□ 기술해설 □

기술 및 시장 동향

객체지향 데이타베이스 시스템

한국과학기술원 이 윤 준*

목 차

과거 여러해 동안 객체지향(object-oriented) 기술은 프로그래밍 언어, 사용자 인터페이스, 데이타베이스, 운영체제, 전문가 시스템 등에서 사용되어 왔다. 그동안 객체지향 데이타베이스라는 제품들이 시장에 발표되었으며, 또한 관계형 데이타베이스 시스템을 개발 판매하는 회사들도 그들 제품에 객체지향 기능을 추가하였다고 발표하였다. 특히 몇몇 회사는 현재 관계형 기능과 객체지향 기능을 결합한 시스템을 내놓았다. 이 글에서는 객체지향 데이타베이스 시스템의 기술과, 국내에서 판매되고 있는 시스템들을 소개하므로써 독자들에게 차세대 데이타베이스 기술로서의 객체지향 데이타베이스 기술에 대한 이해를 돕고자 하는 것이다.

1. 객체지향 데이타베이스 시스템의 소개 및 정의

과거 여러해 동안 객체지향 개념은 전산학 분야에서 특히 데이타베이스, 프로그램언어, 지식표현, 그리고 컴퓨터 구조 분야에서 가장 중요한 연구토픽이 되어오고 있다. 사실 객체지향 개념은 위에서 나열한 학문분야의 공통 부분이며, 이는 앞으로 intelligent high-performance programming 시스템의 한 부류를 이루는데 중요한 열쇠가 될 것으로 예측된다.

그러나 위에서 언급한 분야에서 객체지향 개념이 부각되고 있는 반면에 일반적으로 객체지향이 무엇을 의미하고 객체지향 데이타베이스가 무엇인지에 대해서는 매우 혼란한 상태에 있는 실정이다. 사실 기본적인 객체지향 개념은 프로그래밍 언어 분야와 인공지능 분야, 그리고 데이타베이스 분야에서 전개되어온 것이 사실이나 불행하게도 이러한 분야에서 객체가 정확히 무엇이다라고 하는 의견일치를 보지 못하고 있는 실정이다.

Simula-67이 처음으로 객체지향 개념을 갖는 프로그래밍 언어로 발표된 이후 프로그래밍 언어 분야에서의 객체지향 프로그래밍은 두가지의 다

* 종신회원

1) 본절은 UniSQL제품의 국내 판매자인 한국컴퓨터통신(주)의 최홍식님께서 작성하셨습니다.

2) 본절은 VERSANT 제품의 국내 판매자인 (주)다우기술의 최인호님과 김윤덕님께서 작성하셨습니다.

른 방향으로 연구가 진행되어 왔다. 첫번째는 새로운 객체지향 언어를 개발하는 것이었다. 대표적인 것으로는 Smalltalk, Traits, Eiffel, Trellis/Owl 등이 있다. 두번째로는 기존 언어를 확장하는 것이었다. 예를 들면 LISP를 확장한 Flavors, Object LISP, OakLisp, LOOPS, 그리고 Common LOOPS들이 있고, C언어를 확장한 object C와 C++이 있다. Pascal을 확장한 Clascal도 개발되었다. 지식을 표현하는 방법으로 Minsky의 프레임에 관한 개념이 소개된 이후로 인공지능 연구분야에서는 KEE나 ART와 같은 프레임-기반 지식 표현 언어가 개발되었다. 데이타베이스 분야에서는 시멘틱 데이타 모델에 객체지향 개념을 심는 연구가 진행되어 왔다. 가장 잘 알려진 시멘틱 데이타 모델로 E/R, SDM, 그리고 DAPLEX가 있다.

일반적으로 데이타베이스의 데이타 모델이라는 것은 실세계의 객체 또는 엔티티들의 연관관계 및 이들의 구성요소들을 논리적으로 구성하는 것을 의미한다. 또한 데이타베이스 언어는 이러한 데이타 모델을 표현할 수 있는 문법을 갖춘 언어이다. 데이타베이스 시스템은 이러한 데이타 모델을 구현한 것이라 말할 수 있을 것이다. 이는 관계형 데이타베이스 시스템은 데이타의 관계형 모델을 구현한 데이타베이스 시스템이고, 객체지향 데이타베이스는 객체지향 데이타 모델을 구현한 데이타베이스 시스템이다.

현재 객체지향 데이타 모델에 대한 표준이 정립되지 않은 상태에서 코아 객체지향 개념을 기술하고자 한다. 물론 객체지향 데이타베이스 시스템은 기존의 다른 데이타베이스 시스템과 같이 객체들에 대한 지속성 있는 저장장소와 스키마를 공급해야만 한다. 또한 이 시스템은 스키마 정의를 위한 사용자 인터페이스와 객체들을 생성하고, 수정 검색이 가능하도록 되어야 한다.

기본 시스템은 과거 수년동안 관계형 데이타베이스 시스템에 여러가지 종류의 특징들 및 기능이 추가된 것과 마찬가지로 여러가지 범위에서 확장시킬 수 있을 것이다. 첫째, 질의 언어는 기본 시스템의 일부분으로 정의되어야 한다. 둘째, 트랜잭션 관리나 트리거와 같은 일관성 유지 기능들이 추가되어야 한다. 셋째, 2차 인덱싱 및 클러스터링과 같은 성능에 관련된 기능이 추가되어야 한다. 넷째, 다중 사용자 환경하에서 동시성 제어 및 권한부여 기능들이 확장되어야 한다. 다섯째, 응용 모델링을 단순화 시킬 수 있고, 복합 객체들을 쉽게 모델링할 수 있어야 하며, 객체들에 대한 버전들을 표현할 수 있는 시멘틱 데이타 모델 개념이 기본 시스템에 추가로 정의되어야 한다.

대부분의 객체지향 프로그래밍 언어와 지식 표현 언어에서 공통이 되는 객체 개념들에 근거하고 있는 코아 객체지향 데이타 모델을 규정하고 소개한다.

객체와 객체 식별자 : 객체지향 시스템과 언어안에서는 실세계에서의 어떠한 엔티티라도 객체로서 모델링이 되어야한다. 이 객체는 한개의 유일한 식별자를 갖고 있어야 한다.

애트리뷰트와 메소드 : 모든 객체들은 어떠한 상태(state)와 행동(behavior)를 갖는다. 객체의 상태는 객체의 애트리뷰트들에 대한 값들의 집합이다. 그리고 객체의 행동은 객체의 상태를 조정하는 메소드(이는 프로그램 코드이다)들의 집합이다. 어떠한 객체의 한 애트리뷰트의 값도 그 객체의 소유권한을 갖는 객체이다. 한 애트리뷰트는 한개의 값 또는 값들의 집합이다. 객체안에 캡슐화 되어 있는 상태와 행동은 외부의 다른 객체로부터 메세지 전달(또는 함수 호출)을 통하여 불러 내거나 억세스할 수 있다.

