기조칼럼

객체관계형 데이터베이

오늘날의 데이터베이스 스키마(schema)와 애플리케이션들은 아주 복잡하다. 뿐만 아니라, 하나의 기업내에서 운용되는 각각의 애플리케이션들은 서로 다른 데이터베이스 스키마에 의존하고 있으며, 이들의 '연결(interoperation)'을 관리하는 업무는 기업의 전산 혹은 정보담당 부서의 몫이되고 있다. 문제는 통합이 어렵다는 것이다.

아주 단순한 주문 입력을 하나의 비즈니스 모델로 갖고있는 조직을 생각해보자. 이 경우 주문/출하/과금과 같은 정형화된 프로세스를 표준화하기 위해서는 복잡한 데이터 모델링 이슈부터 해결해야 한다.

고객의 유형을 정하는 데만도 직접 매장을 찾는 고객인지, 우편주문 고객인지, 아니면 심지어 충동 구매자인지 등과 같이 복잡할 수 있다. 주문 혹은 구매품에 대한 출하 방식 역시 우편발송인지, 당일 배달을 위한 익스프레스 서비스인지 등과 같이 구분될 수 있다. 이와 같은 구매(주문) 및 출하(발송) 정보들은 결국 중앙의 과금시스템으로 모여져야 한다.

강병제/한국오라클 대표이사

이같은 '연결'은 곧 서로 다른 애플리케이션들간의 그리고 서로 다른 데이터베이스 스키마간의 연결을 의미하는 것이므로 결국 개발 측면에서 비용을 유발시킨다. 이 경우 전체 시스템은 고객, 주문서, 송장 등과 같은 공통적인 데이터 모델을 갖는다.

기업내 데이터베이스 기반 애플리케이션의 최근 경향은 다양한 비정형 데이터를 활용하는데 모아지고 있다. 예를 들어 보험 애플리케이션의 경우, 고객 정보에는 이름, 주소, 법규 위반 기록, 자동차 정보 등이 해당될 수 있다.

이들 데이터를 갖고 답할 수 있는 질의로는 "이 고객의 위험 요소로는 무엇이 있는가?", 혹은 "출고 3년 이하의 차량을 소유한 고객은 몇 명이나 되는가?" 등이 있을 수 있다. 그러나, "시내에서 가장 사고가 많이 일어나는 교차로의 반경 1Km내에 거주하는 고객의 명단은?"과 같은 질의는 위의 데이터만으로는 만족시킬 수 없다.

이같은 질의를 애플리케이션에서 만족시키기 위해서는 GIS에서나 사용될 법한 소위 "Spatial Data"가 필요하다. 사고가 가장 많이 발생하는 교차로와 각 고객들의 위치, 거리 등과 같은 공간인식에 필요한 데이터 등이 요구된다. 이와 같은 비정형 데이터들은 관계형 데이터베이스내에 저장될 수 있으나, 실제 프로세싱은 데이터 저장만으로 해결되지는 않는다.

객체지향형과 객체관계형의 비교

순수 객체 지향형의 데이터베이스는 실제 주로 CAD/CAM이나 과학적인 용도의 애플리케이션 등에 국한되어 사용되어지고 있다. 이러한 종류의 애플리케이션은 '복잡한 구조를 갖는 데이터들을 얼마나 효과적으로 모델링하여 실제 얼마나 잘 디스플레이(display)할 수 있는지'가 '동시 사용자가 얼마나 지원되는지', '사용자의 수가 증가하면서 데이터 액세스를 위한 트랜잭션 컨트롤은 어떻게 해결하는지', 아니면 '클러스터 시스템은 지원하는지' 등과 같은 요구사항보다 훨씬 그 우선순위에서 앞서게 된다.

바꿔말해, 앞서 예시한 주문/출하/과금과 같은 업무를 지원하는 애플리케이션은 실제 데이터가 얼마나 잘 디스플레이되는가 보다는 얼마나 신뢰성이 있고, 가용성이 있으며, 확장성이 보장되는지가 더 중요하다고 할 수 있으며, 이러한 이유로 인해 아직까지는 관계형 데이터베이스 기반의 애플리케이션들이 기업 업무 애플리케이션의 주류를 이루고 있다.

객체관계형 데이터베이스의 유형

현재 주류를 이루고있는 관계형 데이터베이스 제공업체들은 객체기술을 효과적으로 활용할 수 있는 소위 관계-객체형 또는 객체-관계형 데이터베이스를 개발하고 확산시키기 위한 노력을 꾸준히 진행하고 있다.

이런 활동은 대체로 ▲기존의 객체관계형 데이터베이스 엔진을 관계형

스의 출현과 향후 전망

데이터베이스 엔진과 통합 ▲새로운 객체관계형 엔진을 새로 개발 ▲기존의 관계형 데이터베이스 엔진에 새로운 기능을 추가하여 발전 등 3가지 형태로 구분된다.

