰 수 있도록 쉽게 구현할 수 있다. 넷째, 시스템을 설계 구현하는데 있어 빨리 모형화(prototy ping)할 수 있다. 다섯째, 병행처리(concurrent processing)를 자연스럽게 지원해 줄 수 있다. 여섯째, 사용자에게 편리한 그래픽 인터페이스를 지원하기 쉽다. AI[STE86], CAD/CAM [AFS85], OIS[AHL84] 등의 새로운 응용분야에서는 많은 양의 데이터를 처리할 수 있는 데이터베이스 기능이 필요할 뿐만 아니라, 또한 그 데이터 형태도 정형(formatted) 데이터는 물론 텍스트, 그래픽스, 이미지, 음성 등의 멀티미디어 비정형(unformatted) 데이터의 처리 기능을 요구한다. 따라서 본 연구에서는 위와같은 응용분야의 기본 시스템이 될 객체지향 개념을 기반으로 한 차세대 데이터베이스 시스템의 연구 방향을 제시하고자 시스템의 필요 요건과 설계 방법을 논하고자 한다. 본 논문의 구성은 다음과 같다. 2장에서는 객체지향 패러다임을 설명하고, 3장에서는 객체지향 데이터베이스 시스템의 요건을 필수적인 기능과 선택적인 기능으로 나누어 제시하였고, 4절에서는 객체지향 데이터베이스 시스템 설계 방법을 클라이언트-서버 프로세스 구조, 저장관리자, 객체관리자, 사용자 인터페이스로 구분하여 논하였으며 5절에서 본 논문의 결론을 기술하였다.

2. 객체지향 패러다임

객체지향 패러다임(paradigm)은 최초의 객체지향 프로그래밍 언어라 할 수 있는 시뮬레이션(simulation) 언어 SIMULA[NYG66]에서 그 개념이 적용되어, 대화식이며 인터프리터(interpreter) 방식으로 구현된 Smalltalk[GOL83] 언어에서 그 개념이 정립되었다. 그 후 많은언어－Flavor, C++[STR86][STR89], CLOS[MOO89]들이 객체지향 패러다임을 도입하여 설계되었고, 그 응용시스템들도 개발되었다. 그러나 객체지향 개념은 각 언어 또는 시스템에 따라 그 특성에 맞게 여러 가지로 정의되어 사용되고 있으나 완전히 통일된 개념으로 정착되어 있지 않다.

객체지향 개념은 객체(object), 메시지(message), 계승(inheritance)의 세가지 요소가 그 근간을 이루고 있다. 실세계의 모든 개체(entity)들은 시스템 내에서 객체로 모델링되어 인스턴스(instance)로 표현되며 공통된 특성을 가지는 인스턴스들이 모여 하나의 클래스(class)를 구성하게 된다. 각 인스턴스 객체는 애트리뷰트로 자신의 데이터를 저장하고, 메쏘드(method)를 이용해 그 데이터에 대한 연산을 규정한다. 이러한 점은 데이터와 연산을 별도로 취급하는 기존의 절차적 언어와 구별되는 중요한 기준이 된다. 객체지향 개념에서의 연산은 시스템에 존재하는 객체들 간에 메시지를 주고 받음으로써 이루어진다. 메시지를 보내는 형태는 (message target, selector, parameters, …)로 되어 있는데, message는 메시지를 보낸다는 의미이고, target은 메시지를 받는 객체를 지정하며, selector는 메시지를 받은 객체에 적용할 연산자 즉 메쏘드를 의미하며, parameters는 그 연산자가 필요로 하는 매개변수들 이다. 한 객체가 메시지를 받으면 이 메시지의 선택자(selector)에 해당하는 메쏘드를 실행시켜 부수효과(side effect)를 일으키거나 객체를 반환(return)함으로써 응답하게 된다. 이 점은 모든 연산의 수행이 절차적으로 기술되는 절차적 언어에 비해 각 객체가 자신의 연산을 내부에서 스스로 수행하게 함으로써 데이터 추상화를 제공하는 중요한 개념이 된다. 객체지향 패러다임에서 계승이란 하위 객체가 상위 객체의 특성(애트리뷰트, 메쏘드 등)을 모두 계승받을 수 있도록 해 주는 개념이다. 이 개념을 이용해 유사한 객체들이 공통적인 부분을 상위 객체로 정의함으로써 중복된 정의를 피할 수 있으며, 한 번만 정의되므로 이들의 수정도 용이하게 할 수 있다. 계승관계로 형성된 객체들 사이의 구조를 계승 계층(inheritance hierarchy)이라 하고 IS-A 관계가 있으므로 하위 객체는 상위 객체를 대신할 수 있다. Smalltalk-80에서는 한 객체가 오직 하나의 상위 객체만을 가질 수 있도록 하는 트리 형태의 계층 계승을 지원하고, Flavor, C++와 같은 시스템은 여러 개의 상위 객체

를 가질 수 있도록 허용하는 다중 계승(multiple inheritance)을 지원한다. 데이터베이스에서는 실세계의 엔티티 사이에 존재하는 복잡한 의미적 관계성을 자유롭게 모델링할 수 있어야하므로 다중 계승을 지원하는 추세에 있다[BAN87]. 보통의 객체지향 개념에서는 복잡한 구조체를 쉽게 표현하는 복합 객체가 있다. 회로 설계나 복잡한 개체를 컴퓨터에서 표현할때 대개 이러한 복잡한 객체는 서로 연관된 여러 개의 상대적으로 덜 복잡한 구성요소(component)의 그룹으로서 표현한다. 이런 복합 객체를 쉽게 표현하기 위해서는 여러 객체들이 HAS-A 관계를 가지고 순환적으로 정의함으로써 자연스럽게 모델링할 수 있다. 앞으로의 데이터베이스 시스템은 다양한 응용시스템을 지원할 필요가 있으므로 이 복합 객체는 데이터베이스 시스템에 있어서 매우 유용한 개념이다.

3. 객체지향 데이터베이스 시스템의 요건

3.1 필수적인 기능

객체지향 데이터베이스 시스템은 두 가지의 기준을 반드시 만족해야 한다. 즉 데이터베이스 시스템으로서의 기준과 객체지향 프로그래밍 언어에서 제공되고 있는 객체지향 시스템으로서의 기준을 만족해야 한다. 전자의 기준은 지속성, 보조기억 장치의 관리, 동시성 제어, 회복 기능, 그리고 특정 질의 기능과 같은 다섯 가지의 기능을 포함하고, 후자의 기능은 복합 객체, 객체 식별자, 캡슐화, 타입 또는 클래스의 계층적인 구조, 속성 계승, 늦은 바인딩과 결합된 오버라이딩, 확장성, 그리고 계산 표현의 완전성과 같은 여덟 가지의 기능을 포함하고 있다.