클래스 : 클래스는 애트리뷰트와 메소드들의 같은 집합을 공유하는 모든 객체들을 모아 놓은 것으로 정의한다. 어떠한 객체도 그 객체를 포함하는 클래스가 존재하며, 객체는 그 클래스의 인스턴스가 된다. 객체와 클래스간의 관계는 instance-of 관계이다. 클래스는 사용자정의 데이타 타입이다. 클래스는 또한 프리미티브 타입이 될 수 있다. 그러나 integer, string, 그리고 boolean과 같은 프리미티브 클래스는 애트리뷰트를 갖지 않는다. 객체의 애트리뷰트들의 값은 어떠한 클래스 범위에 포함된다. 이때 이 클래스는 그 객체의 애트리뷰트의 도메인(domain)이라 부른다.

클래스 계층과 계승 : 객체지향 시스템은 존재하는 클래스로부터 사용자가 새로운 클래스를

생성할 수 있도록 허용한다. 이때 이 새로운 클래스는 존재하는 클래스의 하위클래스라고 부른다. 이 새로운 클래스는 상위클래스의 모든 애트리뷰트와 메소드들을 계승받을 수 있다. 클래스는 많은 하위클래스를 거느릴 수 있다. 때로는 어떠한 시스템은 클래스에 대하여 오직 한 상위클래스만을 허용하기도 한다. 이는 오직 한 클래스로부터 애트리뷰트와 메소드들을 계승받기 때문에 이를 단일 계승(single inheritance)이라 부른다. 단일계승을 제공하는 시스템 안에서는 클래스들은 트리를 구성한다. 반면에 다승 계승을 제공하는 시스템에서는 클래스는 뿌리를 갖는 방향성 그래프(directed graph)를 형성한다.

2. 객체지향 데이타베이스 시스템의 잠재성

객체지향 프로그래밍 언어(OOPL)는 객체들을 조직화하기 위해 클래스를 생성하고, 객체를 생성하며, 계승 계층을 통해 하위 클래스가 상위클래스(superclass)로부터 애트리뷰트와 메소드를 계승 받을 수 있도록 하며, 특정 객체를 접근하는 메소드를 호출하는 기능을 지원한다. OODB는 이와 같은 기능을 지원함과 동시에 오늘날의 관계형 데이타베이스 시스템(RDB)이 지원하는 표준기능을 지원해야 한다. 이러한 표준기능으로는 자료검색을 위한 비절차식(nonprocedural)질의 기능, 자동 질의 최적화, 동적 스키마 변경(클래스 정의 변경 및 계승 구조 변경). 질의 처리 성능 향상을 위한 접근기법의 관리(예. B^{+-}트리, 확장 해쉬, 정렬 등), 트랜잭션 관리. 동시성 제어, 시스템 고장으로부터 회복, 보안 및 권한 부여 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 권한 부여, 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 두가지 접근 방식이 있다. 하나는 OOPL로 작성된 프로그램의 자료를 저장, 관리하는 방식으로, 현재 몇 개의 OODB가 C^{++}나 Smalltalk 프로그램의 자료를 저장, 관리하여 준다. 물론 RDB를 이용하여 이러한 객체를 관리할 수 있으나, RDB는 객체의 개념이 없으며, 특히, 메소드와 계승을 지원하지 못한다. 그러므로, 메소드와 계승을 지원하고, 객체를 튜플로 변환할 수 있는 소위, "객체관리자(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 클래스, 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 객체를 데이타베이스 시스템의 객체로 전환하는 단계가 필요하겠지만 이는 RDB에 객체 관리자를 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 응용프로그래머의 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 동안의 화일 시스템에서 관계형 데이타베이스에 이르기까지 자료관리 기술이 추구한 목표이다. OODB는 매우 복잡한 데이타베이스를 설계하고 관리하는 어려움을 줄일 수 있는 잠재성을 갖고 있다.

또 한가지 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 다음에서 더 자세히 거론하겠다.

- RDB에서는 계층형 자료(hierarchical data), 복합중첩 자료(composite nested data) 등이 모두 릴레이션과 튜플로 표현되어야 한다. 게다가 여러 릴레이션에 결쳐 있는 자료를 검색하기 위해서는 비용이 많이 드는 결합연산 (join operation) 을 수행해야 된다. OOPL의 한 애트리뷰트의 자료형은 시스템이 지원하는 원시적 자료형과 사용자가 정의한 자료형(클래스)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 간단한 기호들이다. RDB는 새로운 자료형의 추가에 대비하여 구축되어 있지 않기 때문에, 새로운 자료형을 추가하려면 시스템의 구조로부터 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 클래스로서 생성될 수도 있으며, 심지어 이미 존재하는 클래스의 애트리뷰트와 메소드를 계승받아 하위클래스로서 생성될 수도 있다.
- 객체 캡슐화는 데이타베이스의 자료 뿐만 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.

3. 객체지향 데이타베이스 시스템의 시장 및 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 클래스의 생성, 상위 클래스로부터 애트리뷰트와 메소드의 계승, 각 클래스의 인스턴스 생성, 객체 식별자를 이용한 인스턴스의 추출, 메소드의 수행 등이 있다.

이들 객체지향 데이타베이스 시스템들은 1987년초 부터 판매되기 시작했다. 그러나, 이들 중 대부분은 아직 평가중이거나 시제품의 수준이다. 즉 아직 중대한 응용분야에 심각하게 사용된 적이 없다. 게다가, 꽤 많은 제품이 인위적으로 제품설치 건수를 올리기 위해 무상으로 공급되기도 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++, Smalltalk)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 응용 프로그램을 작성하기 위해 이렇게 서로 크게 다른 두 가지의 언어를 습득하고 사용한다는 것은 상당한 부담이 된다. C++나 Smalltalk은 이미 클래스와 클래스 계층을 정의하는 기능을 지원한다. 실제로 이들은 범용프로그래밍 언어와 데이타베이스 언어를 통합하는 기초로 사용될 수 있다. 초기의 대부분의 OODB는 첫 단계로 클래스나 클래스의 인스턴스에 지속성을 지원하려고 했다. 즉, 클래스나 인스턴스를 보조 기억장치에 저장함으로서 클래스를 정의하고 생성한 프로그램이 종료되더라도 이들을 사용할 수 있도록 했다.

현존하는 대부분의 OODB는 RDB에 비교할 만한 복잡한 질의 기능을 제공하지 못 한다. 이들은 단지 객체를 검색하는 단순한 방법만을 지원한다. 또한 객체지향 데이타베이스 시스템은 자료에 여러가지 제약을 주기도 한다. 대부분의 시스템이 비지속적인 자료가 OID를 가지지 못하게 함으로서 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 질의 처리와 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), 누적함수(예를 들어, MAX, MIN, AVG)와, GROUP BY 기능, 그리고 클래스들 간의 결합 등과 같은 RDB의 기능을 지원하지 못 한다. 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 OODB에서는 그렇지 못하다.
- RDB는 릴레이션에 새로운 애트리뷰트를 추가하거나 기존의 릴레이션을 삭제하는 등의 데이타베이스 스키마에 관련된 기능을 제공한다. 그러나 OODB는 새로운 클래스의 추가를 허용하면서, 이미 존재하는 클래스에 새로운 애트리뷰트나 메소드의 추가, 새로운 상위 클래스의 추가 및 삭제, 클래스의 삭제 등과 같은 데이타베이스 스키마 변경을 지원하지 않는다.
- RDB에서는 시스템 관리자가 시스템 인자를 변경함으로서 성능을 조정할 수 있다. 이와 같은 시스템 인자로는 메모리 버퍼의 크기, 자료의 추가를 위한 잉여 공간의 크기등을 들 수 있다. 그러나 대부분의 OODB는 이와 같은 시스템 성능조정을 위한 충분한 기능을 제공하지 못한다.