이미 존재하던 두 가지 제품을 하나로 통합하는 방식은 우선 사용자들에게 투명성을 제공할 수 있도록 두 개의 서버간에 게이트웨이를 제공하는 것으로 시작하기 때문에 다른 방식에 비해 비교적 단기간에 솔루션을 소개할 수 있다는 장점을 지닌다.

그러나 이 방법은 종종 자동차와 보트를 하나로 합치는 것으로 비유되곤 한다. 다시 말해 두가지의 장점을 하나로 합친 것으로 보여도 주어진 환경하에서 결국 둘 중의 한가지가 갖는 장점만을 활용하는데 그친다는 것이다. 두 제품이 지니는 장점을 공히 활용하면서 만족할 만한 성능을 기대하는 데는 최소한 수 년이 걸리는 것으로 전문가들은 지적한다.

전혀 새로운 객체관계형 엔진을 새로 개발하는 방식은 어감면에서 가장 나은 솔루션처럼 들린다. 새로 개발되는 제품은 그 성능이나 기능이 시장에서 인정받는 데에만 최소한 몇 년이 걸리게 되고 이로 인해 실제 고객이 이를 선택하기에는 상당한 용기가 필요하게 된다.

관계형 데이터베이스 서버를 발전시켜 나가는 방식은 위의 두 가지 방식에 비해 다음과 같은 이점을 지닌다. 첫째, 십수년간 입증된 또 확산된 - 그래서 어찌보면 익숙한 - 관계형 데이터베이스 기술을 지렛대를 사용하듯이 이용할 수 있다. 둘째, 현재 전 세계적으로 가장 많이 사용되는 관계형 데이터베이스 기반의 애플리케이션에 객체 기술로의 전이를 위한 "통로(Paths)"를 제공한다.

다시 말해 객체 지향 애플리케이션과 기존에 사용해오던 애플리케이션이 공존할 수 있다는 것이다. 뿐만 아니라, 관계형 데이터베이스가 제공하는 신뢰성, 확장성, 보안, 질의 최적화 등이 객체 기술이 도입되더라도 그대로 보장될 가능성이 가장 크다.

이같은 사실을 고려할 때, 객체관계형 데이터베이스의 선각자인 마이클 스톤브레이커가 세번째 방법을 가장 높게 평가하는 이유도 쉽게 이해할 수 있다. 이 방법의 문제는 오랜 시간이 소요된다는 것이다. 오라클의 경우 이 방식을 택하여 1993년 부터 지금까지 작업을 지속해오고 있다.

ORDB에 적합한 애플리케이션

복잡한 데이터(Complex Data)를 하나의 논리적인 단위로 표현하고자 하는 애플리케이션은 객체관계형 데이터베이스 시스템의 이상적인 후보라고 할 수 있다.

예를 들어, 웹 페이지들은 종종 이미지, 사운드, 비디오 등과 함께 사용자들이 입력할 수 있는 필드로 구성되어 진다. 이들 각각의 웹 페이지들은 데이터베이스 웹 서버 상에서 하나의 객체, 즉 오브젝트로 표현될 수 있고, 다중 티어(Tier) 시스템에서 애플리케이션 서버는 이들 웹 페이지 오브젝트를 요청하여 클라이언트인 웹 브라우저 애플리케이션에서 사용되도록 HTML 코드로 변환해준다. 웹 상에서 주문 입력이 가능한 애플리케이션의 경우 사용자에 의해 입력된 데이터는 출하 및 과금을 위한 애플리케이션에도 제공되어야 하고, 이외에도 의사 결정을 위한 데이터 분석에까지도 활용될 수 있어야 할 것이다.

객체관계형 DB의 요구 사항

객체관계형 데이터베이스(ORDBMS)는 관계형의 데이터 저장 방식과 객체 모델링이 가장 효과적으로 결합될 수 있어야 한다.

ORDBMS는 사용자가 정의한 데이터 타입을 유연하게 사용할 수 있어야 하고, 관계형 데이터베이스에 비해 멀티미디어나 대형 오브젝트(Large Data Object)를 더욱 잘 지원할 수 있어야 하며, SQL과 같은 DBMS 표준은 물론 CORBA나DCOM과 같은 객체 표준을 준수해야 한다. 게다가 기업내의 주요 업무를 지원하는 애플리케이션을 위해 질의 성능이나 확장성, 보안 등의 측면에서 현재 RDBMS의 수준을 능가할 수 있어야 한다.

관계형 데이터베이스에서 사용할 수 있는 데이터 타입은 Character, Number 그리고 Date와 같이 한정되어 있다. 이들 데이터 타입에 주어지는 오퍼레이션 역시 데이터베이스 엔진내에 이미 고칠 수 없도록 코딩되어 있다.

객체관계형 데이터베이스는 사용

자들이 정의하는 데이터 타입과 이들 데이터 타입상에서 실행되는 오퍼레이션을 지원할 수 있어야 한다. 사용자가 정의한 데이터 타입은 시스템내에서 다른 데이터들과 아무런 문제없이 함께 사용되어야 한다.