3.1.1 복합 객체

복합객체는 일반적으로 이종 객체들의 모임, 즉 여러 다른 클래스들에 속하는 요소 객체들의 집합으로 볼 수 있으며 복합 객체는 단순한 객체돌을 조직화함으로써 만들어 진다[KIM88]. 가장 기본적인 객체들은 정수, 문자, 스트링, 진리값, 부동소수와 같은 원자적인 타입의 객체들이다[KIM89]. 이로부터 시작하여 튜플, 집합, 백(bags), 리스트, 배열 등과 같은 객체들이 복합 객체의 예들이다. 특히 시스템은 집합, 리스트, 그리고 튜플은 최소한 지원해야 한다. 집합은 데이터들이 모여서 데이터베이스를 형성하는데 이를 표현하는데 가장 자연스러운 복합 객체이며, 튜플은 현실세계의 개체의 특성을 표현하는 가장 자연스러운 복합 객체이다. 물론 집합과 튜플은 관계형 모델에서도 널리 사용하는 객체들이다. 그리고 리스트와 배열은 현실 세계에서의 순서를 표현하는데 필수적인 객체들이다. 복합 객체를 지원하기 위해서는 그러한 객체들에 적용될 수 있는 적절한 연산자들도 지원해야 한다. 복합 객체에 대한 연산은 그 객체의 모든 구성요소들에 대하여 전이적으로 전파되어야 한다. 예를들어 완전한 복합 객체에 대한 검색, 삭제, 또는 생성하는(deep copy, shallow copy) 연산자들은 그 복합 객체의 모든 구성요소에 대하여 점진적으로 적용되어야 한다.

3.1.2 객체 식별자

객체 식별자를 가진 데이터 모델에서는 그들이 가지는 값과 무관하게 존재할 수 있으므로 객체의 일치성(equivalence)에는 두 가지의 개념이 존재한다. 즉, 두 객체가 동일하다(identical)는 개념과 두 객체의 값이 서로 같다(equal)는 개념이다. 이러한 개념에 의하여 객체 공유(object sharing)와 객체 수정(object update)에 대하여 함축된 의미를 지닌다[HAL76].

객체 공유 문제에는 객체 식별자를 기반으로 하는 모델에서는 다른 두 객체가 같은 한 객체를 서로 공유할 수가 있고, 객체 수정 문제에는 모든 객체들이 공유가 가능하여 중복되지 않으므로 오직 하나의 객체만 수정해도 된다.

객체 식별자는 집합, 튜플, 리스트 그리고 배열 등의 복합 객체에서 각 원소 객체와 순환적인 복합 객체를 처리하는 강력한 도구로서 복합 객체들의 자료처리를 위한 기반이 된다. 복합 객체를 지정(assign)하거나, 복사(copy)할 때 복합 객체는 그 구성객체들에게 순환적으로 연산자를 적용하는 여부에 따라 깊은 또는 얕은(deep or shallow assignment and copy) 연산자들을 지원해야 한다. 그리고 복합 객체의 동일성과 동질성을 테스트하는 연산자(deep or shallow identity and equality)들에도 마찬가지 이다. 이러한 객체 식별자는 프로그래밍 언어에서의 변수 이름과 메모리 위치에 의하여 유추되고 복합 객체에 대한 연산자들은 객체지향 언어의 객체에 대한 연산자들에 대응하

는 개념이다.

3.1.3 캡슐화

캡슐화의 개념은 연산자의 구현(implementation)과 명세(specification)를 명확히 구별하고 모듈화를 위한 필요성에서 유래되었다. 모듈화는 프로그래머들이 팀을 이루어 설계하고 구현하는 복잡한 응용프로그램을 구조화 하는데 필수적이다. 캡슐화는 프로그래밍 언어적인 관점과 데이터베이스에서 이를 적용한 관점이 서로 다르다. 프로그래밍 언어에서의 캡슐화는 추상 데이터 타입(abstract data type)에서 유래되었다. 이러한 관점에서 객체는 인터페이스 부분과 구현 부분으로 되어 있다. 인터페이스 부분은 그 객체에 적용될 수 있는 연산자들의 집합에 대한 명세를 의미한다. 구현 부분은 데이터 부분과 절차 부분으로 되어 있는데, 데이터 부분은 객체의 표현 즉, 객체의 상태를 의미하며 절차 부분은 각 연산자에 대하여 어떤 언어로 구현을 설명하는 부분이다. 데이터베이스 관점에서는 객체는 프로그램과 데이터를 모두 캡슐화 한다. 캡슐화는 일종의 논리적 데이터 독립성(logical data independence)을 제공한다. 어떤 타입의 데이터 부분과 구현 부분을 변경하더라도 이 타입의 객체를 사용하는 응용프로그램을 전혀 변경시키 않고도 그대로 사용할 수가 있다. 그러므로 응용프로그램은 시스템의 하위 계층에서 구현 상의 변화로부터 보호받을 수가 있다.

3.1.4 타입과 클래스

타입은 같은 성질을 가진 객체들의 집합의 공통된 특성으로 추상 데이터 타입의 개념에 해당한다. 이 개념은 인터페이스와 구현의 두 부분으로 되어 있는데, 인터페이스 부분은 타입의 사용자에게 보이고 구현 부분은 오직 타입의 설계자에게만 보인다. 인터페이스는 입력 인자들의 타입과 결과의 타입으로 구성된 시그너처(signature)를 가지는 연산자들의 집합이다. 타입 구현 부분은 객체 데이터의 내부적인 구조를 설명하는 데이터 부분과 인터페이스 부분의 각 연산자를 구현하는 절차를 의미하는 연산 부분으로 구성된다. 클래스의 명세(specification)는 타입과 같지만, 클래스는 실행시(run-time)에 동적인 의미를 더 많이 가지고 있다. 이것은 인스탄스를 만드는 객체 공장(object factory)과 인스탄스를 관리하는 객체 창고(object warehouse)와 같은 성질을 가지고 있다. 객체 공장은 클래스의 *new*와 같은 연산자를 사용하여 새로운 객체를 만들기 위하여 사용된다. 객체 창고는 그 클래스에 속하는 객체들의 집합 즉 외연(extension)에 해당하며 이러한 객체들을 관리한다. 클래스는 객체지향 시스템에서 일반적으로 사용되는 연산자와 애트리뷰트 이외에 데이터베이스의 관계를 쉽게 표현하도록 하는 특별한 타입인 Relationship을 지원한다[ATW93]. 이러한 타입의 애트리뷰트는 관계를 표현하기 위하여 관련된 객체나 객체들의 집합을 참조하기 위하여 사용된다[TOM89]. 일반적인 데이터베이스의 일대일, 일대다, 다대다의 관계는 이와같은 Relationship 타입의 애트리뷰트를 사용하여 쉽게 표현된다.