3.2 개선점

이상과 같이 열거된 문제점 때문에 현존하는 OODB는 RDB의 기능을 지원할 수 있도록 수정되어야 한다. 그러나, 현재 중대한 응용분야에서 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 클래스 내 객체의 갯수, 애트리뷰트의 최대값과 최소값, 클래스 내 서로 다른 값의 갯수 등을 저장하여야 한다. 또한, 이들 외에도 클래스의 객체가 차지하는 자료 페이지의 갯수, 클래스의 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 일관성 있게 유지할 수 있도록 자료접근 기법을 설계하여야 한다. 즉 접근기법은 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 작업 영역 관리기가 수정되어야 한다. 작업 영역은 사용자가 필요로 하는 객체를 주기억 장치에 관리하기 때문에 객체가 수정되면, 데이타베이스에 저장된 값과 서로 다를 수 있다. 따라서, 질의 처리기는 작업 영역에서 수정된 객체를 고려해야 한다. 예를 들면, 작업 영역에서 수정된 모든 객체를 질의를 처리하기 전에 데이타베이스에 반영한다면 질의 처리기는 가장 최근 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. 작업 영역 관리기를 주기억 장치의 작업영역에서 처리되는 객체에 대한 스키마를 변경할 수 있도록 확장해야 한다. 예를 들어, 어떤 클래스에서 한 애트리뷰트를 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 한

다.

④ 하나의 클래스가 상위 클래스로부터 애트리뷰트와 메소드를 계승받기 때문에 그 상위 클래스의 스키마 변경도 계승되어야 한다. 예를 들어, 상위 클래스에서 한 애트리뷰트가 삭제되면 그 애트리뷰트는 모든 하위 애트리뷰트에서도 삭제되어야 한다. 즉, 클래스의 변경을 위해서는 클래스 자신 뿐만 아니라 하위 클래스까지도 로그해야됨을 의미한다. 따라서 스키마 관리자는 변경된 스키마를 계승하기 위해서 동시성 제어 기능도 포함해야 한다.

OODB의 개발은 기술적인 어려움뿐만 아니라 객체지향 개념에서 유래된 어려움도 있다. 대부분의 OODB는 데이타베이스 시스템이라기 보다는 객체지향 프로그래밍 언어에 지속성을 지원하는 저장 시스템이라고 할 수 있다. 이들은 객체지향 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, 시스템의 고장으로부터 데이타베이스의 일관성을 유지시켜 준다. 또한, 다수의 사용자 환경에서 데이타베이스 일부에 대한 사용권한을 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 안전하게 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 없을 때까지 검색하도록 설계되어야 한다. 따라서, 응용 프로그램에 데이타베이스 시스템의 커서(cursor) 기능을 도입해야 하며, 예측할 수 없는 갯수의 결과를 저장하기 위한 자료구조와 알고리즘이 필요하다. 또한, 중첩 질의, GROUP BY, 누적 함수, 집합 질의 등을 표현하기 위한 기능을 응용 프로그래밍 언어의 문법과 동일하도록 제공해야 한다. 아울러 프로그래밍 언어로서 사용되는 주 프로그래밍 언어(host programming 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.

4. 객체지향 데이타베이스 시스템의 소개

4.1 UniSQL/X

UniSQL/X는 미국 UniSQL사(대표 김원 박사)에서 개발한 워크스테이션용 데이타베이스 시스템으로서 관계형 데이타베이스 시스템과 객체지향 데이타베이스를 하나로 통합한 차세대 데이타베이스 시스템이다. 관계형 데이타베이스 시스템이 갖추고 있는 모든 기능을 수용하면서 객체지향 데이타베이스 기능을 갖춘 시스템이다. 표준 ANSI-SQL을 따르고 여기에 객체지향 개념을 추가시킨 SQL/X를 갖추고 있다. 이는 C 언어를 기반으로 객체지향 언어 C^{++}를 만들었듯이 관계형 데이타베이스를 위한 표준 SQL에 객체지향 개념을 추가하여 SQL^{++}로 명명하는 것과 같은 개념이다. UniSQL사 제품에는 UniSQL/X와 UniSQL/4GE 그리고 UniSQL/M이 있다.

4.1.1 관계형 데이타베이스 시스템의 확장

UniSQL/X에서 기존 관계형 데이타베이스 시스템들이 공통적으로 갖고 있는 기능들을 빼면 객체지향 개념을 위하여 추가된 기능만이 남는다. 지면관계 상이 추가로 확장된 기능들만 설명한다.

확장 1 : 사용자가 임의의 데이타 타입을 정의하여 사용할 수 있다.

이는 데이타 타입으로 숫자, 문자, 날짜, 시간등 시스템에서 제공하는 기본적인 타입뿐만 아니라 사용자가 정의한 임의의 타입을 사용할수 있다(그림 1 참조).

확장 1로부터 얻는 효과는 다음과 같다. 첫째는 주어진 데이타 타입으로만 모델링해야 하는 제약에서 벗어날 수 있다. 이로 인하여 실세계의 현상을 보다 자연스럽게 표현할 수 있다. 둘째는 복잡하게 내포된 데이타(Complex Nested Data)를 모델링할 수 있다. 세째는 테이블 자체를 데이타 타입으로 정의할 수 있으므로 테이블에서 한 행의 열 값이 다른 테이블 그 자체일 수 있다. 세째는 그림 1의 예에서와 같이 "'홍길동'이라는 사람의 봉급이 얼마이고, 그사람의 취미는 무엇이며, 근무처는 어느 도시인가?"라는 질의를 한다면, 관계형 데이타베이스에서는 죠인(JOIN) 연산을 위하여 Employee, Activity, Company, Address 4개 테이블을 접근하여 해당 데이타를 검색한다. 그러나, UniSQL/X에서는 SQL/X를 사용하여 다음과 같이 질의할 수 있다.

```
SELECT 봉급, 취미.명칭, 근무처.주소.도시명
FROM Employee
WHERE 이름='홍길동';
```

관계형 데이타베이스 시스템에서와는 다르게 자원 소모와 성능에 영향이 많은 죠인 연산이 필요없다. "근무처·주소·도시명"과 같은 표현에 기인하며, 이를 경로 표현(Path Expression)이라고 부른다. 이는 3개의 테이블 Employee, Company, Address로부터 얻어지는 항목이다. UniSQL/X는 이를 메인 메모리 오퍼레이션으로 수행한다.

