후방-질의 공존의 스키마 변화

나영국[0]
컴퓨터공학과, 한경대학교
ygra[0]@kiss.or.kr

Backward-Query Compatible Schema Changes

Young-Gook Ra[0]
Dept. of Computer Engineering, Hankyong National University

요 약

변화하는 사업 환경에 적절히 대응하기 위해 온라인 데이터베이스의 스키마를 변화할 수 있는 능력은 필수적이다. 그러나 데이터베이스 스키마 변화는 그 스키마에 대해 쓰여진 기존의 응용 프로그램에 의해 제약을 받아왔다. 현재의 스키마를 변경하는 동시에 기존의 스키마를 보존하여 그 스키마에 쓰여진 질의 프로그램이 스키마 변화 이 후에도 지속적으로 동작할 수 있게 하여 기존 프로그램 재작성의 부담을 덜어준다. 이 논문은 기존의 스키마를 뷰로써 유지함으로써 뷰 활용 후방-질의 공존 (backward-query compatible) 스키마 변화의 알고리즘과 시스템을 제안한다.

1. 서 론

데이터베이스 디자이너는 모델링 대상의 환경을 데이터베이스에 정확히 반영하는 스키마를 만들어 내고자 한다. 그 스키마는 대상 환경이 변화하여도 여전히 유효할 정도로 안정적인 것으로 가정한다. 그러나 현실적으로 스키마는 데이터베이스 디자이너가 기대하는 정도로 안정적이지 못하다. 데이터 모델의 불안정성은 모델되는 환경의 변화, 요구분석시 실수, 모델링 자체의 실수 등에 연유한다. Marche[1]는 데이터 모델의 안정성은 근거없는 주장이며 소프트웨어 프로그램에 비해 상대적으로 안정적이라는 의미라고 얘기한다. 그는 경험적인 연구를 통하여 50% 이상의 주키가 유지되는 테이블은 2년 후에 1/7에 불과하다고 얘기한다.

스키마 변화는 테이블을 더하고/빼고, 속성을 더하고/빼고, 복잡한 테이블을 여러 테이블로 분해하는 등이다. 이러한 변화들이 전문 소프트웨어 도구에 의해 행해지면 특히 그 도구들이 데이터 무결성을 확인하고 데이터를 변화하는 스키마에 따라 변경하는 등의 기능을 가지고 있다면 더욱 데이터베이스 관리자의 일을 덜어준다. 사실 많은 도구들이 관계형 모델을 위해서[2], 네트워크 모델을 위해서[3], 객체지향 모델을[4] 위해서 제안되었다.

비록 풍부한 스키마 변화 연산이 지원된다 해도 사용자는 스키마에 쓰여진 기존의 응용 프로그램에 대한 스키마 변화의 영향에 때문에 어려움을 겪는다. 데이터 모델에 상관없이 응용 프로그램과 질의의 재작성이 스키마 변화의 병목이라고 보고되곤 한다. Shneiderman[5]은 건강 보험 응용에서 600개의 PL/I-IMS 응용 프로그램의 각각을 전환시키고 테스트하는데 2개월의 작업이 필요했다고 보고했다.

그림 1은 스키마 변화에 기인한 프로그램 재작성 문제를 설명한다. Prog1이 그림 1의 왼쪽의 데이터베이스에 동작되고 prog2는 개발되면서 현 스키마로서 만족시킬 수 없는 새로운 데이터 서비스를 요구한다. 그 서비스를 만족시키기 위해 그림 1의 오른쪽의 스키마로 변경한다. 그러나 기존의 응용 프로그램 prog1은 그림 1에서 보듯이 변경된 스키마에서 동작하지 않을 수 있다.