3.1.5 형질 계승

형질 계승은 두 잇점이 있다. 즉 실세계에 대하여 정확하고 자연스러운 표현을 제공하므로 강력한 모델링 도구가 될 수 있고, 응용프로그램에서 명세와 구현에 있어서 공통된 부분을 분해하는데 많은 도움을 얻을 수가 있다. 또한 이로부터 애트리뷰트와 연산자를 계승받음으로써 코드를 재사용할 수가 있다. 형질 계승에는 최소한 4 종류 즉, 대치(substitution) 계승, 포함(inclusion) 계승, 제약(constraint) 계승, 세분화(specialization) 계승이 있다. 대치 계승은 계승받은 하위 타입은 언제나 상위 타입을 대신할 수 있는데 객체의 구조보다는 행동의 특성에 기인된다. 포함 계승은 하위 타입의 모든 객체들은 또한 상위 타입의 객체도 되는데 객체의 행동보다는 구조의 특성에 기인된다. 제약 계승은 포함 계승의 부분적인 경우로 하위 타입의 모든 객체들은 상위 타입 객체들의 모든 제약을 만족한다. 세분화 계승은 하위 타입의 객체들은 상위 타입의 객체들에 비하여 더 많은 정보를 가지고 있어서 더 세분화된 객체가 된다.

3.1.6 오버라이딩(overriding)과 오버로우딩(overloading)

사칙 연산자(+, −, *, /)들은 정수와 부동소수에 대하여 같은 이름이 서로 다르게 구현되어 다형성(polymorphism)을 가지는 연산자들이다. 정수의 덧셈

연산자와 부동소수의 덧셈 연산은 서로 다르므로 피연산자들의 타입을 고려하여 하나의 연산자를 호출해야 한다. 그리고 타입 또는 클래스의 계층 구조에서 하위 타입은 상위 타입의 형질을 계승 받는다. 하위 타입의 객체는 상위 타입의 객체에 비하여 좀 더 세분화된 특성을 가지므로 각 연산자들도 그 객체에 좀 더 세밀한 행동(behvior)을 필요로 하는데, 대부분의 객체지향 시스템에서는 같은 이름을 가지면서 새로이 구현된 다른 연산자를 재정의하여(overriding) 사용한다. 물론 상위 타입의 연산자들도 재사용될 수 있고 새로운 연산자를 정의 할때 이용되기도 한다. 일반적으로는 하위 객체를 상위 객체에 비하여 좀 더 세부적인 일을 하므로 상위 연산자에 추가적인 일을 첨가한다. 이렇게 하나의 이름에 대하여 서로 다른 여러 연산자들이 관련되어(overloading) 있으므로 컴파일시(compile-time)에 연산자 이름으로만 적합한 구현을 바인드할 수 없고, 실행시에 바인드된 피연산자 객체들을 고려하여 시스템이 적절한 연산자를 바인딩해야 한다. 이러한 늦은 바인딩(late binding)은 타입 체크를 어렵게 할 뿐만 아니라 불가능한 경우도 있다.

3.1.7 확장성

데이터베이스 시스템은 미리 정의되어 있는 타입들로 시작하여 응용프로그램을 작성한다. 이러한 타입들을 기반으로 하여 사용자들이 적절히 하위 타입으로 확장하여 새로운 타입을 정의하여 사용할 수 있는 수단을 제공해야 한다. 이러한 새로운 타입의 정의는 캡슐화를 만족해야 하고 이 타입에 적용할 수 있는 연산자들의 정의도 포함해야 한다. 사용자 정의 타입은 사용 상에 있어서 시스템에서 미리 정의된 타입과 전혀 차이가 없다.

3.1.8 계산 표현의 완전성

계산 표현의 완전성은 프로그래밍 언어의 관점에서는 모든 계산식을 표현할 수 있어 계산의 완전성은 명백하지만, 데이터베이스의 관점에서는 개체를 다루는 SQL이 완전하지 못하므로 관심을 가지는 새로운 기능이다. 객체지향 데이터베이스 시스템의 언어를 설계할 때 대부분은 계산의 완전성을 가지는 기존의 언어를 확장하여 연결하려고 시도하고 있다. 이러한 시스템에서는 원자적 값에 대하여 간단한 계산만 수행하여 자료를 저장하고 검색하는 데이터베이스 시스템에 비하여 표현 능력에 있어서 제약이 없는 강력한 기능을 제공한다.

3.1.9 지속성

지속성은 한 프로그램의 프로세스가 사용하는 데이터들을 보조기억 장치에 저장함으로써 자신은 수행이 끝나더라도 계속 살아 있어 다른 프로세스가 이를 다시 적재하여 재사용할 수 있도록 하는 기능이다. 일반적으로 프로그램의 변수가 의미하는 객체는 실행이 끝나면 없어지고 다시 실행하면 변수들을 지정(assignment)하여 사용해야 한다. 지속성은 타입에 독립적이어야 한다. 즉 어떤 타입에 관계없이 그리고 사용자가 데이터가 지속되도록 명시적으로 이동하거나 복사하지 않고도 모든 객체들은 어떠한 변환도 없이 암시적으로 지속되어야 한다.

3.1.10 보조기억 장치의 관리

보조기억 장치의 관리 기능은 데이터베이스 관리 시스템의 가장 기본적인 기능이다. 이 기능에는 색인 관리, 데이터 클러스터링(clustering), 데이터 버퍼링(buffering), 접근 경로 선택, 질의 최적화 등의 기능을 제공해야 한다. 이러한 기능들의 사용자들에게는 전혀 보이지는 않지만 시스템의 성능에는 중요한 요소가 된다. 시스템의 논리적인 계층과 물리적인 계층 사이에는 데이터의 독립성이 보장되어야 하므로 색인 관리, 디스크 공간의 관리, 보조기억 장치와 주기억 장치 사의 데이터 이동을 지원하는 모든 기능을 제공해야 한다.

3.1.11 동시성 제어

다중 사용자들의 동시성 제어는 최소한 현재의 데이터베이스 시스템이 지원하는 수준은 제공되어야 한다. 데이터베이스 시스템에서 작업을 동시에 수행하는 사용자들이 서로 어울리며 제어되어야 한다. 또한 시스템은 공유 객체에 대한 제어와 연산자의 원자성(atomicity) 개념에서 표준을 준수해야 한다.