확장 2 : 테이블 한행의 열 값이 한개 이상의 객체일 수 있다.

관계형 데이타베이스 시스템에서 한행의 열 값은 오직 한가지 타입으로 한개의 값만 저장할 수 있다. 따라서 그림 1에서 Employee의 취미가 여럿일 경우 또는 사진이 여러장일 경우에는 새로운 테이블을 생성하여야 한다. 이때 이름을 키로 하여 Employee 테이블과 관계를 설정해야 한다. UniSQL/X는 이러한 제약점을 해소할 수 있도록 확장되었다(그림 2 참조).

확장 2가 주는 효과는 다음과 같다. 예를 들어 어느 한 객체가 여러개의 부품들로 구성되고, 각 부품은 다시 여러개의 부품으로 구성 되는 모델을 쉽고 자연스럽게 표현할 수 있다. 부품을 이루는 객체의 속성(attribute)들은 확장 1에따라 사용자가 정의한 임의의 타입일 수 있다. 따라서 한 객체는 여러가지 복합된 타입을 갖는 객체들로 구성할 수 있다. UniSQL/X는 ANSI-SQL를 확장한 SQL/X를 통하여 복합된 타입을 갖는 객체들을 일관성 있게 관리해 준다.

확장 3 : 테이블 한행의 열 값에 수행되는 프로그램을 정의 등록할 수 있다.

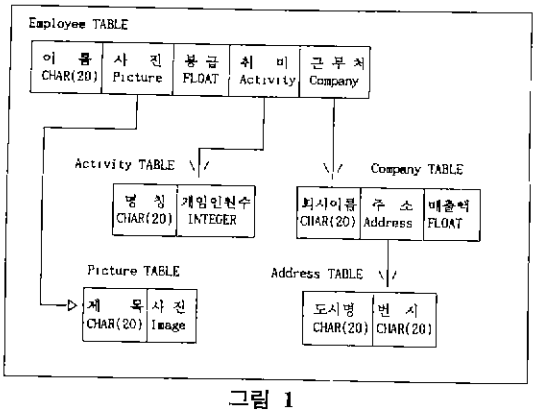


그림 1

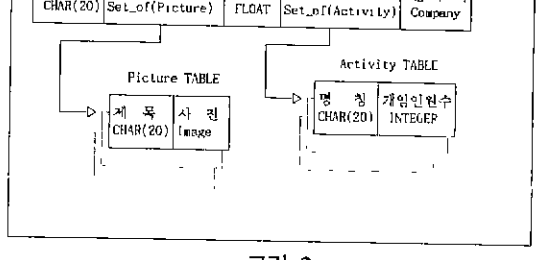


그림 2

```
SELECT 사진 Image to photo FROM Employee WHERE 이름 = '홍길동' ;
CALL display(photo) ON Image .
```

예 1

객체지향 개념에서 객체는 객체의 상태를 표현하는 속성(attribute)과 이 속성을 다루는 메소드(method)로 구성된다. 따라서 객체지향적 데이타베이스는 이러한 객체들을 관리 하는 시스템 이라고 말할 수 있다. 〈예 1〉에서는 Picture 타이블에 이미지를 디스플레이 하는 display function-C 언어로 작성된 프로그램을 메소드로 등록했을 경우 이를 수행하는 SQL/X의 예를 보이고 있다. 예는 Employee 테이블에서 '홍길동' 이라는 사람을 찾아 이사람의 사진을 디스플레이 하는 SQL/X문이다.

확장 3이 주는 효과는 다음과 같다. 객체지향 데이타베이스는 데이타 뿐만 아니라 이를 처리하는 응용 프로그램들도 데이타베이스 내에서 일관성 있게 관리할 수 있고 또한 수행할 수 있다. UniSQL/X는 객체지향 모델을 지원함에 따라 객체를 이루는 속성 및 메소드들을 데이타베이스 내에서 일관성 있게 관리한다.

확장 4 : 같은 종류의 객체들를 클래스로 정의할 수 있고 클래스는 상위 클래스(Superclass)로부터 모든 속성을 상속(Inheritance)받고 하위 클래스(Subclass)로 상속시킬 수 있다.

클래스는 같은 속성을 갖는 객체들을 모아 하나의 공통된 틀(structure)을 정의 한 것이다. 테이블을 클래스로 정의할 수 있으며 하위 클래스는 상위 클래스가 갖고 있는 모든 속성 뿐만 아니라 메소드도 계승 받는다. 한 클래스는 한개 이상의 상위 클래스를 갖을 수 있으며, 그들로부터 속성을 상속받는다(Multiple Inheritance). 하위 클래스와 상위 클래스간의 관계는 IS-A 관계라고 부른다(그림 3 참조).

확장 4가 주는 효과는 다음과 같다. 클래스의 상속 개념이 주는 가장 큰 효과는 예전에 설계된 데이타베이스 및 응용 프로그램를 쉽게 재사용(reuse 개념)할 수 있다는 것이다. UniSQL/X는 데이타베이스에 클래스들로 부터로 같은 속성을 갖는 클래스를 생성할때, 그림 3에서와 같이 단지

```
CREATE CLASS Manager AS SUBCLASS OF  Employee
(관리인원 SET_OF(Employee),
 운영비   MONETARY).

CREATE CLASS Sales_Person  AS SUBCLASS OF Employee
(Quota    MONETARY).

CREATE CLASS Sales_Manager  AS SUBCLASS OF Sales_Person,Manager .
```

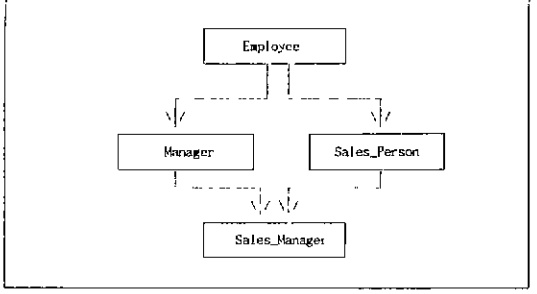


그림 3

'AS SUBCLASS OF'로 정의하고 추가가 필요한 속성만을 정의 하면 된다.

4.1.2 UniSQL/4GE 어플리케이션 개발 환경

지금까지는 멀티미디어 어플리케이션과 객체지향 기술 개발의 이점이 폭넓게 인식되었으나 두 기술이 가지고 있는 복잡성 때문에 어플리케이션을 효과적으로 만들기 어려웠다.

UniSQL/4GE는 어플리케이션 개발자,데이타베이스 설계자 그리고 일반 사용자들에게 객체지향적 데이타 모델링과 멀티미디어 데이타 관리등을 연계시켜 다룰수 있도록 제반 환경을 제공한다. UniSQL/4GE는 비쥬얼에디터(VisualEditor)와 미디어마스터(Media Master) 그리고 오브벡트마스터(ObjectMaster)로 구성된다.