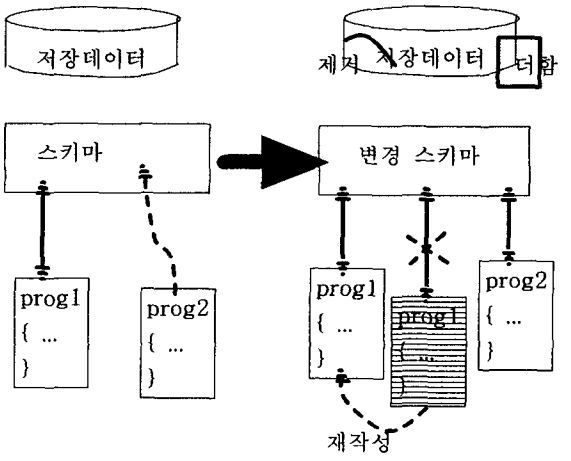

그림 1. 현 스키마 변화 도구의 문제점

옛 스키마가 보존되어 질의를 스키마 변화 전처럼 허락하고 관련 데이터는 항상 최신으로 유지된다면 최소한

질의 프로그램의 재작성을 피할 수 있어 프로그램 재작성의 노력을 덜어준다. 이 성질을 후방-질의 공존(backward-query compatibility)이라 한다.

그림 2는 후방-질의 공존 스키마 변화를 위한 우리의 접근 방식을 보여준다. 스키마 변화는 베이스 스키마 BS에 적용되고 BS는 BS'으로 직접 변화한다. 그 변화 후에 새로운 응용 프로그램 Prog2는 BS가 아니고 BS'에 대해 개발된다. 기존의 질의 프로그램 query1과 갱신 프로그램 prog1은 기대하는 데이터 뷰, BS, 가 존재하지 않기 때문에 더 이상 동작하지 않을 수 있다. 이 문제에 대한 우리는 기존의 스키마를 뷰 스키마로 구현하여 질의 프로그램 query1에 제공함으로써 query1의 지속적인 동작이 보장된다. 그러나 갱신이 포함된 프로그램 prog1은 변경 스키마 BS'에 대해서 prog'으로 재작성 되어야 한다.

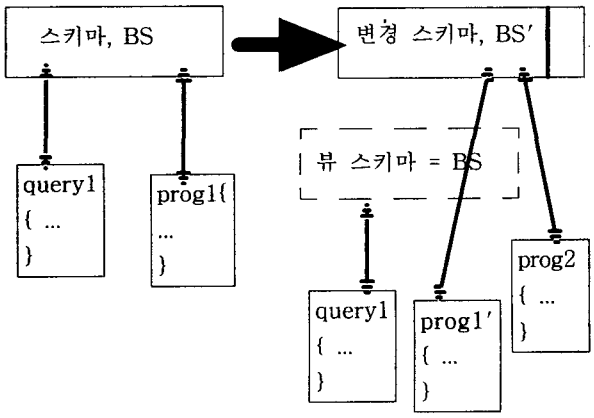

그림 2. 후방-질의 공존 (backward-query compatibility)을 위한 우리의 접근 방식.

기존 프로그램 지원을 위한 데이터베이스 뷰의 다른 사용이 그림 3에 보여진다. 사용자가 스키마 변화 연산을 적용하면 스키마를 직접 변경하지 않고 목표 스키마를 뷰로써 발생시킨다.

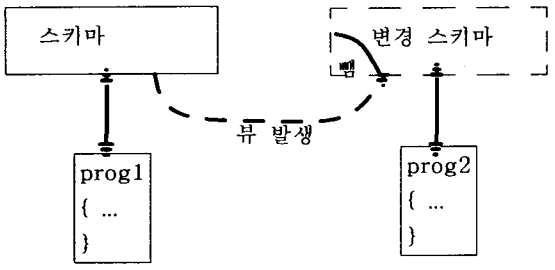

그림 3. 목표 스키마를 뷰로 발생.

우리가 제안하는 시스템의 기본 아이디어는 다음과 같다: 용량 증가 (capacity-augmenting)가 아닌 스키마 변화 연산에는 그림 3에서 보듯이 목표 스키마를 뷰로써 발생시키고 용량 증가 스키마 변화는 그림 2에서 보듯이 베이스 스키마를 직접 목표 스키마로 변화시키고 원래 스키마를 뷰로써 복구시킨다.

본 논문은 2장에서 스키마 변화 연산자들을 설명한다. 3장에서 뷰 갱신 가능성을 언급하며, 4장에서 관련 연구를 비교하며 5장에서 결론을 설명한다.