3.1.12 회복 기능

회복 기능은 최소한 현재의 데이터베이스 시스템

이 지원하는 수준은 제공되어야 한다. 하드웨어와 소프트웨어가 실패하는 경우에 데이터의 손실없이 이전 상태로 회복하는 기능을 제공해야 한다.

3.1.13 특정 질의 기능

질의어의 형태적인 면에서가 아니라 제공되는 기능 면에서 특정 질의어의 기능을 제공해야 한다. 예를 들어 그래픽 브라우저 같은 도구는 비록 질의어가 아니지만 이와 같은 기능을 충분히 가지고 있다. 질의 기능은 쉽고 편리하게 표현되고 how보다는 what을 표현하는 선언적이면서 고급 수준이어야 하고, 처리면에서 질의 최적화 기능이 제공되어 효율적이어야 하며, 가능한 어떠한 데이터베이스에도 적용할 수 있도록 응용프로그램에 좌우되지 않아야 한다.

3.2 선택적인 기능

3.2.1 다중 계승

일반적으로 형질의 계승은 하나의 상위 타입 또는 클래스로부터 형질을 계승받는다. 다중 계승은 하나의 부모가 아니라 다중의 부모로부터 형질을 계승받는 것을 의미한다. 객체지향 분야에서 아직도 다중 계승에 대하여 합의가 이루어지지 않고 있다. 다중 계승에서의 문제점은 충돌의 해결이다. 애틀뷰트나 연산자들의 같은 이름들이 여러 부모로부터 계승받을 경우 어느 부모로부터 계승받을 것이냐 하는 문제가 있다. 이러한 문제의 해결은 계승받을 부모를 지정하거나, 모두로부터 계승받고 이들을 구별하는 기능을 제공하는 방법들이 있다.

3.2.2 타입 체크와 변환

컴파일시(compile-time)에 타입을 체크하는 정도는 언어에 따라 서로 다르지만 많이 할수록 좋다. 타입을 체크하여 올바르지 못한 타입은 적절히 변환해야 한다.

3.2.3 분산 데이터베이스 기능

이 특성은 객체지향 데이터베이스 시스템 기능과 무관하지만 분산 데이터베이스 기능은 객체지향 데이터베이스 시스템에서는 관계형 데이터베이스 시스템보다도 분산 기능이 더 유용하다. 객체지향 데이터베이스 시스템을 잘 활용하기 위하여 분산 기능을 제공해야 한다.

3.2.4 트랜잭션 설계

많은 새로운 응용프로그램에서는 기존의 데이터베이스 시스템에서 제공하는 트랜잭션 모델은 만족스럽지 못하다. 여기서는 트랜잭션은 매우 길어지는 경향이 있고, 일련성(serializability)의 기준도 지키기 어렵다. 그래서 객체지향 데이터베이스 시스템에서는 사용자가 직접 트랜잭션을 길게하거나 중첩되게 설계할 수 있는 기능을 제공할 수 있다.

3.2.5 버 전

대부분의 CAD/CAM과 CASE 같은 새로운 응용프로그램에서는 설계를 하는 작업이 많은데 설계의 각 부분을 여러 형태로 설계하고 이들의 각 버전을 저장하고 서로 조합하여 가장 좋은 설계를 얻는 과정을 거친다. 이러한 버전을 관리하는 기능은 이러한 작업에서는 매우 중요하다.

3.2.6 스키마 변화

데이터베이스는 실세계가 변함에 따라 그 내용 뿐만 아니라 그의 구조도 변하기 마련이다. 이에 대응하여 데이터베이스의 스키마도 적합한 형태로 변신하여 실세계에 적응해야 한다. 이러한 변화에는 클래스의 구조(structure)가 변경될 수도 있고, 클래스를 접근하는 방법 즉 행동(behavior)이 변경될 수도 있고, 클래스들의 계층 구조가 변할 수도 있다. 클래스 구조의 변화에는 애트리뷰트가 새로이 첨가되거나, 애트리뷰트의 이름이나 타입이 변경되거나, 애트리뷰트가 삭제되는 경우들이 있다. 이러한 변화들은 다음과 같다.

(1)클래스 계층 구조에서의 한 클래스의 내용에 대한 변화 변화

[1]애트리뷰트에 대한 변화

①새로운 애트리뷰트를 첨가

②존재하고 있는 애트리뷰트를 삭제

③존재하는 애트리뷰트의 이름을 변경

④존재하는 애트리뷰트의 도메인을 변경

⑤같은 이름으로 다중 계승받을 경우 애트리뷰트를 계승받는 부모를 변경

[2]메쏘드에 대한 변화

①새로운 메쏘드를 첨가
②존재하고 있는 메쏘드를 삭제
③존재하는 메쏘드의 이름을 변경
④존재하는 메쏘드의 코드를 변경
⑤같은 이름의 메쏘드를 다중 계승받는 경우 부모를 변경

(2)클래스 계층 구조의 경로에 대한 변화 변화
1한 클래스를 다른 클래스의 상위 클래스로 첨가
2한 클래스를 다른 클래스의 상위 클래스에서 삭제

(3)클래스 계층 구조에 대한 변화 변화
1한 클래스를 새로이 첨가
2존재하는 한 클래스를 삭제
3한클래스의 이름을 변경

이러한 스키마의 변화에 따라 변화된 클래스에 속하는 모든 객체들은 이 변화에 많은 영향을 받는다.

4. 객체지향 데이터베이스 시스템 설계

본 시스템은 UNIX 운영체계에서 클라이언트-서버(client-server) 구조로 설계한다. 시스템의 개략적인 구조는 크게 세 개의 층으로 구성하는데, 최하층은 저장시스템(storage system)으로 세부 단계의 화일 입출력을 담당한다. 중간층에서는 사용자 인터페이스를 제외한 거의 모든 데이터베이스 기능을 지원하도록 한다. 이 단계에서 주기억장치에서의 객체들을 관리하는 버퍼 기능을 위주로 하여 스키마 관리, 트랜잭션 관리, 동시성 제어, 회복 기능, 버전 관리, 색인 관리, 클러스터링, 그리고 접근 경로(path expression) 등을 지원하도록 한다. 최상위층은 사용자 인터페이스를 위주로 하여 응용프로그램과 객체지향 데이터베이스 시스템 사이를 매개하는 단계로 그림 사용자 인터페이스(graphical user interface), 전처리기(preprocessor), 인터프리터, 객체지향 데이터베이스 관리자 기능 등을 지원하도록 한다. 여기서는 스키마와 관련된 DDL (Data Definiton Languagae)과 데이터를 조작하는 DML(Data manipulation Language)의 기능을 SQL과 C++을 결합하는 형태로 확장하여 표현하고, 이들은 제공되는 도구를 사용하여 스키마를 정의하거나 자료를 조작하도록 한다. 특히 구축된 데이터베이스 자료를 검색할때 가장 널리 사용되고 있는 표준 질의어 SQL(Structured Query Language)을 객체지향적인 개념을 지원하도록 확장된 형태의 객체지향 SQL을 설계한다.