비쥬얼에디터는 UniSQL/X DBMS 스키마 정보와 데이타베이스 정보를 시각적인(OSF/Motif, X-Window,OpenLook) 환경하에서 편집할 수 있는 기능을 제공한다. 즉 클래스의 정의, 스키마 정의 및 생성, 확인 그리고 클래스간의 상속성 관계, 클래스간의 복합 구조화된 계층구조들을 대화식으로 생성, 수정, 탐색할 수 있다.

미디어마스터는 멀티미디어 레포트를 작성하는 도구로써 UniSQL/X 사용자 들에게 WYSIWYG형 편집기를 지원한다. 이미지,텍스트,그래픽 표현과 오디오 데이타를 포함하는 복잡한 보고서를 대화식으로 작성할수 있다.

오브젝트마스터는 X-Window와 OSF/Motif를 기반으로 하는 어플리케이션을 개발할수 있도록 제반 기능을 제공한다. 오브젝트마스터의 주된 특징은 프로그램언어를 이용하지 않고 그리고 GUI 기능을 모르는 사용자들도 원하는 윈도우 어플리케이션을 만들수 있다는 것이다.

끝으로 UniSQL/X는 관계형 데이타베이스의 기능들을 기본으로 하고 있으므로 기존 데이타베이스에 익숙한 사람들이 접근하기 아주 용이하다. 뿐만 아니라 객체지향 모델링이 가능하므로 복잡한 구조의 응용분야에 적용하기에도 적합하다.

4.2 VERSANT

1988년 미국 Versant사가 개발한 VERSANT는 균형있는 클라이언트-서버 구조를 기반으로 분산환경에서 확장가능한 형태로 운용되도록 설계되어 있다(VERSANT Scalable Distributed Architecture). 또한 객체 단위의 로킹을 제공함으로써 효율적인 동시성 제어, 동적인 디스크 관리 및 스키마 확장등의 기능을 제공하며, 데이타베이스의 새로운 응용 분야에서 많이 요구되고 있는 그룹 단위의 작업을 효율적으로 수행하는데 필요한 체크인-체크아웃, 장기 트랜잭션(long transaction), 객체의 버전화 기능등을 제공한다.

VERSANT의 현재 버전에서는 C와 C++, Objectworks—Smalltalk 등의 프로그래밍 언어 인터페이스를 제공하고 있다. 각 프로그래밍 언어 인터페이스는 그 언어의 고유한 모든 기능과 프로그래밍 스타일이 객체지향 데이타베이스 모델에 적합하도록 운용된다. 인터페이스 언어로 표현할 수 있는 모든 행위와 데이타 유형은 VERSANT 데이타베이스 스키마의 일부가 된다. 따라서 프로그래밍 언어는 데이타베이스의 일부가 되나, 데이타베이스는 특정 언어에 종속되지 않는다.

또한 Versant는 사용자의 생산성을 높이기 위한 도구로 VERSANT C++ Application ToolSet을 제공한다. 여기에는 사용자의 화면폼을 그래픽적으로 작성할 수 있는 VERSANT Screen, 사용자의 보고서를 그래픽적으로 작성할 수 있게 해주는 VERSANT Report, 그리고 표준 SQL으로 객체지향 데이타베이스의 질의할 수 있도록 확장한 VERSANT Object SQL과 이러한 OSQL을 C++ 프로그래밍 언어에 내장하여 사용할 수 있도록 한 VERSANT Embedded Object SQL등이 있다.

4.2.1 VERSANT ODBMS 구조

VERSANT 데이타베이스 시스템은 다중-클라이언트/다중-서버 구조를 갖는다. 실제 구현시에는 총 2^{16}개의 데이타베이스와 서버 및 워크스테이션 노드로 구성된 분산 데이타베이스를 생성할 수 있다. 하지만 이러한 구성방식은 실제로는 사용자의 운영체제와 하드웨어에 의해 제한된다.

VERSANT 환경에서 서버는 하나 혹은 그 이상의 데이타베이스 여러 사용자에게 동시에 접근할 수 있도록 서버 프로세스가 동작하는 컴퓨터를 뜻한다. 특정 그룹의 모든 사용자가 접근할 수 있는 데이타베이스를 그룹 데이타베이스라고 하며, 모든 사용자가 접근할 수 있는 그룹 데이타베이스를 유니버설 데이타베이스라고 한다.

VERSANT 환경에서 클라이언트는 지속형 데이타베이스 작업공간에 접근할 수 있는 응용 프로세스로써 다른 클라이언트와 함께 다중 그룹 데이타베이스를 동시에 접근할 수 있다.

클라이언트 프로세스와 서버 프로세스라는 용어는 VERSANT 환경에서 특정 하드웨어를 지칭하는 것이 아니라 그 역할에 따라 정해진다. 예를 들어, 자체 디스크를 갖는 워크스테이션은 클라이언트 프로세스를 수행시킬 수도 있고, 시스템 화일을 보유하며 개인 혹은 그룹 데이타베이스를 갖고 다른 워크스테이션에 대해 서버로써 동작할 수도 있다.

많은 사용자가 접근할 수 있는 그룹 데이타베이스 이외에, 한 사용자가 사용하는 작업공간인 개인 데이타베이스를 생성할 수 있다. 개인 데이타베이스는 그룹 데이타베이스에서 체크 아웃한 객체들을 대상으로 그룹 데이타베이스와는 개별적으로 일정 기간동안 그 객체에 대한 작업을 할 수 있다.

4.2.2 VERSANT Manager

"VERSANT Manager"는 클래스들을 관리하고, 질의어와 데이타베이스 갱신을 다루며, 객체 캐쉬와 장기 트랜잭션등을 지원한다. 언어 인터페이스를 사용하여 VERSANT Manager와 통신하는 응용 프로그램은 장기 트랜잭션의 시작이나 변경의 완료등과 같은 연산을 수행한다. VERSANT Manager는 또한 보조기억장치에 저장된 객체를 관리하는 VERSANT Server들과 통신한다.

VERSANT Manager는 기능적으로 여러개의 보조적인 소프트웨어 모듈로 구성되어 있다. VERSANT Schema Manager라는 모듈은 클래스를 정의하고 관리한다. VERSANT Query Processor는 특정 객체에 대한 질의 기능과 객체 클래스에 대한 반복적 질의기능을 지원한다. 다른 모듈들은 객체의 캐쉬, 세션, 장기 트랜잭션, 분산 트랜잭션의 지원등과 같은 기능을 제공한다.

4.2.3 VERSANT Server

"VERSANT Server"라는 용어는 소프트웨어 모듈을 지칭하는 것으로 VERSANT Server 소프트웨어가 수행되는 컴퓨터인 "서버"를 지칭하는 것은 아니다. VERSANT Server는 VERSANT ODBMS의 기저 레벨로 운영체제와 연결되어 데이타베이스에 데이타를 저장하고 검색하며, VERSANT Manager와 통신한다.