예를 들어 사용자가 주문이라고 하는 데이터 타입을 Order라고 정의하는 경우, 이 타입의 속성(Attribute)은 관계형 테이블의 컬럼과 유사할 것이다. 메쏘드(Method)는 관계형 데이터베이스에서 저장 프로시져(Stored Procedure)를 정의할 때와 마찬가지 방법으로 정의되면 된다. (Order라는 데이터 타입의 Attribute에는 주문번호, 주문 고객명, 주문품, 수량 등이 정의될 수 있고, Method로는 Shipping, Holding, Cancel등이 정의될 수 있을 것이다.)

"Point"와 같은 데이터 타입은 더욱 복잡할 수 있는데, 이는 X축과 Y축의 좌표 개념으로 정의될 수 있다. 이같은 데이터 타입에 작동하는 함수로서는 거리를 산출하는 것이라든지, 주어진 반경내의 영역을 찾아내는 것 등이 가능할 것이다. 이럴 경우 "새로 개설한 매장으로부터 반경 2Km 안의 경쟁사 매장의 위치 및 갯수는?"과 같은 질의도 만족시킬 수 있다.

객체관계형 데이터베이스는 또한 오브젝트(객체)를 상세 정도에 따라 모델링할 수 있어야 하고, 필요시 더 복잡한 오브젝트를 만들어내기 위해 다른 오브젝트를 참조할 수 있어야 한다. 예를 들자면, 회사내에서 하나의 부서는 직원, 자산, 관리자, 위치 등 몇 개의 오브젝트가 모여 구성된다. 따라서 ORDBMS내에서 "부서"를 모델링하기 위해서는 부서를 구성하는 소위 '컴포넌트 오브젝트(Component Object)'들의 특성을 잘 정의해야 한다.

시스템이 진정한 객체관계형이 되기 위해서는 데이터 타입의 확장들이 메쏘드에 동적으로 링크되어 서버 코드를 다시 코딩하는 일이 없도록 해주어야 하고, 클라이언트나 서버에서 선택적으로 활성화되고 실행될 수 있어야 하며, 적절한 보안/통제 기능이 제공되어져 서버 자체의 보안이 영향받아서는 안된다.

멀티미디어와 같은 비정형 데이터는 기존 관계형 데이터(행과 열로 표현되어 '스칼라 데이터'라고도 함) 기반 애플리케이션의 유용성을 높일 수 있지만, 오브젝트 저장을 위해 스페이스 요구량이 급격히 증가된다. ORDBMS는 대형 오브젝트 (LOB: Large OBjects) 를 수준급의 성능으로 처리할 수 있어야 한다. 또한 데이터베이스는 클라이언트 애플리케이션에서 커다란 객체 가운데 원하는 부분만을 끌어내는 'Piece-wise' 액세스를 지원해야 한다.

개발측면의 개선

객체 도입의 필요성 중에 개발자들이 가장 공감을 느낄 수 있는 것은 아마도 오브젝트 컴포넌트(Component)의 재사용성이 아닌가 싶다. 오브젝트 컴포넌트의 재사용성은 복잡한 비즈니스를 모델링하는데 더없는 장점으로 떠오르고 있다. Visual C++과 Java와 같은 객체 지향 개발 환경은 컴포넌트 기반의 소프트웨어 개발에 촉진제 역할을 한다.

한편 애플리케이션의 개발은 객체지향인데 반하여, 그 애플리케이션을 통해 액세스되는 데이터는 관계형 데이터베이스내에 주로 저장되고 있는 것이 현실이다. 여기서 해결해야 하는 매핑(Mapping)은 새로운 부담이 되고 있으며, ORDBMS는 이같은 "Mismatch"를 줄여준다.

1~2년내 양기술 접목 최적화

오늘날 기업의 미션 크리티컬 애플리케이션의 기반을 이루는 관계형 데이터베이스와 객체 기술의 접목은 피할 수 없는 현실로 다가서고 있다.

객체관계형 데이터베이스는 업체별로 다른 시도를 통해 소개되고 있으나 중요한 사실은 관계형 데이터베이스의 장점과 객체 기술의 장점만을 제공하는 "이상적인" 데이터베이스를 찾기는 아직까지는 쉽지 않다는 것이다. 비정형 데이터 처리는 객체 기술이, 질의 성능의 최적화 및 확장성은 관계형 기술이 우월하므로 업체별 개발 방식은 그 성능이나 활용면에서 각각 다를 수 있다는 점을 지적하고 싶지만 향후 1~2년내에 양기술의 접목은 최적화 될 것으로 기대된다.

중요한 것은 객체 기술을 효과적으로 도입함으로써 앞으로의 수고나 비용을 줄이는 것도 중요하지만 기존의 투자분을 보호하는 것도 이에 못지 않게 중요하다는 사실이다. DC