4.1 클라이언트-서버 프로세스 구조

사용자의 응용프로그램, 객체층, 시스템 클래스 라이브러리가 함께 링킹되어 하나의 클라이언트 프로세스가 되고, 저장층이 서버 프로세스가 되어 객체들에 대한 화일 입출력을 담당한다. 이들 두 프로세스의 통신은 TCP/IP를 이용한 통신모듈이 객체 단위로 전달된다. 클라이언트 프로세스와 서버 프로세스는 같은 컴퓨터 상에 수행될 수도 있고, 서로 다른 컴퓨터 상에서도 수행될 수가 있다. 서버 프로세스는 데이터베이스가 구축되어 있는 서버 컴퓨터에서만 수행된다. 클라이언트 컴퓨터에서 클라이언트 프로세스가 서버 컴퓨터에 연결을 요구하면 서버 컴퓨터의 데몬(daemon) 프로세스가 해당 클라이언트 프로세스를 위한 서버 프러세서를 생성(fork)해 두 프로세스가 서로 대응되고, 서버 프로세스들은 공유 메모리(shared memory)의 페이지 버퍼(page buffer)를 통하여 데이터를 공유한다. 이들의 데이터에 대한 동시성 제어는 공유 메모리에 있는 잠금 테이블(lock table)을 이용한다.

4.2 저장 관리자

최근 객체지향 개념을 가진 EXODUS 같은 저장시스템들이 많이 개발되고 있다. 일반적인 저장시스템에서 객체지향 개념을 갖는 저장시스템을 지원할 때 객체 식별자는 저장시스템에서 주로 사용되고 있는 RID 개념으로 쉽게 해결되지만, 단지 다중매체를 지원할 수 있는 대용량의 자료를 저장하는 문제만 해결되면 이러한 객체지향 개념을 가지고 있지 않은 저장시스템이더라도 객체지향 데이터베이스 시스템의 저장시스템으로 사용하는데 큰 문제점이 없다. WiSS 같은 최근의 저장시스템들은 긴 레코드(long data item)를 지원하고 있으므로 충분히 객체지향 개념을 가지도록 수정하여 사용할 수가 있다.

저장 관리자는 일반적인 동시성 제어, 회복 기능은 객체지향 데이터베이스 시스템을 위하여 특별히 수정하지 않고 저장시스템을 그대로 사용하더라도 아무런 문제가 없다. 이 저장시스템에서 지원하는 페이지 단위의 록(lock)은 록 관리자(lock manager)를 이용

```
schema class OBJECT {
private:
    OID identifier;
public:
   // 활성화 및 비활성화에 관한 연산
   static char* wrapper();                    // 객체를 위한 메모리 확보
   statuc void setOffset(Activation&);        // 지속성 객체 활성화 정보
   // 생성자와 소멸자
   OBJECT();                                  // 생성자
   OBJECT(MemoryClassDesc*);                  // 메모리 정보에 의한 생성
   OBJECT(Activation&);                       // 스키마 정보에 의한 생성
   virtual ~OBJECT();                         // 소멸자
   // 타입변환에 관한 연산
   OID getIdentifier();                       // 객체의 OID로 타입변환
   BOOL setIdentifier(const OID*);            // 객체의 OID를 설정
   void* getBodyAddress();                    // 객체 관리자에서 객체주소
   operator OID();                            // OID로 타입변환의 연산자
   // 동일성 및 동치성에 관한 연산
   BOOL operator ==(OID);                     // OID의 객체와 동일한가?
   BOOL operator !=(OID);                     // 동일한 객체가 아닌가?
   BOOL shallowEqual(OID);                    // 두 객체가 같은 값인가?
   // 스키마에 관한 연산
   void* as(MemoryClassDesc*);                // 해당 클래스로 타입변환
   MemoryClassDesc* classDescriptor();        // 객체 스키마의 설명자
   char* className();                         // 객체의 클래스 이름
   char* databaseName();                      // 객체의 데이터베이스 이름
   BOOL isInstanceOf(MemoryClassDesc*);       // 해당 클래스의 객체인가?
   BOOL isDrivedOf(MemoryClassDesc*);         // 하위 클래스의 객체인가?
   // 객체의 버퍼에 관한 연산
   BOOL pin();                                // 스워핑 우선순위가 낮게
   BOOL unpin();                              // 스워핑 우선순위가 높게
   BOOL dirty();                              // 객체가 수정되었음
   // 지속성에 관한 연산
   BOOL isPersistent();                       // 지속성 객체인가?
   virtual BOOL makePersistent();             // 지속성 객체로 저장
   // 능동성에 관한 연산
   virtual BOOL ruleCheck();                  // rule을 check
   virtual BOOL integrityCheck();             // integrity를 check
   virtual BOOL triggerCheck(int = 0);        // trigger를 check
   virtual BOOL constraintCheck();            // constraint를 check
   // 객체의 삭제에 관한 연산
   friend BOOL deleteObject(OID*);            // OID의 객체를 삭제
   friend BOOL deleteObject(OBJECT*);         // OBJECT의 객체를 삭제
   friend BOOL deleteObject(char*);           // 해당 이름의 객체를 삭제
   friend BOOL deleteMemoryObject(OID*);      // OID 객체를 버퍼에서 삭제
  };
```

(그림 1) 루트 클래스의 OBJECT

하여 다중 사용자를 제어하고, 교착상태 관리자(deadlock manager)는 교착상태를 인식하고 이러한 교착상태를 해결할 수 있으며, 로그 관리자(log manager)와 쉐도우 관리자(shadow manager)는 회복을 위한 정보들을 수집 관리하며, 회복 관리자(recovery manager)는 이러한 정보들을 이용하여 트랜잭션 실패와 시스템 실패 후에 회복 기능들을 거의 객체지향 데이터베이스 시스템에서 이용하면 된다.

4.3 객체 관리자

응용프로그램과 질의처리기는 데이터베이스 객체들에 대하여 생성, 수정, 삭제, 지속성 부여 등의 데이터베이스 연산들과 복사, 동일성 검사, 동치성 검사 등의 객체지향 연산을 지원하도록 한다. 이 연산은 클래스 계층구조의 루트 클래스인 OBJECT를 통하여 형질을 계승받는데 그 정의는 다음 (그림 1)과 같다.