VERSANT Server는 단기 트랜잭션을 정의하고, 동시성 제어를 위한 객체에 대한 로크 및 회복을 위한 로깅을 제공한다. 객체는 VERSANT Server 레벨에서 고정된 갯수의 필드를 갖는 단순한 물리적 데이타 구조이다. 각 VERSANT Server는 하나의 데이타베이스에 접근한다. 따라서, 사용자가 클라이언트 응용 세션을 시작하면 VERSANT Manager와 VERSANT Server를 자동적으로 사용하게 되는 것이다.

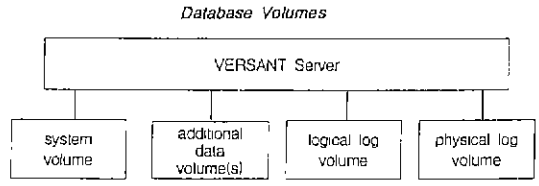
4.2.4 내부 통신

한 컴퓨터에서 VERSANT Manager와 VERSANT Server는 하나 혹은 두개의 프로세스로 운영된다. VERSANT Manager와 VERSANT Server가 단일 프로세스로 운영되면 프로세스간 통신 부담이 제거되므로 성능이 향상된다. 어떤 응용이 단일 프로세스로 운영된다 해도 네트워크 계층을 통한 내부 통신이 발생한다. 네트워크 계층은 메시지를 적절한 네트워크 프로토콜로 변환한다. 이러한 형태의 내부 통신때문에 VERSANT Manager는 데이타베이스가 분산되어 있다하더라도 네트워크상의 모듈과 통신할 수 있으며, 따라서 각 응용은 서로 다른 컴퓨터 상에 위치한 프로세스를 수행할 수 있다.

VERSANT Server와 운영체제 사이에는 각 하드웨어 플랫폼에 특정적인 Virtual System Layer가 있다. Virtual System Layer는 각 하드웨어들 사이에 이식성을 제공한다. 다른 하드웨어에 이식을 위하여 사용자는 VERSANT의 기초 데이타 타입이나 Class Library 데이타 타입을 사용해서 각 플랫폼에서 적절히 재 컴파일 해야 한다.

4.2.5 저장 구조

하나의 데이타베이스는 하나 이상의 볼륨으로 구성할 수 있으며, 디스크에 저장된다. 각 볼륨은 화일이나 raw device가 될 수 있다. 한 데이타베이스에 대한 시스템 볼륨은 데이타베이스를 생성하는 초기 과정의 일부분으로 자동적으로 만들어 지며, 클래스에 대한 기술과 객체 인스턴스를 저장하는데 사용된다. 데이타 볼륨의 추가는 "add volume" 유틸리티를 이용해 데이타베이스 볼륨을 확장할 수 있다. 트랜잭션의 수행은 논리적 로그 볼륨과 물리적 로그 볼륨에 기록되며, 논리적 및 물리적 로그 볼륨들은 데이타베이스가 만들어 질 때 생성된다.



응용 프로그램과 VERSANT 라이브러리들은 화일로써 저장된다. 클래스와 응용 프로그램을 생성하려면 먼저 적절한 VERSANT의 .h 화일들을 포함한 프로그램을 작성하고, 그 프로그램

을 컴파일한 후 적절한 VERSANT의 .a 라이브러리 화일들과 링크시킨다. C++ 인터페이스를 이용할 때 클래스 정보를 데이타베이스에 저장하기 위해 스키마 변경 유틸리티인 sch2db를 이용하여야 한다.

VERSANT Manager는 객체 캐쉬를 관리한다. 응용 프로그램에서 어떤 객체를 요구하면 VERSANT Manager는 그 객체가 이미 객체 캐쉬에 있는지 여부를 살펴보고, 만약 있다면 VERSANT Manager는 응용 프로그램에게 그 객체를 전송한다. 그러나 없다면, VERSANT Server에게 그 객체를 객체 캐쉬에 적재하도록 요청한다. VERSANT Server는 데이타베이스 저장장치와 서버 페이지 캐쉬를 관리한다. 메모리내의 객체를 관리하기 위해 VERSANT Manager는 Object Cache Table이라는 해쉬 테이블을 사용하며, 이 테이블은 Logical Object Identifiers와 메모리 포인터, 객체의 상태를 나타내는 정보등을 포함하고 있다. 해쉬 테이블의 크기는 가용한 메모리 양에 의해 제한되며, 활성화된 각 데이타베이스 세션마다 하나의 Object Cache Table이 존재한다.

캐쉬 메모리가 모두 소모되었을때 객체의 swapping을 위해 Object Cache Table은 Cached Object Descriptor를 생성하여 관리한다. 응용 프로그램은 객체의 실질적인 위치에 대한 포인터를 사용하지 않고 Cached Object Descriptor에 대한 포인터를 사용한다. 이처럼 중간 단계를 갖는 간접적인 포인팅 방법은 VERSANT Manager가 필요에 따라 메모리로부터 객체를 swap in/out 할 수 있도록 해준다.

한 세션에서 어느 한순간 단 하나의 활성화된 단기 트랜잭션만 있다. 활성화된 그 트랜잭션만이 Object Cache Table에 접근할 수 있다. Object Cache Table의 각 엔트리는 실제 객체를 가리키고 있는 Cached Object Descriptor를 포인팅하고 있다. 만약 그 메모리 포인터가 널이면 그 객체는 이미 swap out 된 것이다. Cached Object Descriptor의 갯수는 가용한 메모리 크기에 의해 제한된다. 트랜잭션이 완료되면 객체 캐쉬내의 수정된 객체나 새로운 객체가 VERSANT Server로 보내져 변경사항이 기록된다.

4.2.6 VERSANT ODBMS 특징 기존 DBMS의 기능을 포함한 기능성

VERSANT는 데이타베이스 관리 시스템으로 기존의 상용 DBMS등에서 기대되는 것과 동일한 특징들을 제공한다. VERSANT는 데이타의 지속적인 저장(persistent storage), 다중 사용자 지원, 데이타 무결성, 동시성 제어, 백업 및 회복, 분산 트랜잭션과 2단계 완료등을 포함하여 투명한 분산 데이타베이스를 구축하는데 필요한 핵심적인 DBMS 기능을 제공한다.

전통적인 DBMS들처럼 VERSANT는 통합 응용 개발 도구들을 지원하며, 다양한 플랫폼에서 수행되고, 여러 프로그래밍 언어에서 접근 가능하다. VERSANT의 플랫폼으로는 Sun, IBM, HP, NCR, DEC, Silicon Graphics, Sequent, NeXT, Intergraph 및 OS/2에서 동작하는 IBM/PC 및 그 호환기종들이 있다. VERSANT Language Interface로는 C, C++, Smalltalk 및 Object SQL 등이 있으며, 또한 VERSANT는 여러 유형의 third party 응용개발 제품 및 CASE 도구등과 결합된다.