2. 스키마 변화 연산

2.1. 속성 더하기

연산자 **add-attribute** attr **to** tbl 은 테이블 tbl에 속성 attr을 더한다. 예를 들어 'add-attribute 고객주소 to 주문' 연산은 그림 4에서 처럼 왼쪽 테이블에 직접 '고객주소' 속성을 더하여 중앙 '주문' 테이블로 변화시킨다. 변화 이전의 '주문' 테이블은 변화 이후의 '주문' 테이블에 뷰 연산 'create view select 번호, 주문일, 고개ID, 고객이름 from 주문'을 적용하여 발생시킨다.

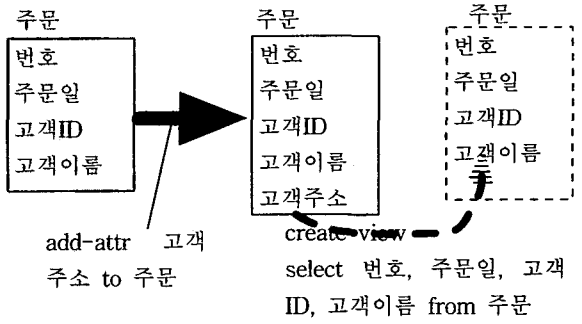

그림 4. 속성 더하기 연산

2.2. 분해

연산자 **decompose** decompTable **from** sourceTable **of** attributes **withPKs** PKAttributes 는 속성들 attributes를 가지고 PKAttributes를 주키로 하는 테이블 decompTable을 생성하고 sourceTable에서 PKAttributes를 제외한 attributes 속성들을 제거한다. 예를 들어 그림 5에서 보듯이 'decompose 고객 from 주문 of 고객ID, 고객이름, 고객주소 withPKs 고개ID'는 '고객ID', '고객이름', '고객주소' 속성을 가지고 '고객ID'를 주키로 하는 '고객' 테이블을 생성하고 '고객ID'를 제외한 '고객' 속성들은 주문 테이블에서 제거된다. 주문 테이블의 '고객ID'는 남아서 '고객' 테이블의 주키 '고객ID'를 가리키는 외래키 역할을 한다 (그림6에서 얇은 선 화살표로 외래키를 표시).

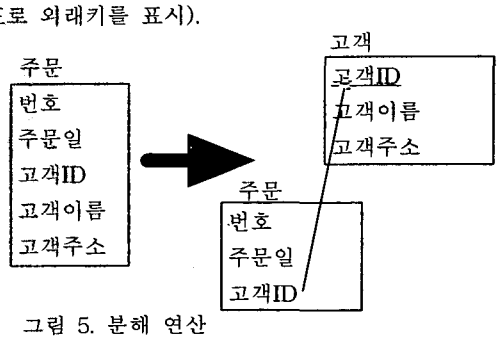

그림 5. 분해 연산

분해 연산은 대상 테이블을 직접적으로 변화시키므로 변화 이전의 대상 테이블을 뷰로써 복원시킬 필요가 있다. 그림 6에서 분해 이전의 '주문' 테이블은 '고객' 테이블과 분해 이후의 '고객' 테이블의 조인 뷰 테이블 '주문'을 생성함으로써 복원되었다는 것을 알 수 있다.