여기에서 기본적으로 제공되는 연산자들은 활성화 및 비활성화에 관한 연산, 타입변환에 관한 연산, 동일성 및 동치성에 관한 연산, 스키마 관리에 대한 연산, 버퍼 관리에 대한 연산, 지속성에 관한 연산, 능동성에 관한 연산, 삭제에 관한 연산 등으로 구성된다. 모든 데이터베이스 객체에 지정되는 객체 식별자 OID는 데이터베이스 식별자, 클래스 식별자, 인스탄스 식별자로 구성되는데 각각 객체가 소속된 데이터베이스, 특정 데이터베이스 내의 해당 클래스, 그리고 해당 클래스에 속하는 특정 객체들의 고유 번호를 의미한다. 이러한 OID들은 여러 데이터베이스 상에서 유일성을 보장하며, 각각의 데이터베이스 객체는 자신의 OID를 통하여 항상 접근이 가능하다. 객체들을 버퍼 상에서 관리하는 객체관리자는 데이터베이스의 객체들이 접근될 때마다 버퍼에 존재하는지 조사하고 없으면 버퍼로 읽은 후에 접근한다. 그러므로 응용프로그램에서 데이터베이스 객체를 접근하는 코드는 전처리기에서 오버로딩 기법을 사용하여 이러한 접근과정을 따르도록 코드를 자동적으로 변환하고, 객체지향 질의 처리기에서는 질의 처리과정에서 이러한 절차를 따라 처리함으로써 사용자는 아무런 부담없이 사용하도록 한다. 객체 관리자는 많은 객체들을 Fix/Unfix 프로토콜을 기반으로 하고 clock 알고리즘을 응용하여 객체대체(object switching) 기법으로 객체들의 버퍼를 운영한다.

4.4 사용자 인터페이스

객체지향 데이터베이스 시스템에서의 사용자 인터페이스는 적어도 객체지향적인 DDL과 DML의 기능, 데이터베이스를 포함하는 응용프로그램을 작성하는 데 필요한 언어와 처리기, 객체지향 데이터를 쉽게 볼 수 있게 하는 도구 등을 제공해야 한다. 이러한 객체지향적인 스키마를 정의하는 언어, 객체지향 데이터를 검색하거나 조작하는 언어, 그리고 응용프로그램을 개발하기 위하여 사용하는 언어는 객체지향 언어인 C++를 확장하여 사용한다.

4.4.1 스키마 관리

스키마 정의는 데이터베이스 관리자가 스키마 관리도구(schema tool)를 사용하여 관계형에서 테이블과 뷰에 해당하는 class를 정의하고, 이를 패싱하여 객체지향 데이터베이스의 사전에 등록하여 객체들을 다루는데 모든 정보를 제공할 수 있도록 한다. 이 class의 정의에 애트리뷰트와 메쏘드는 C++에서의 형식을 그대로 사용하여 표현하고, DDL에 있는 스키마의 기능을 추가적으로 부여함으로써 객체지향 스키마를 정의한다. 스키마를 정의하거나, 스키마의 내용을 변경하거나, 스키마를 삭제하는데 데이터베이스 관리자가 스키마 관리도구를 사용하여 처리한다. 처리된 내용은 스키마 관리자(schema manager)에 의하여 스키마의 생성, 변경, 그리고 삭제의 기능을 수행한다. 스키마가 정의되면 응용프로그램에서 이들의 정보를 이용할 수 있도록 class를 정의하는 헤드 화일과 이 클래스의 메쏘드 화일을 각 스키마에 대하여 생성된다. 그리고 메쏘드 화일은 지속적인 객체를 접근하는 코드에 대해서는 디스크에 있는 객체를 접근하도록 변환한 후 C++ 컴파일러에 의하여 목적 화일로 컴파일되어 데이터베이스 스키마들의 메쏘드들의 라이브러리로 구축된다. 스키마가 생성되면 각각의 타입들은 계층구조를 형성하는데 스카마 관리도구는 이 구조를 그래픽으로 보여주고 관리자에게 이들에 대한 연산을 쉽게 하도록 도와준다.

4.4.2 객체지향 질의처리기

DBMS에는 사용자에게 보통 특정의 질의어(ad hoc query)와 프로그래밍 언어의 인터페이스를 제공한다. 이러한 환경에서는 질의문이 프로그래밍 언어에 포함

되어 서로 다른 DBMS 질의어와 프로그래밍 언어 환경 사이에서 데이터가 복사 되는데 이런 과정에서 두 언어 사이의 타입과 계산 모델이 서로 달라 결합 불일치(impedance mismatch)가 발생한다. 여기서는 현재 많이 사용되는 객체 지향 언어인 C++를 질의 기능을 가지도록 확장함으로써 같은 타입 시스템과 데이터 공간을 사용하고 하나의 응용 프로그램 실행 환경을 제공하는 데이터베이스 프로그래밍 언어를 사용한다. 이 언어는 같은 프로세스에서 실행되고 DBMS와 주소 공간이 동일하다는 점에서 일반 프로그램에 내장된 질의어를 가지는 응용 프로그램과 다르다. 객체지향 데이터베이스 시스템을 지원하는 데이터베이스 프로그램 언어에서 데이터베이스를 위한 인터페이스는 단순한 질의와 복잡한 질의를 적절하게 처리하는 두 방법을 지원한다. 이 두 방법은 단순한 질의와 SQL 형태의 일반적인 질의를 적절하게 처리하기 위하여 사용된다.

첫째로 일반적인 SQL형태가 아닌 단순한 형태의 질의로 객체들의 한 집합에 적용되어 각 원소들에 대한 단순한 서술문(predicate) 형태의 질의를 쉽게 처리하기 위하여 설계되었다. 그러므로 이러한 질의의 처리는 Collection 클래스에서 원소함수로 정의된 query를 사용하여 표현하는데 이 원소함수는 다음과 같이 정의된다[ATW93].

```
int Collection::query(Collection<T>&result, const
char* simpleQury) const;
```

여기서 첫 번째 매개변수 Collection<T>는 결과 클래스 타입 T에 속하는 질의 결과 객체들의 집합이고, 두 번째 매개변수 simpleQuery는 질의를 표현하는 서술문을 의미한다. 질의 대상(target) 집합에 query가 적용되면 이 대상 집합의 모든 원소에 대하여 simpleQuery를 적용하여 참이 되는 객체들을 결과 집합의 원소로 한다. 간단한 예를 들면 다음과 같다.

```
Collection<Person> queryResultSet;
Persons→query(&queryResultSet, "this→age>20");
```

이 예는 Person 타입의 객체들의 집합인 Persons에 대하여 각 원소에 해당하는 객체들의 나이가 20보다 더 큰 사람을 찾는 질의는 아래와 같이 간단히 표현된다.