그렇지만 전통적인 DBMS와 다르게, VERSANT는 데이타의 객체지향 모델에 기반을 두고 있으므로 객체지향 모델과 VERSANT의 독특한 구현 방법에 따른 여러 잇점을 제공한다. 이들 잇점들에는 복잡한 데이타를 모델링 할 수 있는 능력, 고도의 코드 재이용성, scalable distribution 및 응용 개발의 생산성 향상등이 있다. 따라서 이들 장점들은 응용 개발 생명주기 동안 시간과 비용을 절약시켜주게 된다.

4.2.7 Scalable Distribution

VERSANT는 VERSANT Scalable Distributed Architecture(VSDA)라고 알려진 독특한 구조로 구현되었다. VSDA란 VERSANT를 이용하여 시스템 구성방식이나 네트워크등을 고객의 필요나 구성의 변경등에 따라 조정(scale)할 수 있다는 것을 뜻한다. VSDA에 의해 제공되는 세가지 주요 특징은 투명한 데이타 분산, 고도의 workgroup 기능, 그리고 동적인 스키마 관리와 확장 등이 있다.

① 투명한 데이타 분산:

VERSANT는 완전히 투명하게 분산된 ODBMS이다. 응용 프로그램들은 다중 노드상에서 여러개의 VERSANT ODBMS 서버에 동시에 연결할 수 있다. 이들 서버에 의해 관리되는 객체들을 그 저장 위치에 관계없이 투명하게 접근할 수 있다. 이러한 『위치 투명성*(location transparency)*』은 응용 프로그램들과 데이타의 저장 위치가 서로 독립적이다라는 것을 의미한다. 객체들은 다른 노드상에 있는 객체들을 투명하게 포함할 수 있다. 참조 국부성을 높이고 성능을 향상시키기 위하여 객체들은 한 노드에서 다른 노드로 재배치 될 수 있다. 다중 서버상에서 객체를 갱신하는 트랜잭션은 분산 데이타의 무결성을 만족시키기 위해 2단계 완료 프로토콜을 사용하여 자동적으로 완료된다. 투명 분산*(transparent distribution)*이란 *VERSANT* 데이타베이스를 쉽게 재구성할 수 있어서 계속 변화되는 비지니스 요구에 부응할 수 있다는 것을 의미한다.

② Workgroup 지원을 위한 향상된 기능:

VERSANT는 오늘날 네트워크에서 사용되는 workgroup 기능을 지원 한다. 응용 프로그램들은 『장기 트랜잭션』을 이용하여 공유 서버로부터 객체를 데스크탑 워크스테이션으로 『체크-아웃』하여 지역적 프로세싱을 한다. 이것은 각 개인이 객체*(*예를 들면, 사업 계획이나 공학 설계 다이어그램등*)*를 그들의 지역 워크스테이션으로 체크-아웃하여 네트워크로부터 분리시킨후 *(*즉, *portable computer)*, 객체를 변경하고 네트워크에 재접속 시킨후, 그 객체를 공유 데이타베이스에 다시 체크-인 시킬 수 있음을 의미한다. 이러한 것들은 네트워크나 서버가 고장나더라도 완전한 동시성 제어와 트랜잭션 관리의 지원을 받는다. *workgroup*의 더나은 기능을 지원하기 위해 *VERSANT*는 객체 버전 기법을 포함하여 서로 다른 그룹들이 한 객체의 서로 다른 버전상에서 동시에 작업할 수 있도록 한다. 작업 수행후 그들을 자동적으로 결합시킨다. 이를 위하여 *VERSANT*는 객체의 *derivation history*를 유지하고 있으며, 따라서 내장된 변경 제어 시스템을 제공한다. 그러므로 분산 *workgroup* 환경을 지원하는 응용 프로그램을 쉽게 만들수 있게 해준다.

③ 동적인 스키마 관리 및 Evolution:

VSDA의 또다른 특징은 데이타베이스 스키마를 동적으로 수정하고 진화(evolve)시킬 수 있다는 것이다. 이는 사용자 환경의 변경에 따라 사용자 응용의 구조 변환이 가능하다. 스키마의 변경(예, 새로운 속성을 클래스에 추가)에 따라 데이타베이스를 unload하고 reload해야 할 필요가 없다. 즉 VERSANT는 시스템 사용의 중단없이 수행중에 스키마를 진화 시킬 수 있다. VERSANT에서는 Schema Evolution을 『*lazy-update*』 알고리즘을 이용해 구현하였다. 이는 객체를 참조될 때 수정한다는 것으로, 스키마 변경에 따르는 부담을 시간을 두고 분산시킬 수 있다. *Lazy schema update*는 시스템 성능을 최적화하며 각 응용에 대해서 예측가능한 성능을 산출할 수 있다. *VERSANT*는 또한 실행 시간중의 클래스 생성과 스키마 접근을 할 수 있다. 이는 응용 개발을 보다 유연하게 하여 데이타 브라우저와 같은 동적인 응용을 만드는데 필요한 기능을 제공해 준다.

4.2.8 성능의 최적화

VERSANT는 다양한 응용분야에 대해 그 성능을 최적화하며, 고도의 동시성 제어와 균등한 클라이언트-서버 구조 및 동적인 디스크 관리등을 통하여 다양한 상황에서 성능을 극대화 시킬 수 있는 특징들을 포함하고 있다.

① 고도의 동시성 제어:VERSANT는 데이타 일치성을 보장하기 위하여 업계에서는 진보된 동시성 제어 기법을 제공하여 고도의 성능을 발휘한다. VERSANT의 동시성 제어 기법들은 대개 객체 단위의 로킹과 중첩 트랜잭션에 기반을 두고 있다. VERSANT의 객체단위 로킹은 페이지단위 시스템에서 발견되는 "false waits", 즉 서로 다른 객

체에서 작업중인 두 사용자가 우연히 서로의 작업을 방해하게 되는 현상을 제거함으로써 동시성을 극대화 시킬 수 있다. 객체 단위 록킹은 단기 트랜잭션을 사용하는 응용에서도 중요하지만 특히 분산 응용에서 오랜 시간동안, 즉 몇일 혹은 몇주 동안 로크를 보유해야 하는 응용에서 false wait를 피하고자 할때 절대적으로 중요하다. VERSANT에서는 중첩 트랜잭션을 이용해 새로운 형태의 클라이언트 내부 동시성 제어 기법을 제공한다. 이는 단일 워크스테이션 상에서 수행되는 다중 윈도우하의 상호 관련있는 프로세스들의 작업을 동기화 시킨다. 일반적인 『*flat transaction*』은 워크스테이션을 적절히 지원하지 못한다. 왜냐하면 이 트랜잭션은 각 윈도우를 서로 다른 사용자로 보기때문에 두개의 윈도우는 각각의 작업 - 심지어 데드록 상태일 경우에도-에 대해 대기 상태로 들어갈 수 있기 때문이다. 중첩 트랜잭션을 통해 *VERSANT*는 다중 윈도우 데이타베이스 응용을 가능케 하며, 사용자 스크린 상의 윈도우 계층구조를 모델링해주는 트랜잭션 계층구조를 제공한다. 따라서 하나의 클라이언트 워크스테이션 상에서의 여러 윈도우들이나, 네트워크 상에서의 여러 클라이언트 워크스테이션들 사이에서 필요로 하는 동시성 제어를 제공하는 그래픽 다중 윈도우 응용 프로그램을 쉽게 작성할 수 있게 해준다.