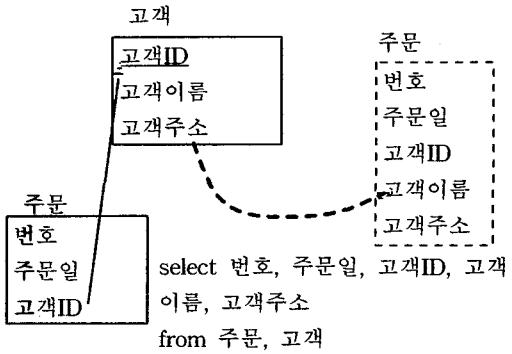

그림 6. 분해 연산의 옛 스키마 복원

2.3. 나머지 후방 공존의 스키마 변화 연산들

연산자 **delete-attribute** attr **from** tbl은 테이블 tbl에서 속성 attr을 제거한다. 연산자 **create-table** tbl **with** a_1, $type_1$, ..., a_n $type_n$은 a_1, .., a_n 속성들로 이루어진, 각각 $type_1$, ..., $type_n$ 타입을 가진, 테이블 tbl을 생성한다. 연산자 **drop-table** tbl은 테이블 tbl을 제거한다. 연산자 **merge** table1 **and** table2 **basedOn** attributes는 공통 속성을 제외한 table2의 속성들을 table1에 더해진다. 새 테이블, 이름이 table1과 같음, 의 인스턴스들은 table1과 table2의 인스턴스들을 공통 속성 attributes로 조인하여 얻어진다. 연산자 **change-pk** table **from** attributes1 **to** attributes2는 table의 주키를 attributes 1에서 attributes2로 바꾼다. 연산자 **add-fk** attribute1 **of** table1 **references** attribute2 **of** table2는 attribute1 속성을, table2 테이블의 attribute2 속성을 가리키는 외래키로 만든다. 연산자 **del-fk** attribute1 **of** table1 **references** attribute2 **of** table2는 table1 테이블의 attribute1 속성이 table2 테이블의 attribute2 속성을 가리키는 외래키 제약조건을 제거한다.

3. 의미 보전 (semantic preserving) 갱신 가능 뷰

우리의 접근 방식에서는 뷰는 광범위하게 사용된다. 뷰는 목표 스키마를 구현하거나 원래 스키마를 복구한다. 연산자 **add-attribute**와 **delete-attribute**에서 사용되는 뷰들은 갱신가능이다. 왜냐하면 이들이 단지 하나의 테이블에서 생성되고 where 구문이 없어 기본 테이블에서 튜플은 뷰의 삽입된 튜플을 반드시 생성할 수 있기 때문이다. 그러나 **decompose** 혹은 **merge**에서 사용되는 뷰는 한 개이상의 테이블로부터 정의되기 때문에 갱신이 되지 않는다.

4. 관련 연구

스키마 변화 도구의 첫 종류는[5] '직접 스키마 변화 시스템' 이다. 이는 단순하고, 성숙하며 대부분의 DBMS에서 지원된다. 그러나 이들은 응용 프로그램과 질의 재작성 문제에 제약을 받는다. 두 번째 종류는 '스키마 버전 시스템'으로[7] 옛 스키마 버전의 데이터가 최신화되지 않으므로 역시 재작성 노력을 덜어주지 못한다. 세 번째 종류는 '프로시저 방식 시스템'으로[8] 타입 불일치를 해결하는 프로시저를 제공하게 하는 방식이며 이는 사용자에 부담을 준다. 네 번째 종류는 '뷰 방식 시스템'으로 잘 정의된 뷰를 사용함으로써 구현시 숨어 있는 문제가 없다.

5.결론

전통적인 스키마 변화 도구의 약점인 프로그램 재작성 문제를 완하하기 위해 뷰로 옛 스키마를 유지하는 해결책이 제시되었다. 이 방식을 관계형 스키마 변화에 적용하여 후방-질의 공존 (backward-query compatible) 의 스키마 변화 연산자들을 소개하였다. 완벽한 공존성 (compatibility)을 위해서는 관계형 뷰의 갱신에 대한 추가 연구가 필요하다.

참고문헌

[1] S. Marche, "Measuring the Stability of data models", European Journal of Information Systems, vol. 2, no. 1, pp. 37-47, 1993.

[2] V. M. Markowitz and J. A. Markowsky, "Incremental restructuring of relational schemas", in IEEE International Conference on Data Engineering, pp276-284, 1988

[3] G. Thomas and B. Shneiderman, "Automatic database system conversion: A transformation language approach to sub-schema implementation", in IEEE Computer Software and Applications Conference, pp80-88, 1980.

[4] J. Banerjee, W. Kim, H.J. Kim, and H.F. Korth, "Semantics and implementation of schema evolution in object-oriented databases", SIGMOD, pp311-322, 1987.

[5] B. Shneiderman and G. Thomas, "An architecture for automatic relational database system conversion", ACM Transactions on Database Systems, vol. 7, no. 2, pp235-257, June 1982.

[6] V. Vidal and M. Winslett, "Preserving update semantics in schema integration", in International Conference on Information and Knowledge Management, pp.263-271, 1994.

[7] W. Kim and H. Chou, "Versions of schema for OODBs", in Proc. 14th VLDB, pp.148-159, 1988.

[8] A. H. Skarra and S. B. Zdonik, "The management of changing types in an object-oriented databases", in Proc. 1st OOPSLA, pp.483-494, 1986.