둘째로 일반적인 SQL 형태의 SELECT-FROM-WHERE 형식에서 객체지향적으로 확장된 질의를 처리하는 방법이다. 객체지향 질의어는 아직 표준적으로 공인된 질의어가 없기 때문에 어떠한 형태이든 향후에 표준으로 사용될 질의를 표현하는 스트링(string)의 질의 표현을 사용하여 oql이라는 함수가 이를 처리하도록 다음과 같이 설계한다.

```
int oql(<T> &resultObject, const char* queryExpres-
sion, ...);
```

이러한 형태의 질의는 질의처리 함수 oql이 queryExpression를 처리하여 그 결과를 resultObject에 반영한다. 호스트 프로그램의 변수를 사용하고 가변 길이의 매개변수 ...을 지원한다. 이러한 질의에 표현하기 위해서는 그 변수가 호스트 언어의 변수를 의미하는 지시자 $와 대응되는 숫자를 표시해야 한다. 그리고 함수 oql은 질의의 결과로 생성된 객체들의 수를 반환하는데 음수는 에러를 의미한다. 그리고 또 다른 형태로 다음과 같은 표현들을 지원한다.

```
int oql(Collection<T> &resultSet, const char* query-
Expression, ...);
int oql(int&, const char* queryExpression, ...);
int oql(char&, const char* queryExpression, ...);
int oql(double&, const char* queryExpression, ...);
int oql(char*&, const char* queryExpression, ...);
```

이러한 형태의 질의는 질의의 결과가 집합, 정수, 문자, 실수, 또는 스트링이 된다. 예를 들어

```
Collection<Person> resultSet;
int bound = ·····;
·····
int ctr = oql(&resultSet,
        "SELECT Pair(), relatePair(m, w) \
         FROM m IN Man, m IN Woman \
         WHERE m.age>$1 AND m.age
         =w.estimatedAge()", bound);
```

이 질의에서 Pair 클래스의 생성자 Pair에 의하여 한 객체를 만들고 이 객체에 연산자 relatePair()을 호출한다. 그리고 호스트 언어의 변수 bound는 $1을 대신하여 사용되었다. 그리고 질의의 결과 집합인 resultSet에 집합 연산자 Iterator를 사용하여 집합 단위의 연산(set-at-a-time)을 할 수 있고 이 집합에 반복자를 만들어 원소 객체에 개별적으로 연산(record-at-a-time)을 할 수도 있다.

객체지향 개념을 지원하는 표준으로 공인된 SQL이 아직은 없기 때문에 SQL의 검색 연산 SELECT에 객체지향적으로 간단히 확장되는 내용은 다음과 같다. BNF에 나오는 commalist와 dotlist는 쉼표와 마침표로 분리되는 하나 또는 그 이상의 리스트(list)를 의미한다.

```
query-spec ::= SELECT [ALL|DISTINCT] selection
                table-exp
selection ::= scalar-exp-commalist|*
table-exp ::= from-clause [where-clause] [group-by-clause]
              [having-clause]
column-ref ::= [column-qualifier .] name
```

여기서 selection은 조인된 결과의 scalar-exp이 프로젝션(projection)되는 목적 리스트(target list)인데 객체지향 개념을 지원하기 위하여 아래와 같이 확장된다. 이것은 질의의 결과가 객체 지향적으로 처리되도록 생성자에 의하여 객체를 조인된 객체들로부터 질의 객체의 생성 절차를 지정한다. 이 절차는 최종적으로 질의를 실체화 하는 과정에서 일어난다.

```
selection ::= target-class, statement-commalist|*
target-class ::= taget-class-name|target-class-constructor
statement ::= [path-name-dotlist .] attribute = scalar-exp
              |[path-name-dotlist .] member-function-call
column-ref : := [path-name-dotlist .] attribute
              |[path-name-dotlist .] member-function-call
              |external-function-call
member-function-call ::= member-function-ref
                         ([scalar-exp-commalist])
external-function-call ::= external-function-ref
                           ([scalar-exp-commalist])
```

질의의 결과로 생성되는 객체는 질의에 의하여 조인된 객체들을 매개체로 하여 결과 클래스의 생성자로 생성시킨다. 그러므로 위의 BNF에서 selection은 결과 클래스의 생성자에 해당하는 표현으로 확장되었다. 그래서 selection은 객체를 생성하는 절차를 의미한다. 그 밖의 객체의 상태를 결정하기 위하여 연산자를 호출하거나, public 가시도를 가는 애트리뷰트는 대입문(assignment)을 사용한다. 이와 같은 생성자와 연산자의 전달인자(parameter)와 대입문의 오른편 수식으로 질의에 의하여 조인 객체들을 사용할 수 있다.

이 외에도 경로 수식(path expression)에 의한 여러 단계 참조, 원소 함수를 호출함으로써 질차에 의하여 계산된 값에 해당하는 가상 애트리뷰트, 외부 함수 호출 등을 표현할 수 있도록 column-ref도 확장된다. 원래의 column-ref는 테이블의 열(column) 이름이다. 가상 애트리뷰트(virtual attribute), 경로 수식, 외부 함수 호출 등을 지원하도록 확장한다. 질의의 결과는 객체들의 집합이므로 시스템 클래스 Set의 연산자를 사용하여 원하는 연산을 계속할 수 있다.

4.4.3 전처리기

스키마 관리자(schema manager)가 관리하는 정보는 객체의 지속성 문제와 관련하여 객체를 생성하거나, 데이터베이스 객체가 아닌 C++ 객체에 지속성을 부여하거나, 데이터베이스에서 객체를 주기억장치에 올려 객체 관리자에 등록되어 유지되도록 확장된 C++로 작성된 응용프로그램을 C++로 변환하는 전처리기에서 주로 사용된다.