② 균등한 클라이언트-서버 구조:

VERSANT는 균등한 클라이언트-서버 구조로 구현되어 데이타베이스 작업이 클라이언트와 서버로 균등하게 분할된다. 예를 들어, object caching은 클라이언트상에서 구현되어 네트워크 트래픽을 감소시키고, 객체에 대한 "arm traversal" 속도를 크게 향상시킨다. 이와 반대로 질의 처리는 서버상에서 구현되어 오직 요청된 객체만을 서버에서 클라이언트로 전달함으로써 네트워크 트래픽을 감소시킨다. Page caching은 서버상에서 수행되어 자주 사용되는 페이지상에 있는 객체의 위치를 찾아내는데 따르는 디스크 I/O를 감소시킨다. DBMS 기능을 서버와 클라이언트로 분할 함으로써 VERSANT는 네트워크 트래픽을 줄이고 성능을 향상시키며 데스크 탑과 서버의 연산 능력을 충분히 활용할 수 있다.

③ 동적 디스크 관리:

지능적인 디스크 공간 관리는 특히 대형 객체와 복잡한 데이타가 있는 환경에서 데이타베이스 성능을 극대화 시키는데 아주 중요하다. VERSANT는 다른 ODBMS와는 달리 온라인 방식의 디스크 재사용과 자동적인 데이타 클러스터링등을 통한 지능적인 디스크 최적화를 실현한다. 이러한 특징은 참조 국부성을 향상시키고, I/O를 감소시키며 전반적인 성능을 향상시켜준다.

4.2.9 폭 넓은 형태의 응용 프로그램 개발 해결책

어떤 DBMS라도 그 최종 목적은 응용 프로그램의 개발을 가능케 하는 것이다. 이를 위해 VERSANT는 자체적인 통합 응용개발 toolkit과 다양한 third-party 응용개발 환경을 제공하고 있다. VERSANT C++ Application ToolSet은 일련의 응용 개발 도구들로 그래픽 객체지향 응용들을 쉽게 만들 수 있도록 해준다. VERSANT C++ Application ToolSet에는 다음과 같은 도구들이 있다.

① VERSANT Screen:

C++ 기반의 응용 개발 도구로 특히 그래픽 클라이언트-서버 환경에서 사용자 화면폼을 쉽게 만들수 있도록 설계되었다. VERSANT Screen은 그래픽 사용자 인터페이스(GUI) 화면 생성기일 뿐만 아니라, IC++이라 불리는 C++ 기반의 스크립트 언어를 포함하여 응용 개발을 더 빠르고 쉽게 할 수 있게 한다.

② VERSANT Report:

사용하기 편한 그래픽 보고서 작성기로 사용자들로 하여금 자신에게 맞는 보고서를 신속히 생성할 수 있게 한다. VERSANT Report는 "drag-and-drop" 인터페이스와 통합된 데이타베이스 클래스 브라우저를 그

특징으로 하며 사용자가 원하는 보고서를 "point and click" 방식으로 손쉽게 작성할 수 있게 한다.

③ VERSANT Object SQL:

SQL 기반의 데이타 관리 언어로 표준 SQL의 사용이 용이한 질의어 및 갱신 기능에 객체 기법(object technology)의 능력을 추가한 것이다. Object SQL은 대중적인 SQL 언어를 확장한 것이기 때문에 오늘날 수많은 SQL 프로그래머들에게 친숙한 인터페이스이다. 또한 C%% 언어에 내장시켜 사용할 수 있는 Embedded Object SQL이라는 제품도 있다.

Versant사에서 제공하는 도구들이나 Versant사의 third-party partner에서 제공하는 도구들을 사용하여 사용자들은 VERSANT ODBMS의 기능, 성능, 생산성등을 충분히 활용할 수 있는 객체지향 응용 프로그램을 쉽게 작성할 수 있다.

끝으로, VERSANT ODBMS는 확장성 및 충분한 분산기능과 고도의 성능을 자랑하는 구조, 또한 사용자의 생산성을 높여줄 수 있는 다양한 응용 개발도구들을 제공하고 있으며, 사용자가 그들의 환경에 대한 전반적인 제어가 가능하도록 설계되어, 객체 지향 기술에서 사용자들이 요구하는 유연한 응용 프로그램 개발이 가능하도록 해준다. 따라서 VERSANT ODBMS는 제조업, 네트워크 관리, CASE, 문서 관리등과 같은 응용분야의 요구사항을 만족시키는 적합한 DBMS이다.

현재 VERSANT는 Release 2.1까지 나와있으며, 조만간 Release 3을 발표할 계획이다. Release 3에서 이전 버젼에 비해 향상된 기능들은 다음과 같다.

◈온라인 백업 및 점층적 백업 기능 추가
◈데이타베이스의 보안성 강화
◈commit 알고리즘의 개선으로 commit와 checkpoint commit의 성능 개선
◈로깅에 드는 오베헤드를 줄임으로써 성능 개선

5. 맺음말

지금까지 객체지향 데이타베이스 시스템을 간단히 소개하고 그 장점들을 살펴보았다. 또한 객체지향 데이타베이스 시스템이 정착되고 널리 사용되기 위하여 필요한 것들에 대하여도 알아보았다. 그러나 이제 서서히 객체지향 데이타베이스 시스템은 기존의 데이타베이스 관리 시스템으로 해결하기 어려운 분야에서 그 응용분야를 개척하고 있으며, 많은 개발자들이 이를 요구하고 있는 실정이다. 또한 객체지향 데이타베이스 공동체에서도 그표준을 위하여 ISO/IEC JTC1 SC21 WG7. OMG, ANSI OODBTG 등에서 그 노력을 기울이고 있다. 국내에서도 학계뿐만아니라 연구소, 산업체에서 객체지향 데이타베이스 관리 시스템을 개발중인 것으로 알고 있다.

특히 본고에서는 국내에서 현재 판매되고 있는 두 시스템-UniSQL과 VERSANT-을 소개하였다. 이를 위하여 이를 취급하는 회사로 부터 옥고를 받았으며, 이를 위하여 수고해주신 분들께 다시 한번 감사를 드린다. 끝으로, 정보과학회지 제11권 3호에 이미 실렸던 "객체지향 데이타베이스 기술"에서 많은 부분을 발췌하고 다시 편집하였슴을 알린다.

이 윤 준

1977 서울대학교 계산통계학과 졸업
1979 한국과학기술원 전산학과 석사
1983 France, INPG-ENSI-MAG 박사
1983 ~1984 France, IMAG 연구원
1984 ~현재 한국과학기술원 전산학과 부교수
1989 MCC(미) 초빙 연구원
1990 CRIN(불) 객원 교수

관심 분야 : 데이타베이스 시스템, 정보검색, 실시간 데이타베이스 등