응용프로그램은 일반적으로 먼저 서버와 연결을 하고, 원하는 데이터베이스를 open하고, 트랜잭션을 시작시키고, 데이터베이스에 저장된 객체들에 대하여 접근 여부를 결정하고, 원하는 객체에 대하여 lock을 건 후에, 그 객체를 가지고와서 이들을 조작하고, 다시 데이터베이스에 기록을 하고, lock을 푼 후에, 해당 데이터베이스를 닫고, 서버와 연결을 해제하는 순으로 진행된다. 이처럼 응용프로그램들은 데이터베이스에 대한 인터페이스 함수들과 데이터베이스의 지속성 객체들을 접근하는 부분으로 되어 있다. 이러한 객체들을 일반적인 C++ 객체와 구별없이 사용할 수 있도록 스키마 관리도구에서 생성된 스키마들의 헤드 파일들을 include하고, 지속성 객체들은 오버로

드하여 자동적으로 fetch가 되게 코드를 전처리기에 의하여 변환하고, 메쏘드들은 스키마 라이브러리와 함께 링크되게 한다.

결과적으로 전처리기는 응용프로그램과 스키마 헤드 화일을 입력으로 받아 어휘분석 과정을 거쳐 지속성 객체들이 오버로드 되는 C++로 변환한다. 이것은 다시 C++ 컴파일러에 의하여 컴파일되고 목적코드가 생성되고, 마지막으로 스키마의 메쏘드와 함께 링크되어 응용프로그램이 완성된다.

4.4.4 객체 브라우저

객체 브라우저는 저장된 데이터베이스에 속한 한 클래스의 객체들을 화면에 테이블 형태로 보여주는 사용자 인터페이스 도구이다. 객체 브라우저가 생성되면 트랜잭션이 시작되면서 해당 클래스의 외연에 속하는 모든 객체들을 원소로 하는 임시적 시스템 클래스인 Collection 클래스의 객체들과 전반적인 정보가 함께 생성된다. 이 집합 객체는 원소 객체들의 OID를 유지하는데 iterator를 사용하여 앞 또는 뒤 버튼을 눌러 계속적으로 살펴보게 한다. 브라우저에서는 한 객체의 필드가 다른 객체에 대한 참조 즉 OID이거나 복합 객체에 대해서는 값 대신에 링크를 표시하고 이 링크를 사용하여 점진적으로 관련된 객체를 살펴보게 한다. 그리고 끝내는 버튼을 누름으로써 트랜잭션을 끝내고 객체 브라우저가 끝나게 된다.

5. 결 론

객체지향 데이터베이스 시스템은 데이터베이스 시스템으로서의 기준과 객체지향 프로그래밍 언어에서 제공되고 있는 객체지향 시스템으로서의 기준을 만족해야 한다. 객체지향 데이터베이스 시스템의 원리를 이론적으로 고찰한바 시스템으로서 갖추어야 할 필요 요건으로 전자의 기준은 지속성, 보조기억 장치의 관리, 동시성 제어, 회복 기능, 그리고 특정 질의 기능과 같은 다섯 가지의 기능을 포함하고, 후자의 기능은 복합 객체, 객체 식별자, 캡슐화, 타입 또는 클래스의 계층적인 구조, 속성 계승, 늦은 바인딩과 결합된 오버라이딩, 확장성, 그리고 계산 표현의 완전성과 같은 여덟 가지의 기능을 포함하고 있음을 알 수 있다. 본 논문에서는 이를 바탕으로 객체지향 데이터베이스 시스템을 C++을 기반으로 하여 시스템 구조를 개략적으로 클라이언트-서버 구조로 저장 관리자, 사용자 인터페이스로 스키마 관리자, 객체지향 질의 처리기, 전처리기, 객체 브라우저를 비롯하여 객체 관리자 설계를 보여 주었다.

참 고 문 헌

[AFS85] Afsarmanesh, H., et al., "An Object-oriented Approach to VLSI/CAD", Proc. of VLDB, Aug. 1985.

[AHL84] Ahlsen, M., et al., "An Architecture for Object Management in OIS", ACM Trans. on Office Information Systems, Jul. 1984.

[ATW93] Atwood, T., et al., "The Object database standard, ODMG-93", R.G.G. Cattell, 1993.

[BAN87] Banajee, J., et al., "Data Model Issues for Objected-oriented Applications", ACM Trans. on Office Information Systems, Apr. 1987.

[GOL83] Goldberg, A., et al., "smalltalk-80:The Language and its Implementation", Addison-Wesley, Reading, MA 1983.

[HAL76] Hall, P., et al., "Relations and Entities", Modeling in Database Management Systems, North-Holland, 1976.

[KIM88] Kim, W., et al. "Integrating an Object-Oriented Programming System with a Database System", in Pro. 3rd Intl. Conf. on Object-Oriented Programming System, Languages, and Application, San Diego, Calif Sept. 1988.

[KIM89] Kim, W., E. Bertino,and J, Garza. "Composite Revisited", in Proc. ACM SIGMOD Intl. Conf. on Management of Data, Portland,Oregon, June 1989.

[MAI86] Maier, D., et al, "Development of an Object-oriented DBMS", Proc. of ACM OOSPLA, 1986.

[MOO89] Moon, D. "The Common LISP Object-Oriented Programming Standard System,"

Object-Oriented Concepts, Applications, and Databases,(ed.W.Kim, and F.Lochovsky), Addison-Wesley, 1989.

[NYG86] Nyggard, K., etal., "SIMULA－An Algol-based Simulation Language", CACM, 1966.

[STE86] Stefik, M., et al., "Object-oriented Programming: Themes and Variations", The AI Magazine, Jan. 1986.

[STR86] Stroustrup, B. The C++ Programmiing Language, Addison-Wesley, Reading, Mass, 1986.

[STR88] Stroustrup, B. "What Object-Oriented Programming?", IEEE Software, May 1988, pp. 10-20.

[TOM89] Tomlinson, C., M. Scheevel, and W. Kim. "Sharing and Organization Protocols in Object-Oriented Systems", Journal of Object-Oriented Programming, Nov./Dec. 1989.

유 양 근

- 서울교대, 강남대(학사)
- 한양대 대학원(석사)
- 단국대 대학원(이학석사)(전산학)
- 단국대 대학원(박사과정 수료)(전산학)
- 한국정보관리학회(종신회원, 학술위원, 편집위원)
- 강남대학교 전산소 소장
- 전국대학 정보전산기관협의회 상임이사
- 전자도서관 연구회 자문위원
- 한국정보처리학회(종신회원)
- 강남대학교 문헌정보학과 부교수

관심분야: 정보처리 및 전산교육, 소프트웨어 공학, 데이타베이스, Digital Library

류 해 영

- 단국대 수학과(학사)
- 단국대 대학원(이학석사)(응용수학)
- 아주대 대학원(박사과정 수료)(컴퓨터공학)
- 한국정보과학회(종신회원)
- 단국대학교 전산통계학과 교수
- 한국정보처리학회(정회원)

관심분야: 소프트웨어 공학, 데이타베이스, 정보공학